PROYECTO HARDWARE

Realizado por:
Adrián Ortega Escalona
adrian.ortega@edu.upct.es
Javier Verdú Sánchez
javier.verdu2@edu.upct.es

ÍNDICE

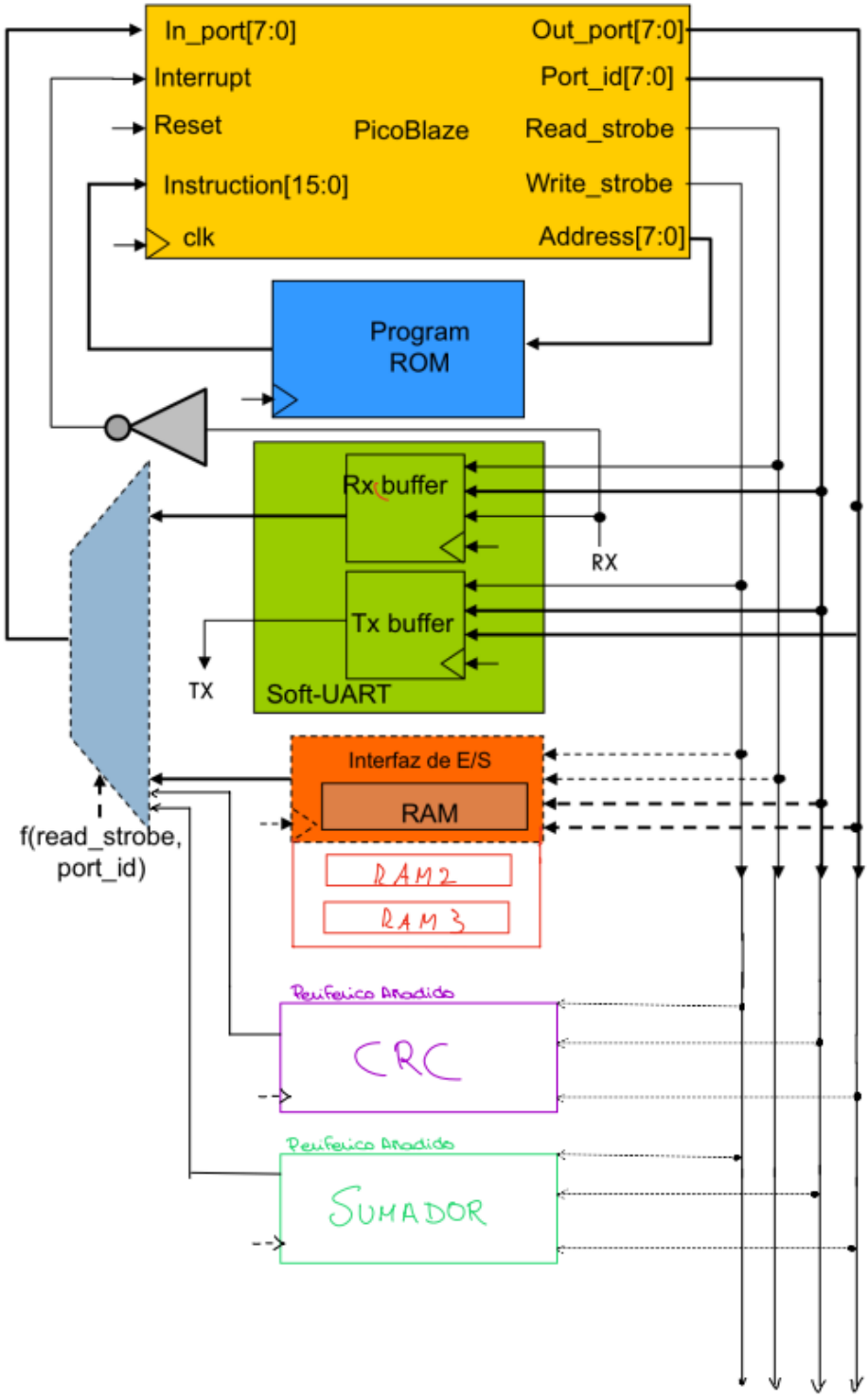
INTRODUCCIÓN

Para la realización de esta práctica hemos partido del proyecto que se completó en las sesiones de prácticas de la asignatura, de tal forma que el nombre de la carpeta del proyecto es “P2c_Picoblaze_Helloworld_RAM_INT_FLIP”.

En esta memoria vamos a explicar las modificaciones realizadas sobre el proyecto anterior. Este proyecto se compone de:

- Un nuevo Periférico llamado “**CRC**” conectado a PicoBlaze.
- Un nuevo Periférico llamado “**Sumador**” conectado al PicoBlaze.
- Modificación de la microarquitectura del PicoBlaze (introducción de una nueva instrucción, llamada “**CRC**”).
- Dos Nuevas RAMs.

