Contador Up-Down Controlado por UART

Roberto Federico Farfán

Objetivo: El objetivo de este trabajo es implementar un contador Up-Down de 12 bits controlado por UART. El sistema digital se implementará en una FPGA Arty Z7-10, utilizando un contador de 12 bits desarrollado en código VHDL y un microcontrolador embebido ARM Cortex A9. La lógica y la estructura de la implementación se desarrollo utilizando los ejemplos descriptos en las clases de Microarquitecturas y Softcores, perteneciente al posgrado de Sistema Embebidos de la UBA. En el trabajo se realizan simulaciones utilizando el software GHDL y GTKwave. Se implemento la lógica en una FPGA realizando pruebas del desarrollo de manera remota.

Introducción

El contador se implementó utilizando una FPGA Arty Z7-10 como se observa en la Figura 1.

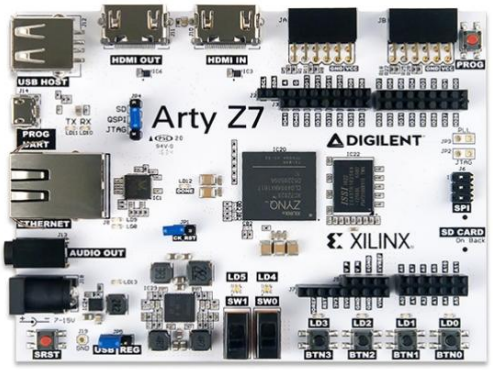

Figura 1. FPGA Arty Z7-10.

Para la implementación final, la programación se realizará de forma remota, utilizando una FPGA que se encuentra en el laboratorio de sistemas embebidos de la UBA.

El diseño del IP asociado al Bus-AXI tiene la siguiente estructura.

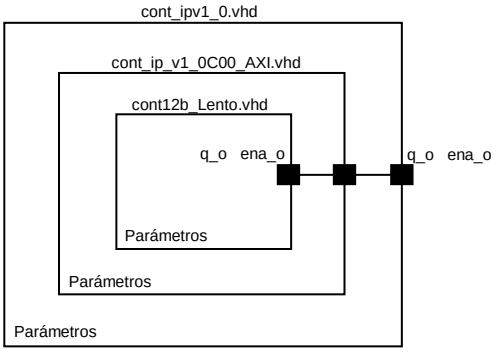

Figura 2. Esquema del IP desarrollado.

Código VHDL: Contador de 12 bits Up-Down.

El contador se desarrolló en código VHDL, utilizando como base el ejemplo implementado en clase denominado cont4b_Lento.vhd. Para este contador se agregaron dos variables más de entrada: up-down y step_i.

```
entity cont12b_Lento is
    generic(
        N: natural := 12;
        CICLOS: natural := 2
    );
    port(
        clk_i   : in std_logic;
        rst_i   : in std_logic;
        ena_i   : in std_logic;
        step_i  : in std_logic;
        up_down : in std_logic;
        q_o     : out std_logic_vector(N-1 downto 0);
        ena_o   : out std_logic
    );
end;
```

Figura 3. Variables de entrada y salida del contador.

En el codigo VHDL se observa la definición de variables genericas. La variable N indica la cantidad de bits de salida del contador y la variable CICLOS, el número de ciclos de reloj que deben pasar para que el contador incremente o decremente.

Por otro lado, se definen la entrada de reloj (clk_i), el reset (rst_i), el habilitador (ena_i), el paso en la cuenta (step_i), la entrada que indica si el contador incrementa o decrementa (up-down), el habilitador de salida (ena_o) y la salida (q_o).

