Escuela Técnica Superior Ingeniería de Telecomunicación

MEMORIA PRÁCTICA 2: PICOBLAZE

Conversión a PicoBlaze de 16 bits

DIEGO ISMAEL ANTOLINOS GARCÍA
ANDRÉS RUZ NIETO

1. Características principales

Características	Picoblaze modificado
Arquitectura	16 bits
Direccionamiento E/S	16 bits
Tamaño de pila	8
Instrucciones adicionales	SWAP
Numero de instrucciones	31
Tamaño de instrucción	26
Longitud fichero ensamblador	65536
Longitud del programa	65536
Número de registros	16

SEÑAL INSTRUCCIÓN			
0	low		S_IN_BIT
1	low		S_CODE0
2	low		S_CODE1
3	low		S_RIGHT
4	low		
5	low		
6	low		
7	low		
8	low		
9	low		
10	low		
11	low		
12	low	b	
13	low	b	
14	low	b	
15	low	b	
16	a		
17	a		ZERO AND CARRY
18	a		ZERO AND CARRY
19	a		CONDITIONAL
20	i		LOGICAL 0
21	i		LOGICAL 1
22	i		
23	i		
24	i		
25	i		

Imagen 1. Nueva Señal

2. Cambios realizados en el programa VHDL

Para la conversión de *PicoBlaze* 8 bits a *PicoBlaze* 16 bits hemos tenido que realizar una serie de modificaciones en el código VHDL. Dichas modificaciones son las siguientes:

1. **Añadir un bit a las instrucciones existentes:** Para ello hemos añadido un cero a la izquierda del bit mas significativo de cada instrucción.

```
-- program control group
constant jump_id : std_logic_vector(5 downto 0) := "011010";
constant call_id : std_logic_vector(5 downto 0) := "011011";
constant return_id : std_logic_vector(5 downto 0) := "010010";
--
-- logical group
constant load_k_to_x_id : std_logic_vector(5 downto 0) := "000000";
constant load_y_to_x_id : std_logic_vector(5 downto 0) := "001000";
constant and_k_to_x_id : std_logic_vector(5 downto 0) := "000001";
constant and_y_to_x_id : std_logic_vector(5 downto 0) := "001001";
constant or_k_to_x_id : std_logic_vector(5 downto 0) := "000010";
constant or_y_to_x_id : std_logic_vector(5 downto 0) := "001010";
constant xor_k_to_x_id : std_logic_vector(5 downto 0) := "000011";
constant xor_y_to_x_id : std_logic_vector(5 downto 0) := "001011";
```

2. **Aumento del contador de programa:** Cambiamos la señal *program_counter_address* de 8 a 16 pasando de 256 a 65536 palabras.

```
constant program_counter_address : natural := 16;
```

3. **Aumento del banco de registros:** Cambiamos la señal *register_bank_address* de 3 a 4 cambiando de 8 a 16 registros.

```
constant register_bank_address : natural := 4;
```

4. **Aumento de la pila:** Cambiamos la señal *stack_counter_address* de 2 a 3 cambiando el tamaño de la pila de 4 a 8.

```
constant stack_counter_address : natural := 3;
```

5. **Adaptación de todos los componentes:** Hemos adaptado todos los componentes para que en vez de utilizar 8 bits operen con 16 bits.

```
component arithmetic_process
    Port (first_operand : in std_logic_vector(15 downto 0);
          second_operand : in std_logic_vector(15 downto 0);
          carry_in : in std_logic;
          code1 : in std_logic;
          code0 : in std_logic;
          Y : out std_logic_vector(15 downto 0);
          carry_out : out std_logic;
          clk : in std_logic);
    end component;
```

```
component logical_bus_processing
    Port (first_operand : in std_logic_vector(15 downto 0);
          second_operand : in std_logic_vector(15 downto 0);
          code1 : in std_logic;
          code0 : in std_logic;
          Y : out std_logic_vector(15 downto 0);
          clk : in std_logic);
    end component;
```

6. **Cambio en el Register Bank:** Modificación en la declaración de las señales *a* y *dpra*, *a* y *b* en *Imagen 1* respectivamente, para adaptarlas a la nueva señal *instruction*.

```
data_registers: register_bank
generic map (register_bank_address)
port map (we => register_write_enable,
          d_bus => ALU_result,
          wclk => clk,
          a => instruction(19 downto 16),
          dpra => instruction(15 downto 12),
          spo_bus => sX_register,
          dpo_bus => sY_register );
```