TOPLEVEL



Para la realización de esta parte una vez hechos y comprobados los periféricos los hemos añadido como componentes a la arquitectura del Picoblaze además de añadir 3 memorias RAM para facilitar el proceso de Lectura de código ASCII para facilitar el código ensamblador y poder hacerlo más eficiente.

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.NUMERIC_STD.ALL;
4
5   entity toplevel is
6       Port (      port_id : out std_logic_vector(7 downto 0); --solo para depurar
7              write_strobe : out std_logic;                    --solo para depurar
8               read_strobe : out std_logic;                    --solo para depurar
9                  out_port : out std_logic_vector(7 downto 0); --solo para depurar
10                  in_port : out std_logic_vector(7 downto 0); --solo para depurar
11                    reset : in std_logic;
12                      clk : in std_logic;
13                       rx : in std_logic;
14                       tx : out std_logic;
15                       LED : out std_logic);     --led de comprobacion y reset
16  end toplevel ;
17
18  architecture behavioral of toplevel is
19
20  ----------------------------------------------------------------
21  -- declaracion del sumador
22  ----------------------------------------------------------------
23  component  sumador
24  Port (
25                data_out : out std_logic_vector(7 downto 0);
26               write_strobe : in std_logic;
27                out_port : in std_logic_vector(7 downto 0);
28                 port_id : in std_logic_vector(7 downto 0);
29                 reset : in std_logic;
30                 clk : in std_logic);
31
32  end component;
33  component crc
34  Port(
35                 crcOut : out std_logic_vector(7 downto 0);
36                 data: in std_logic_vector(7 downto 0);
37                 port_id : in std_logic_vector(7 downto 0);
38                 write_strobe : in std_logic;
39                  reset : in std_logic;
40                 clk : in std_logic);
41
42  end component;
43
44  ----------------------------------------------------------------
45  -- declaracion del picoblaze
46  ----------------------------------------------------------------
47    component picoblaze
48      Port (      address : out std_logic_vector(7 downto 0);
49             instruction : in std_logic_vector(15 downto 0);
50                 port_id : out std_logic_vector(7 downto 0);
51            write_strobe : out std_logic;
52                out_port : out std_logic_vector(7 downto 0);
53             read_strobe : out std_logic;
54                 in_port : in std_logic_vector(7 downto 0);
55               interrupt : in std_logic;
56                   reset : in std_logic;
57                     clk : in std_logic);
58      end component;
```

```
61   -- declaracion de la ROM de programa
62   ----------------------------------------------------------------
63     component programFinal
64       Port (      address : in std_logic_vector(7 downto 0);
65                         dout : out std_logic_vector(15 downto 0);
66                        clk : in std_logic);
67       end component;
68
69
70
71   ----------------------------------------------------------------
72   -- Signals usadas para conectar el picoblaze y el SUM
73   ----------------------------------------------------------------
74   signal sal_sumador :std_logic_vector(7 downto 0);
75   signal sal_crc:std_logic_vector(7 downto 0);
76   ----------------------------------------------------------------
77   -- Signals usadas para conectar el picoblaze y la ROM de programa
78   ----------------------------------------------------------------
79   signal     address : std_logic_vector(7 downto 0);
80   signal instruction : std_logic_vector(15 downto 0);
81
82   ----------------------------------------------------------------
83   -- Signals para debugging
84   ----------------------------------------------------------------
85   signal readstrobe: std_logic;
86   signal writestrobe: std_logic;
87   signal portid: std_logic_vector(7 downto 0);
88   signal inport: std_logic_vector(7 downto 0);
89   signal outport: std_logic_vector(7 downto 0);
90   signal picoint: std_logic;
91   signal data:std_logic_vector(7 downto 0);
92
93   type ram_type is array (0 to 63) of std_logic_vector (7 downto 0);
94   signal RAM : ram_type := (
95   x"0A", x"0D", x"2A", x"20", x"48", x"45", x"4C", x"4C",    -- Hellow world...
96   x"4F", x"20", x"49", x"27", x"4D", x"20", x"41", x"4C",
97   x"49", x"56", x"45", x"21", x"20", x"3A", x"2D", x"44",
98   x"20", x"2A", x"0A", x"0D", x"2A", x"20", x"50", x"52",
99   x"45", x"53", x"53", x"20", x"41", x"4E", x"59", x"20",
100  x"4B", x"45", x"59", x"20", x"54", x"4F", x"20", x"43",
101  x"4F", x"4E", x"54", x"49", x"4E", x"55", x"45", x"20",
102  x"2A", x"0A", x"0D", x"00", x"00", x"00", x"00", x"00" );
103
104  signal rxbuff_out,RAM_out: std_logic_vector(7 downto 0);
105
106  type ram_type2 is array (0 to 63) of std_logic_vector (7 downto 0);
107  signal RAM2 : ram_type2 := (
108  x"0A", x"31", x"20", x"4E", x"75", x"6D", x"65", x"72", --1 numero
109  x"6F", x"2C", x"0A", x"31", x"20", x"4E", x"75", x"6D",
110  x"65", x"72", x"6F", x"2C", x"20", x"56", x"61", x"6C",
111  x"6F", x"72", x"20", x"20", x"64", x"65", x"20", x"6C",
112  x"61", x"20", x"58", x"4F", x"52", x"0A", x"0D", x"32",
113  x"20", x"4E", x"75", x"6D", x"65", x"72", x"6F", x"2C",
114  x"20", x"56", x"61", x"6C", x"6F", x"72", x"20", x"42",
115  x"75", x"63", x"6C", x"65", x"0A", x"0D", x"00", x"00");
116
```