En el siguiente grafico se observa el codigo VHDL del contador.

```
process(clk_i)
    variable cicleCount: integer := 0;
    variable count: unsigned(N-1 downto 0);
begin
    if rising_edge(clk_i) then
        if rst_i = '1' then
            count := (N-1 downto 0 => '0');
            ena_o <= '0';
        elsif ena_i = '1' then
            cicleCount := cicleCount + 1;
                if cicleCount = CICLOS then
                    cicleCount := 0;
                        if up_down = '0' then
                            if step_i = '0' then
                                count := count + 1;
                            else
                                count := count + 2;
                            end if;
                        else
                            if step_i = '0' then
                                count := count - 1;
                            else
                                count := count - 2;
                            end if;
                        end if;
                    ena_o <= '1';
                else
                    ena_o <= '0';
                end if;
        end if;
    end if;
    q_o <= std_logic_vector(count);
end process;
```

Figura 4. Contador Up-Down.

Se definen dos variables: cicleCount y count. La variable cicleCount almancenan los ciclos de reloj que transcurren en el tiempo. Cuando cicleCount se iguala a CICLOS, el contador incrementa o decrementa la cuenta. La variable step_i indica el paso del contador, si

incrementa o decrementa en una o dos unidades. La variable up_down en cero le indica al contador que incremente la cuenta, mientras que la variable mencionada en uno, le indica al contador que decremente.

Simulación del Contador de 12 bits Up-Down.

La simulación del contador se realizo implementando el software GHDL y GTKwave. En la simulación se observan las entradas y salidas del contador. La salida q_o inicialmente se encuentra en cero, con el habilitador en uno (ena_i), el up-down en cero (incrementa), el paso en cero (step_i) y el reset en cero despues de los 50ns.

En la simulación se observa que el contador incrementa en una unidad hasta los 500ns.

Despues del tiempo mencionado, step_i cambia a uno y el contador incrementa en pasos de dos unidades.

Entre los 620ns y los 800ns, el habilitador cambia a cero. Debido a esto la cuenta se mantiene.

Despues de los 1000ns step_i cambia a cero y la cuenta se incrementa en una unidad.

Despues de los 1300ns la entrada up-down cambia a uno, por lo que la cuenta decrementa en una unidad.

Pasando los 1500ns step_i cambia a uno y el contador decrementa en dos unidades.

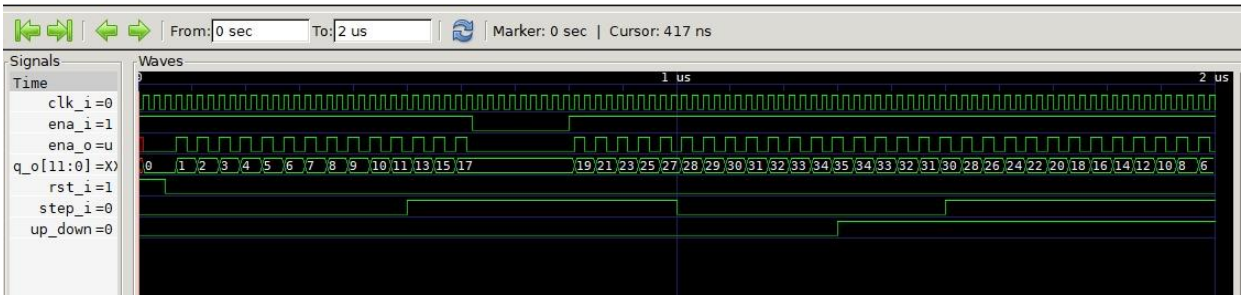

Figura 5. Simulación del Contador Up-Down.

Desarrollo del IP core

Para el diseño del contador se genero un IP core que se vincula al BUS AXI denominado count_ip. El IP implementado se diseño con el número mínimo de registros que permite la aplicación, 4 de 32 bits. Se realizo el control del contador por medio del registro slv_reg0.

```
generic (
    -- Users to add parameters here
    N: natural := 4;
    CICLOS: natural := 1000;
    -- User parameters ends
    -- Do not modify the parameters beyond this line

    -- Width of S_AXI data bus
    C_S_AXI_DATA_WIDTH  : integer   := 32;
    -- Width of S_AXI address bus
    C_S_AXI_ADDR_WIDTH  : integer   := 4
);
---- Number of Slave Registers 4
signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal slv_reg_rden : std_logic;
signal slv_reg_wren : std_logic;
signal reg_data_out :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
signal byte_index   : integer;
signal aw_en    : std_logic;
```

Figura 6. Registros generados.