7. **Cambio en las señales de condición y flag:** Modificación en la declaración de las señales *conditional*, *zero*, *not_zero*, *carry* y *not_carry*, tal y como podemos ver en *Imagen 1*.

```
conditional <= instruction(19);
zero <= '1' when instruction(18 downto 17) = zero_id else '0';
not_zero <= '1' when instruction(18 downto 17) = not_zero_id else '0';
carry <= '1' when instruction(18 downto 17) = carry_id else '0';
not_carry <= '1' when instruction(18 downto 17) = not_carry_id else '0';
flag_condition_met <= (zero and zero_flag) or (not_zero and (not zero_flag))
                or (carry and carry_flag) or (not_carry and (not carry_flag));
```

8. **Cambio en las señales lógicas:** Modificación en la declaración de las señales *logical_code0* y *logical_code1,* tal y como podemos ver en *Imagen 1*.

```
logical_code1 <= instruction(21);
logical_code0 <= instruction(20);
```

9. **Cambio en el establecimiento de señales de decodificación de las instrucciones:** Hemos adaptado las señales de decodificación de instrucciones, encontrándose ahora entre los bits 20-25.

```
i_jump <= '1' when instruction(25 downto 20) = jump_id else '0';
i_call <= '1' when instruction(25 downto 20) = call_id else '0';
i_return <= '1' when instruction(25 downto 20) = return_id else '0';
i_returni <= '1' when instruction(25 downto 20) = returni_id else '0';
i_load_k_to_x <= '1' when instruction(25 downto 20) = load_k_to_x_id else '0';
i_load_y_to_x <= '1' when instruction(25 downto 20) = load_y_to_x_id else '0';
i_and_k_to_x <= '1' when instruction(25 downto 20) = and_k_to_x_id else '0';
i_and_y_to_x <= '1' when instruction(25 downto 20) = and_y_to_x_id else '0';
i_or_k_to_x <= '1' when instruction(25 downto 20) = or_k_to_x_id else '0';
i_or_y_to_x <= '1' when instruction(25 downto 20) = or_y_to_x_id else '0';
i_xor_k_to_x <= '1' when instruction(25 downto 20) = xor_k_to_x_id else '0';
i_xor_y_to_x <= '1' when instruction(25 downto 20) = xor_y_to_x_id else '0';
i_add_k_to_x <= '1' when instruction(25 downto 20) = add_k_to_x_id else '0';
i_add_y_to_x <= '1' when instruction(25 downto 20) = add_y_to_x_id else '0';
```

10. **Cambio en el decodificador de segundo operando:** Modificación para la comprobación de si el registro sY está siendo usado por la instrucción.

```
second_operand <= sY_register when instruction(23) = '1' else instruction(15 downto 0);
```

11. **Cambio en la entidad *program_counter*:** Hemos modificado la señal *count_value* para ampliarla de 8 a 16 bits. Y que el salto de la interrupción pase de la línea 255 se realice en la línea 65535.

```
dual_loadable_counter:
process (clk)
begin
   if clk'event and clk = '1' then
      if interrupt = '1' then
         count_value <= "1111111111111111";
      else if reset = '1' then
         count_value <= "0000000000000000";
      else if T_state = '0' then
         if normal_count = '1' then
            count_value <= count_value + 1;
```

12. **Cambios en toplevel:** Modificamos la RAM de 8 a 16 bits, para ello cambiamos los *portid* de 8 a 16 bits y añadimos 15 ceros a *rxbuff_out* que se añadirán en el puerto de entrada cuando se produzca la lectura.

```
type ram_type is array (0 to 63) of std_logic_vector (15 downto 0);
signal RAM : ram_type := (
x"000A", x"000D", x"002A", x"0020", x"0044", x"0049", x"0045", x"0047",
x"004F", x"0020", x"0059", x"0020", x"0041", x"004E", x"0044", x"0052",
x"0045", x"0053", x"0020", x"002A", x"000A", x"000D", x"002A", x"0020",
x"0031", x"0036", x"0020", x"0042", x"0049", x"0054", x"0053", x"0020",
x"002A", x"000A", x"000D", x"002A", x"0020", x"0032", x"0030", x"0032",
x"0030", x"002F", x"0032", x"0030", x"0032", x"0031", x"0020", x"002A",
x"000A", x"000D", x"0000", x"0000", x"0000", x"0000", x"0000", x"0000",
x"0000", x"0000", x"0000", x"0000", x"0000", x"0000", x"0000", x"0000");
```

13. **Instrucción SWAP:** Introducimos una instrucción nueva llamada SWAP, que se encarga de intercambiar en bloque los 8 bits más significativos por los 8 bits menos significativos.

```
aux <= operand(7 downto 0) & operand(15 downto 8);

bus_width_loop: for i in 0 to 15 generate
begin
   FF:
   process (clk)
    begin
    if (clk'event and clk = '1') then
       Y(i) <= aux(i);
    end if;
   end process FF;
end generate bus_width_loop;
```