```
117   signal rxbuff_out2,RAM_out2: std_logic_vector(7 downto 0);
118
119   type ram_type3 is array (0 to 63) of std_logic_vector (7 downto 0);
120   signal RAM3 : ram_type3 := (
121   x"33", x"20", x"4E", x"75", x"6D", x"65", x"72", x"6F",  --3 Numero, Valor del CRC
122   x"2C", x"20", x"33", x"20", x"4E", x"75", x"6D", x"65",
123   x"72", x"6F", x"2C", x"20", x"56", x"61", x"6C", x"6F",
124   x"72", x"20", x"64", x"65", x"6C", x"20", x"43", x"52",
125   x"43", x"0A", x"0D", x"00", x"00", x"00", x"00", x"00",
126   x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
127   x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00",
128   x"00", x"00", x"00", x"00", x"00", x"00", x"00", x"00");
129
130
131   signal rxbuff_out3,RAM_out3: std_logic_vector(7 downto 0);
132   begin
133
134      sumadorInst : sumador
135      port map(      write_strobe => writestrobe,
136                     data_out => sal_sumador,
137                     out_port=>outport,
138                     reset => reset,
139                     port_id=>portid,
140                        clk => clk);
141
142         crcInst : crc
143      port map(      write_strobe => writestrobe,
144
145                     crcOut => sal_crc,
146                     data =>data,
147                     reset => reset,
148                     port_id=>portid,
149                        clk => clk);
150
151      LED <= reset;  -- para comprobar la programacion encendemos
152                     -- un led cada vez que reseteamos
153
154      read_strobe <= readstrobe;
155      write_strobe <= writestrobe;
156      port_id <= portid;
157      in_port <= inport;
158      out_port <= outport;
159      picoint <= NOT rx;
160      data<=outport;
161
162     processor: picoblaze
163       port map(      address => address,
164                  instruction => instruction,
165                      port_id => portid,
166                 write_strobe => writestrobe,
167                     out_port => outport,
168                  read_strobe => readstrobe,
169                      in_port => inport,
170                    interrupt => picoint,
171                        reset => reset,
172                          clk => clk);
173
174     program: programFinal
```

```
176
177       port map(     address => address,
178                       dout => instruction,
179                      clk => clk);
180
181      --registra el bit tx del puerto de salida, por si éste cambia
182
183      txbuff:process(reset, clk)
184      begin
185         if (reset='1') then
186            tx <= '1';
187         elsif rising_edge(clk) then
188            if (writestrobe = '1' and portid=x"FF") then
189               tx <= outport(0);
190            end if;
191         end if;
192      end process;
193
194      --añade 7ceros a rx para meterlos al puerto de entrada cuando se lea
195      rxbuff:process(reset, clk)
196      begin
197         if (reset='1') then
198            rxbuff_out <= (others=>'1');
199         elsif rising_edge(clk) then
200            if (readstrobe = '1' and portid =x"FF") then
201               rxbuff_out <= rx & "0000000";
202            end if;
203         end if;
204      end process;
205
206      -- Memoria RAM (escritura sincrona / lectura asincrona) INSTANCIA LA RAM
207      process (clk)
208      begin
209         if (clk'event and clk = '1') then
210            if (writestrobe = '1' and portid<x"40") then
211               RAM(to_integer(unsigned(portid))) <= outport;
212            end if;
213         end if;
214      end process;
215         RAM_out <= RAM(to_integer(unsigned(portid)));
216         -- Memoria RAM2 (escritura sincrona / lectura asincrona) INSTANCIA LA RAM
217
218         process (clk)
219      begin
220         if (clk'event and clk = '1') then
221            if (writestrobe = '1' and (portid>=x"40" and portid<x"80")) then
222               RAM2(to_integer(unsigned(portid))) <= outport;
223            end if;
224         end if;
225      end process;
226      RAM_out2 <= RAM2(to_integer(unsigned(portid)));
227
228      -- Memoria RAM3 (escritura sincrona / lectura asincrona) INSTANCIA LA RAM
229
230         process (clk)
231      begin
232         if (clk'event and clk = '1') then
```

toplevel_helloworld_RAM_int.vhd