Se vincularon los bits del registro slv_reg0 con las entrada del contador de la siguiente manera:

```
cont_inst: entity work.cont12b_Lento
        generic map(
            N => N,
            CICLOS => CICLOS
        )
        port map(
            clk_i => S_AXI_ACLK,
            rst_i => slv_reg0(0),
            ena_i => slv_reg0(1),
            step_i => slv_reg0(2),
            up_down => slv_reg0(3),
            q_o => q_o,
            ena_o => ena_o
        );
```

Figura 7. Relación entre el registro slv_reg0 y las entrada del contador.

El bit 0 se relaciona con el rst_i, el bit 1 con el ena_i, el bit 2 con el step_i y el bit 3 con el up_down. El clk_i se conecto al reloj del AXI. De acuerdo a la combinación de señales digitales en la entrada del contador, su comportamiento puede describirse de la siguiente manera:

Tabla 1: Acciones del contador en función de las entradas digitales.

	Up_down	Step_i	Ena_i	Rst_i
1. Reset	0	0	0	1
2. Incrementa en 1	0	0	1	0
3. Incrementa en 2	0	1	1	0
4. Decrementa en 1	1	0	1	0
5. Decrementa en 2	1	1	1	0
6. Parar cuenta	0	0	0	0

En base a las funciones descriptas en la Tabla 1, se confeccionó el menu que el microcontrolador envia como cartel por UART en la implementación final.

El carácter que ingresa por la UART se almacena en la variable byte. Por ejemplo, si el carcater ingresado por consola es uno, la variable cuenta_incremento definida como entera, toma el valor uno y escribe con este valor el registro slv_reg0.

```
int cuenta_incrementa = 0;
char byte = 0;

print("////////////////////////////////////// \n\r");
print("//1- RESET del contador           // \n\r");
print("//2- Incrementa cuenta en 1       // \n\r");
print("//3- Incrementa cuenta en 2       // \n\r");
print("//4- Decrementa cuenta en 1       // \n\r");
print("//5- Decrementa cuenta en 2       // \n\r");
print("//6- PARAR cuenta                 // \n\r");
print("////////////////////////////////////// \n\r");
```

```
while(1){

    byte = XUartPs_RecvByte(XPAR_PS7_UART_0_BASEADDR);

    switch(byte){
    case '1':
        cuenta_incrementa = 1;
        CONT_IP_mWriteReg(XPAR_CONT_IP_0_C00_AXI_BASEADDR, CONT_IP_C00_AXI_SLV_REG0_OFFSET, cuenta_incrementa);
        break;
    case '2':
        cuenta_incrementa = 2;
        CONT_IP_mWriteReg(XPAR_CONT_IP_0_C00_AXI_BASEADDR, CONT_IP_C00_AXI_SLV_REG0_OFFSET, cuenta_incrementa);
        break;
    case '3':
        cuenta_incrementa = 6;
        CONT_IP_mWriteReg(XPAR_CONT_IP_0_C00_AXI_BASEADDR, CONT_IP_C00_AXI_SLV_REG0_OFFSET, cuenta_incrementa);
        break;
    case '4':
        cuenta_incrementa = 10;
        CONT_IP_mWriteReg(XPAR_CONT_IP_0_C00_AXI_BASEADDR, CONT_IP_C00_AXI_SLV_REG0_OFFSET, cuenta_incrementa);
        break;
    case '5':
        cuenta_incrementa = 14;
        CONT_IP_mWriteReg(XPAR_CONT_IP_0_C00_AXI_BASEADDR, CONT_IP_C00_AXI_SLV_REG0_OFFSET, cuenta_incrementa);
        break;
    case '6':
        cuenta_incrementa = 0;
        CONT_IP_mWriteReg(XPAR_CONT_IP_0_C00_AXI_BASEADDR, CONT_IP_C00_AXI_SLV_REG0_OFFSET, cuenta_incrementa);
        break;
    default:
        break;
    }
```

Figura 8. Código en C para comunicación UART y escritura de registro.