3. Cambios realizados en programa ensamblador

Posteriormente hemos tenido que realizar una serie de cambios en el programa ensamblador para completar la conversión de *PicoBlaze* 8 bits a *PicoBlaze* 16 bits. Las modificaciones son las siguientes:

1. **Definición de diferentes constantes:** Hemos definidos las constantes *MAX_LINE_COUNT*, *PROGRAM_COUNT*, *instruction_count*, *CONSTANT_COUNT* Y *REG_COUNT*.

```
#define MAX_LINE_COUNT 65536
#define PROGRAM_COUNT 65536

#define instruction_count 31

#define CONSTANT_COUNT 100
#define REG_COUNT 16
```

2. **Comprobación en la sintaxis de los registros:** Modificamos la comprobación de la sintaxis de los registros, para que el programa funcionase correctamente con los registros s10 y superiores. Ya que anteriormente el programa no estaba preparado para registros de 2 dígitos.

```
int register_number(char *s)
{
    if(*s != 'S') {
        return( -1 );
    }
    if(strlen(s) != 2 && strlen(s) != 3){
        return( -1 );
    }
    if(strlen(s) == 2){
        if((*(s+1) >= '0') && (*(s+1) <= '9')) return (*(s+1) - '0');
        else return( -1 );
    }
    if(strlen(s) == 3){
        if(*(s+1) == '1' && (*(s+2) >= '0') && (*(s+2) <= '5')) return (10*(*(s+1) - '0') + (*(s+2) - '0'));
        else return( -1 );
    }
    else return( -1 );
}
```

3. **Adaptación de los campos de la nueva instrucción:** Adaptamos la escritura de los campos a los nuevos valores.

```
void insert_sXX(int c, int p)
{
    program_word[p] = program_word[p] | (unsigned) (c << 16);
}

/*==================================== */
void insert_sYY(int c, int p)
{
    program_word[p] = program_word[p] | (unsigned) (c << 12);
}

/*==================================== */
void insert_constant(int c, int p)
{
    program_word[p] = program_word[p] | (unsigned) (c);
}

/*==================================== */
void insert_flag(int c, int p)
{
    program_word[p] = program_word[p] | (unsigned) (c << 17);
}
```

4. **Modificación del texto que se escribe en el programa VHLD:** Modificamos el texto que se va a escribir en nuestro programa VHDL acorde a las modificaciones ya realizadas para nuestro *PicoBlaze* 16 bits. Además, cambiamos también el valor de j en ambos bucles *for* a 25 (tamaño de instrucción).

```
fprintf(ffp,"library ieee;\nuse ieee.std_logic_1164.all;\n");
fprintf(ffp,"use ieee.std_logic_unsigned.all;\n\n");

fprintf(ffp,"entity %s is\n",basename);
fprintf(ffp,"\tport( address : in std_logic_vector(15 downto 0);\n");
fprintf(ffp,"\t\tclk : in std_logic;\n\t\tdout : out std_logic_vector(25 downto 0));\n\tend;\n\n");
fprintf(ffp,"architecture v1 of %s is\n\n", basename);

fprintf(ffp,"\tconstant ROM_WIDTH: INTEGER:= 26;\n");
fprintf(ffp,"\tconstant ROM_LENGTH: INTEGER:= 65536;\n\n");
fprintf(ffp,"\tsubtype rom_word is std_logic_vector(ROM_WIDTH-1 downto 0);\n");
fprintf(ffp,"\ttype rom_table is array (0 to ROM_LENGTH-1) of rom_word;\n\n");
fprintf(ffp,"constant rom: rom_table := rom_table'(\n");
for(i = 0; i < PROGRAM_COUNT-1; i++){
    fprintf(ffp, "\t\"");
    for(j = 25; j >= 0; j--)
        fprintf(ffp, "%d", (program_word[i]>>j) & 1); //print binary
    fprintf(ffp, "\",\n");
}
fprintf(ffp, "\t\"");
for(j = 25; j >= 0; j--)
    fprintf(ffp, "%d", (program_word[i]>>j) & 1); //print binary
fprintf(ffp, "\");\n\n");

fprintf(ffp,"begin\n\nprocess (clk)\nbegin\n", basename);
fprintf(ffp,"\tif clk'event and clk = '1' then\n\t\tdout <= rom(conv_integer(address));\n");
fprintf(ffp,"\tend if;\nend process;\nend v1;\n");
```

5. **Modificación de ASM**

```
ADDRESS      FFFF
JUMP         interrup
```