```
233            if (writestrobe = '1' and (portid>=x"80" and portid<x"C0")) then
234               RAM3(to_integer(unsigned(portid))) <= outport;
235            end if;
236         end if;
237      end process;
238      RAM_out3 <= RAM3(to_integer(unsigned(portid)));
239
240   -- Multiplexor inport
241   inport <= RAM_out when (readstrobe = '1' and portid<x"40") else
242             rxbuff_out when (readstrobe = '1' and portid=x"FF") else
243
244             RAM_out2 when (readstrobe = '1' and (portid>=x"4A" and portid<x"8A")) else
245
246
247              RAM_out3 when (readstrobe = '1' and (portid>=x"8A" and portid<x"CA")) else
248
249             sal_sumador when(readstrobe= '1' and portid=x"47")else
250             sal_crc when(readstrobe='1' and portid=x"43")else
251             x"00";
252   end behavioral;
```

PERIFÉRICOS

SUMADOR

Para comenzar, veremos que nuestro periférico sumador, va a ser de 8 bits que al introducir dos caracteres devuelve la suma de estos. Iría conectado a las siguientes salidas de PicoBlaze:

- Out_port [7:0]
- Port_Id [7:0]
- Write_strobe
- data_out

Primero de todo necesitamos hacer el vhdl que necesitaremos para la suma, tendremos que ponerle un port-id para coger el dato de cada valor. Los datos que vamos a importar del componente son el "**0x44**" que será el primer valor, el segundo valor se leerá de la posición "**0x45**". La suma de estos dos, saldra por data_out que esta as.

Luego de hacer este componente, hay que instanciarlo y declararlo en el toplevel y por último, tendremos que añadir esa salida del periférico al multiplexor, sal_sumador con su respectiva posición que sera la "**0x47**" .

Una vez terminado el vhdl, tendremos que implementar el asm. En el cual tendremos que hacer para que se ingresen dos números en dos registros y que te imprima la salida.

Llamamos a "recibe", recibiendo el carácter que se le pasa al PC. Luego hacer un LOAD donde guardaremos el valor recibido de rxreg a txreg. Luego se transmite ese valor. Se resta 30, para poner el valor en decimal. Después pondríamos un OUTPUT para recoger el valor del 0x44. Se vuelve a llamar a recibe y enviar valor del 0x45. Se lee un valor por INPUT txreg,47 se vuelve a sumar 30 y se vuelve a transmitir el resultado final.

Una vez con el asm realizado, para comprobarlo, tendremos que pasarlo a código máquina con nuestro archivo asm y el cpp. Una vez hecho esto tendremos el archivo TESTSUMA.vhd. Una vez hecho todo lo anterior debemos sintetizar, implementar y generar el archivo de programa de nuestro diseño. Simplemente ahora comprobaremos con el Hiperterminal el funcionamiento de nuestro periférico.

```
-- Create Date:    12:53:16 11/07/2023
-- Design Name:
-- Module Name:    sumador - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity sumador is
    Port (
        port_id : in std_logic_vector(7 downto 0);   -- solo para depurar
        write_strobe : in std_logic;                   -- cambio: write_strobe debe ser de entrada
        out_port : in std_logic_vector(7 downto 0);   -- solo para depurar
        in_port : out std_logic_vector(7 downto 0);    -- solo para depurar
        data_out : out std_logic_vector(7 downto 0);
        reset : in std_logic;
        clk : in std_logic
    );
end sumador;

architecture Behavioral of sumador is
    signal R1, R2: std_logic_vector(7 downto 0);
begin
    R1_process: process(reset, clk)
    begin
        if reset = '1' then
            R1 <= (others => '0');
        elsif rising_edge(clk) then
            if write_strobe = '1' and port_id = x"45" then   -- cambio: corregir el nombre de la señal (portid a port_id)
                -- Realiza la suma y almacena el resultado en R1
                R1 <= out_port;
            end if;
        end if;
    end process R1_process;