Implementación.

En la Figura 9 se observa el esquema de la implementación final.

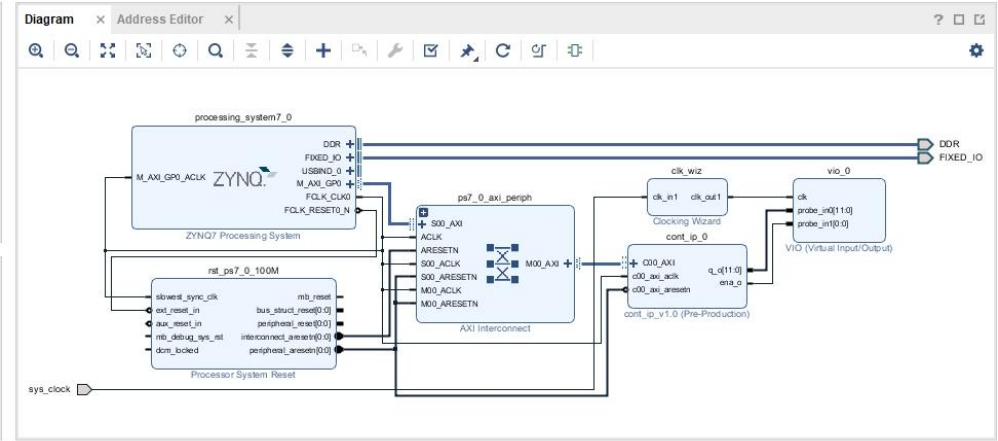

Figura 9. Esquema de la implenatción final del proyecto.

Para observar la cuenta del contador se implemento un VIO, cuyo clk se conecta a un reloj que es independiente del micro.

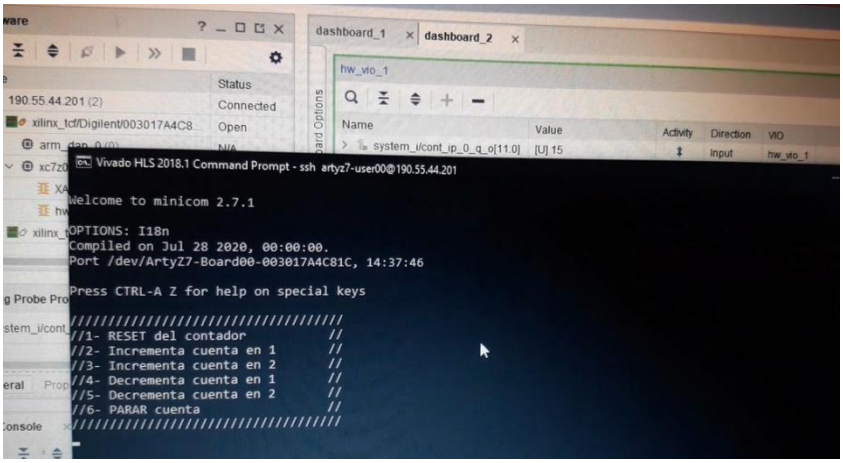

Figura 10. Captura de pantalla de la implementación programada de forma remota.

Problemas en el Desarrollo.

Inicialmente se desarrolló una IP para el contador implementado en la materia Circuitos Lógicos Programables (CLP), cuya estructura se observa en el siguiente código:

```
entity counter is
    port(
        clk_pin:      in std_logic;
        rst:          in std_logic;
        input:   in std_logic_vector(7 downto 0);    --
        data_rdy:     in std_logic;
        counter_data: out std_logic_vector(11 downto 0)
        );
end;
```

```
architecture counter_arch of counter is
    signal count: natural range 0 to 4095 := 0;
begin
    stm: process(clk_pin)
    begin
        if rising_edge(clk_pin) then
            if (rst = '1') then
             count <= 0;
             elsif data_rdy = '1' then --detecta pulso
                 if input = "01001001" then -- char 'I' (INCREMENTA)
                 count <= count+1;
                 elsif input = "01000100" then -- char 'D' (DECREMENTA)
                 count <= count-1;
                 end if;
             end if;
        end if;
    end process;

    counter_data <= std_logic_vector(to_unsigned(count, 12));

end counter_arch;
```

Figura 11. Programa en VHDL del Contador desarrollado en la materia CLP.