    R2_process: process(reset, clk)
    begin
        if reset = '1' then
            R2 <= (others => '0');
        elsif rising_edge(clk) then
            if write_strobe = '1' and port_id = x"44" then   -- cambio: corregir el nombre de la señal (portid a port_id)
                -- Realiza la suma y almacena el resultado en R2
                R2 <= out_port;
            end if;
        end if;
    end process R2_process;

    data_out <= std_logic_vector(unsigned(R1) + unsigned(R2));
end Behavioral;
```

CRC

Hemos creado un CRC de 8 bits que funciona de forma paralela***(x^8 + x^2 + x + 1),*** el cual se le introduce un valor y a ese valor se le calcula su CRC, el cual se trata de un circuito que implementa registros de desplazamiento con realimentación, esta será a través de puertas XOR. En este caso, estamos ante un código de redundancia cíclica en paralelo, lo cual nos permite realizar la operación en un ciclo de reloj. Se le introducirá un valor y el cual pasará por la secuencia de combinación pseudoaleatoria, y obtendremos el resultado. Iría conectado a las siguientes salidas de PicoBlaze:

- Out_port [7:0]
- Port_Id [7:0]
- Write_strobe

Las entradas del CRC Serian

-Data: Este esta conectado al outport por donde le entra la información desde el picoblaze.

-crcIn: esto son puertos intermedios que conectan los biestables del CRC.

Las Salidas del CRC es:

-crcOut: esta apunta a Sal_crc que está conectado al multiplexor en la “43”.

```
24  library IEEE;
25  use IEEE.std_logic_1164.all;
26
27  entity crc is
28    Port(
29                 crcOut : out std_logic_vector(7 downto 0);
30                 data: in std_logic_vector(7 downto 0);
31                 port_id : in std_logic_vector(7 downto 0);
32                 write_strobe : in std_logic;
33                  reset : in std_logic;
34                 clk : in std_logic);
35
36  end crc;
37
38
39
40  architecture Behavioral of crc is
41
42
43        signal R1: std_logic_vector(7 downto 0);
44           signal crcIn: std_logic_vector(7 downto 0);
45
46
47
48
49  begin
50
51   R1_process: process(reset, clk)
52      begin
53          if reset = '1' then
54              R1 <= (others => '0');
55              crcIn<= (others => '0');
56          elsif rising_edge(clk) then
57              if write_strobe = '1' and port_id = x"41" then    -- cambio: corregir
    el nombre de la señal (portid a port_id
58                  R1 <= data;
59              end if;
60          end if;
61      end process R1_process;
```

```
62
63
64
65          crcOut(0) <= crcIn(0) xor crcIn(6) xor crcIn(7) xor R1(0) xor R1(6) xor R1(7);
66          crcOut(1) <= crcIn(0) xor crcIn(1) xor crcIn(6) xor R1(0) xor R1(1) xor R1(6);
67          crcOut(2) <= crcIn(0) xor crcIn(1) xor crcIn(2) xor crcIn(6) xor R1(0) xor R1(1
    ) xor R1(2) xor R1(6);
68          crcOut(3) <= crcIn(1) xor crcIn(2) xor crcIn(3) xor crcIn(7) xor R1(1) xor R1(2
    ) xor R1(3) xor R1(7);
69          crcOut(4) <= crcIn(2) xor crcIn(3) xor crcIn(4) xor R1(2) xor R1(3) xor R1(4);
70          crcOut(5) <= crcIn(3) xor crcIn(4) xor crcIn(5) xor R1(3) xor R1(4) xor R1(5);
71          crcOut(6) <= crcIn(4) xor crcIn(5) xor crcIn(6) xor R1(4) xor R1(5) xor R1(6);
72          crcOut(7) <= crcIn(5) xor crcIn(6) xor crcIn(7) xor R1(5) xor R1(6) xor R1(7);
73      end architecture Behavioral;
```

Se ha realizado un banco de pruebas para la comprobación de su correcto funcionamiento y se ha verificado que cumple su objetivo.

INSTRUCCIÓN CRC

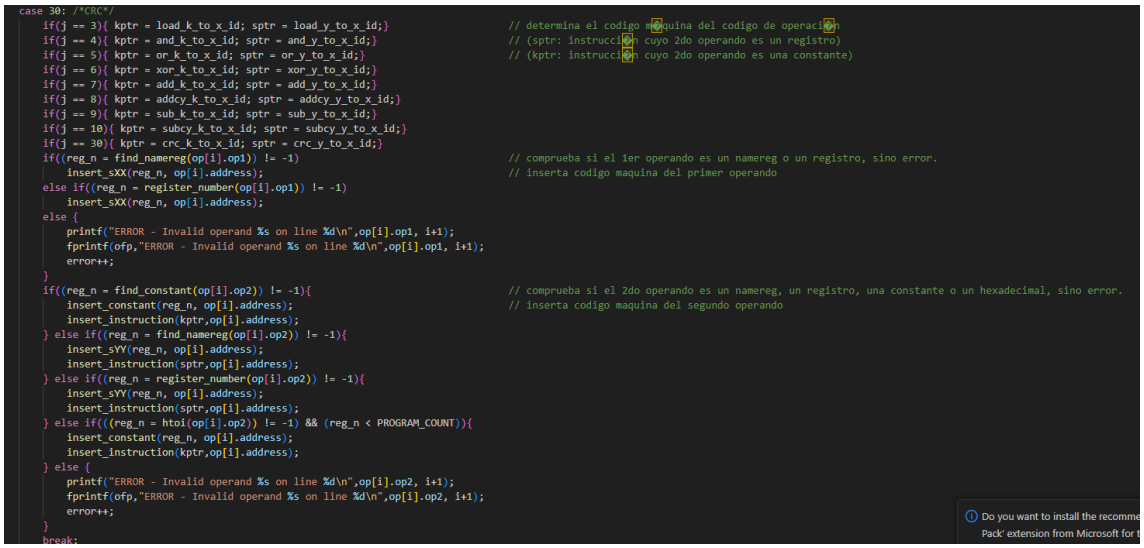
Para la incorporación de una nueva instrucción, hemos ido fijándonos poco a poco, en cómo tenemos que ir cambiando los datos para la suma de esta instrucción. Es una instrucción de dos operandos.

El funcionamiento de esta instrucción es sencillo, simplemente hemos querido que haga la XOR de dos registros.
Para la incorporación de una nueva instrucción, hemos ido fijándonos poco a poco, en cómo tenemos que ir cambiando los datos para añadir esta instrucción. Desde la modificación del cpp.

```
   /*nueva instruccion de ejemplo para el crc*/
 char * crc_y_to_x_id="11101";                                  //Codigo de operacion, 5 bits m s significativos de los 16 que forman las instrucci n
 char *crc_k_to_x_id ="10101";                                 //Cuando es de k a registro, el 4 bit es 0. Y viceversa.

/* input/output group */
   ADDRESS",  /* 28 */
   "FLIP",/* 29 */ /* added new instruction */
   "CRC"}; /*30*/
nt error = 0;
*================================== */
oid free_mem(void)                                                  // libera la estructura que contiene los campos de la instruccion

  int i;
  #define MAX_LINE_COUNT 1000 /* max 1000 lines allowed */           // M ximo numero de lineas del programa, incluye lineas con comentarios.
  #define PROGRAM_COUNT 256   /* total program word */               // Maximo numero de instrucciones en el programa

  /* increase instruction_count for added new instruction */
  #define instruction_count 31/* total instruction set */           // Numero m ximo de instrucciones

  #define CONSTANT_COUNT 100  /* max 100 constant can be declared */
  #define REG_COUNT 8         /* max 8 namereg can be declared */     // Maximo Numero de registros
```

Donde tenemos que aumentar este número máximo de instrucciones, incluimos el código de operación (dos que no estén en uso), añadir el nombre de la nueva instrucción (llamada “CRC”), ver a que grupo de instrucciones pertenece por sus datos. Es una instrucción de dos operandos.

```
        break;                                                  // Para instrucciones aritmetica l gicas comprueba que tiene operando1 y operando2.
    case 3: /* LOAD */
    case 4: /* AND */
    case 5: /* OR */
    case 6: /* XOR */
    case 7: /* ADD */
    case 8: /* ADDCY */
    case 9: /* SUB */
    case 10: /* SUBCY */
    case 30: /*ADD new instruction for CRC*/
        if((op[i].op1 == NULL) || (op[i].op2 == NULL)){
            printf("ERROR - Missing operand for %s on line %d\n",op[i].instruction, i+1);
            fprintf(ofp,"ERROR - Missing operand for %s on line %d\n",op[i].instruction, i+1);
            error++;
        }
        break;
    case 11: /* SR0 */
```



```
case 30: /*CRC*/
    if(j == 3){ kptr = load_k_to_x_id; sptr = load_y_to_x_id;}                  // determina el codigo de maquina del codigo de operacion
    if(j == 4){ kptr = and_k_to_x_id; sptr = and_y_to_x_id;}                    // (sptr: instruccion cuyo 2do operando es un registro)
    if(j == 5){ kptr = or_k_to_x_id; sptr = or_y_to_x_id;}                      // (kptr: instruccion cuyo 2do operando es una constante)
    if(j == 6){ kptr = xor_k_to_x_id; sptr = xor_y_to_x_id;}
    if(j == 7){ kptr = add_k_to_x_id; sptr = add_y_to_x_id;}
    if(j == 8){ kptr = addcy_k_to_x_id; sptr = addcy_y_to_x_id;}
    if(j == 9){ kptr = sub_k_to_x_id; sptr = sub_y_to_x_id;}
    if(j == 10){ kptr = subcy_k_to_x_id; sptr = subcy_y_to_x_id;}
    if(j == 30){ kptr = crc_k_to_x_id; sptr = crc_y_to_x_id;}
    if((reg_n = find_namereg(op[i].op1)) != -1)                                 // comprueba si el 1er operando es un namereg o un registro, sino error.
        insert_sXX(reg_n, op[i].address);                                       // inserta codigo maquina del primer operando
    else if((reg_n = register_number(op[i].op1)) != -1)
        insert_sXX(reg_n, op[i].address);
    else {
        printf("ERROR - Invalid operand %s on line %d\n",op[i].op1, i+1);
        fprintf(ofp,"ERROR - Invalid operand %s on line %d\n",op[i].op1, i+1);
        error++;
    }
    if((reg_n = find_constant(op[i].op2)) != -1){                               // comprueba si el 2do operando es un namereg, un registro, una constante o un hexadecimal, sino error.
        insert_constant(reg_n, op[i].address);                                  // inserta codigo maquina del segundo operando
        insert_instruction(kptr,op[i].address);
    } else if((reg_n = find_namereg(op[i].op2)) != -1){
        insert_sYY(reg_n, op[i].address);
        insert_instruction(sptr,op[i].address);
    } else if((reg_n = register_number(op[i].op2)) != -1){
        insert_sYY(reg_n, op[i].address);
        insert_instruction(sptr,op[i].address);
    } else if(((reg_n = htoi(op[i].op2)) != -1) && (reg_n < PROGRAM_COUNT)){
        insert_constant(reg_n, op[i].address);
        insert_instruction(kptr,op[i].address);
    } else {
        printf("ERROR - Invalid operand %s on line %d\n",op[i].op2, i+1);
        fprintf(ofp,"ERROR - Invalid operand %s on line %d\n",op[i].op2, i+1);
        error++;
    }
    break;
```


Una vez hecha todas estas modificaciones en el programa cpp, pasamos a hacer las modificaciones en el hardware.

```
 --CRC
  constant crc_y_to_x_id : std_logic_vector(4 downto 0) := "11101";
  constant crc_k_to_x_id : std_logic_vector(4 downto 0) := "10101";
--
component crc_group
Port(first_operand : in std_logic_vector(7 downto 0);
          second_operand : in std_logic_vector(7 downto 0);
          Y : out std_logic_vector(7 downto 0);
           clk : in std_logic);
            end component;

component register_and_flag_enable
    Port (i_logical: in std_logic;
          i_arithmetic: in std_logic;
          i_shift_rotate: in std_logic;
          i_flip: in std_logic;                -- added new instruction
          i_returni: in std_logic;
          i_crc: in std_logic;  -- NEW INSTRUCTION CRC
          i_input: in std_logic;
          active_interrupt : in std_logic;
          T_state : in std_logic;
          register_enable : out std_logic;
          flag_enable : out std_logic;
          clk : in std_logic);
    end component;
 --CRC
 signal i_crc: std_logic;
 signal crc_result : std_logic_vector(7 downto 0);

 --CRC INSTRUCTION
       crc_definition: crc_group
       port map (first_operand => sX_register,
                 second_operand => second_operand,
                 Y => crc_result,
                 clk => clk);
```

```
-- added new instruction
i_flip <= '1' when instruction(15 downto 11) = flip_id else '0';

i_crc <= '1' when instruction(15 downto 11) = crc_y_to_x_id or instruction(15
downto 11) = crc_k_to_x_id else '0';



   i_add_sub <= instruction(12);
   i_carry_nocarry <= instruction(11);

   i_arithmetic <= i_add_k_to_x or i_add_y_to_x or i_addcy_k_to_x or i_addcy_y_to_x
                or i_sub_k_to_x or i_sub_y_to_x or i_subcy_k_to_x or i_subcy_y_to_x;

   i_logical <= i_load_k_to_x or i_load_y_to_x or i_and_k_to_x or i_and_y_to_x
             or i_or_k_to_x or i_or_y_to_x or i_xor_k_to_x or i_xor_y_to_x;

   i_input <= i_input_p_to_x or i_input_y_to_x;
   i_output <= i_output_p_to_x or i_output_y_to_x;

   --
   -- get ALU result
   --
   ALU_loop: for i in 0 to 7 generate
   begin
      ALU_result(i) <= (shift_and_rotate_result(i) and i_shift_rotate)
                    or (in_port(i) and i_input)
                    or (arithmetic_result(i) and i_arithmetic)
                  or (flip_result(i) and i_flip)      -- added new instruction
                    or (crc_result(i) and i_crc)
                    or (logical_result(i) and i_logical);
   end generate ALU_loop;

   -- decode second operand
   second_operand <= sY_register when instruction(14) = '1' else instruction(7
downto 0);

end Behavioral;
```

En la cual tendremos que hacer unas cuantas modificaciones en el procesador. Como algunas de, declarar el componente de la instrucción para que sea de dos operandos como nosotros queremos, declarar sus flags, instanciar el componente de la instrucción, etc.

```
20   library IEEE;
21   use IEEE.STD_LOGIC_1164.ALL;
22   use IEEE.STD_LOGIC_ARITH.ALL;
23   use IEEE.STD_LOGIC_UNSIGNED.ALL;
24
25   -- Uncomment the following library declaration if using
26   -- arithmetic functions with Signed or Unsigned values
27   --use IEEE.NUMERIC_STD.ALL;
28
29   -- Uncomment the following library declaration if instantiating
30   -- any Xilinx primitives in this code.
31   --library UNISIM;
32   --use UNISIM.VComponents.all;
33
34   entity crc_group is
35     Port (
36       first_operand  : in  std_logic_vector(7 downto 0);
37       second_operand : in  std_logic_vector(7 downto 0);
38       Y              : out std_logic_vector(7 downto 0);
39       clk            : in  std_logic
40     );
41   end crc_group;
42
43   --
44   architecture crc_definition of crc_group is
45   begin
46     bus_width_loop: for i in 0 to 7 generate
47     begin
48        CRC:
49        process (clk)
50         begin
51         if (clk'event and clk = '1') then
52          Y(i) <= first_operand(i) xor second_operand(i);
53
54         end if;
55        end process CRC;
56     end generate bus_width_loop;
57   --
58   end crc_definition;
59
```

Esta instrucción, diseñada para realizar la operación de exclusión lógica (XOR) entre dos registros, desempeña un papel crucial en nuestro trabajo final, por ello hemos implementado esto para a la hora del calculo del CRC no sea siempre los mismos valores.

FUNCIONAMIENTO

(programFinal.asm)

El funcionamiento detallado del programa es el siguiente:

1. Uso de RAMs para mostrar mensajes por pantalla:

   El programa hace uso de 3 RAMs las cuales va a barrer estas para que lea los mensajes que hay almacenados en estas, desde los portid que hemos marcado.

```
                LOAD            s7,00
parte1:
                INPUT           txreg,s7
                ADD             txreg,00
                JUMP Z          CARGA2
                CALL            transmite
                ADD             s7,01
                JUMP            parte1

CARGA2:         LOAD            s7,4A
mensaje2:       INPUT           txreg,s7
                ADD             txreg,00
                JUMP Z          CARGA3
                CALL            transmite
                ADD             s7,01
                JUMP            mensaje2


CARGA3:         LOAD            s7,8A
mensaje3:       INPUT           txreg,s7
                ADD             txreg,00
                JUMP Z          parte2
                CALL            transmite
                ADD             s7,01
                JUMP            mensaje3
```

2. Bucle de resta del 1 al 9

   Después de mostrar los mensajes en pantalla, el programa entra en un bucle el cual se carga en 9 y se va restando 1 cada vez, hasta llegar a 0 y vuelve a cargar 9. Y se queda ahí hasta que ocurra una interrupción en el programa.

```
parte2:         ENABLE INTERRUPT
bucle1:         LOAD            s6,09
bucle2:         SUB             s6,01
                JUMP NZ         bucle2
                LOAD            s6,09
                JUMP            bucle2
```

3. Interrupción valor recibido

Cuando se presiona un valor aleatorio, el programa salta a una interrupción específica. Y ya es cuando ejecuta todas las acciones que hay en ella. Dentro de esta interrupción, primero se ejecuta la instrucción CRC, la cual como ya hemos comentado antes es una XOR. Se hará la XOR del valor aleatorio con el numero del 1 al 9 aleatorio el cual ha sacado en ese momento del bucle. Este será el primer número que sacará por pantalla. El segundo es el numero aleatorio con el cual hacemos la XOR, para saber el resultado coherente de esta. Este valor hay que saber que te lo da en ASCII y hay que saber que la XOR en hexadecimal, para que no haya confusiones.

4. Periférico CRC

El resultado de esta XOR, se le lleva como valor de entrada al periférico CRC. El cual, este hará su proceso y sacara el CRC del primer valor que nos ha mostrado por pantalla. Por tanto, el tercer valor que te muestra por pantalla es el resultado final del programa.

```
interrup:       DISABLE         INTERRUPT

                CALL            recibe

                CRC             rxreg,S6
                ;LOAD            S6,rxreg



                LOAD            txreg,rxreg
                CALL            transmite
                ADD             s6,30
                LOAD            txreg,S6

                CALL            transmite
                SUB             s6,30
                OUTPUT          rxreg,41
                INPUT           txreg,43
                CALL            transmite
                RETURNI         ENABLE

                ADDRESS         FF
                JUMP            interrup
```