Para el código implementado era necesario un pulso en la entrada data_rdy, el cual permite que la cuenta se incremente en una unidad. Si en el siguiente pulso de reloj, la variable se mantiene en uno y no cambia la entrada de ocho bits ingresada por UART, la cuenta podría seguir incrementándose. Este inconveniente se observó al realizar el control por medio de la escritura de registro. En la Figura 12 se observa el control del contador utilizando los registros slv_reg0, slv_reg1 y slv_reg2.

```
if ((byte & 0x02)){
    rst=0;
    CONTADOR_IP_mWriteReg(XPAR_CONTADOR_IP_0_S00_AXI_BASEADDR, CONTADOR_IP_S00_AXI_SLV_REG0_OFFSET, rst);
    data_rdy=1;
    CONTADOR_IP_mWriteReg(XPAR_CONTADOR_IP_0_S00_AXI_BASEADDR, CONTADOR_IP_S00_AXI_SLV_REG2_OFFSET, data_rdy);
    entrada=73;
    CONTADOR_IP_mWriteReg(XPAR_CONTADOR_IP_0_S00_AXI_BASEADDR, CONTADOR_IP_S00_AXI_SLV_REG1_OFFSET, entrada);
    data_rdy=0;
    CONTADOR_IP_mWriteReg(XPAR_CONTADOR_IP_0_S00_AXI_BASEADDR, CONTADOR_IP_S00_AXI_SLV_REG2_OFFSET, data_rdy);
    sleep(1);
}
```

Figura 12. Escritura de registros para el control del Contador desarrollado en la materia CLP.

El registro slv_reg2 controla la entrada data_rdy y en el registro slv_reg1, se escribe la información de 8 bits que ingresa por la UART. Se observa que las cuentas se incrementan con pasos de 43, por lo que el contador desarrollado en la materia CLP, resulta ser muy rápido para realizar un control por medio de la escritura de registros.

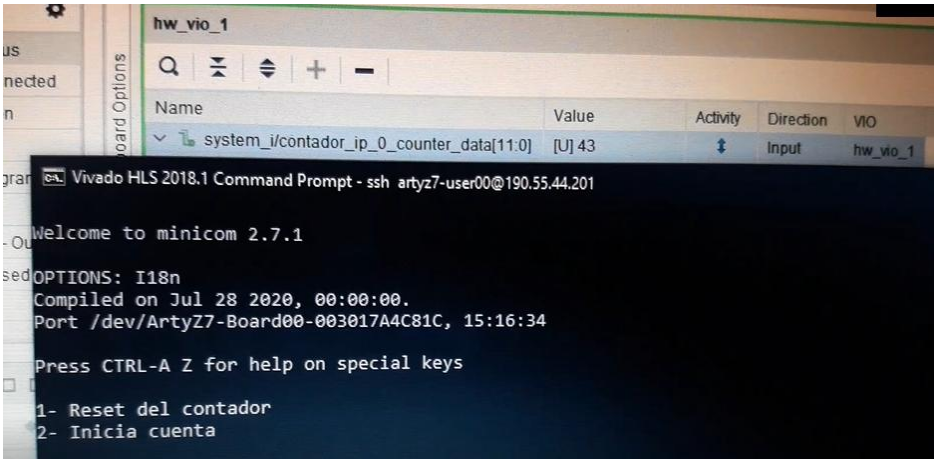

Figura 12. Captura de pantalla de la implementación programada de en forma remota.

Conclusiones.

El objetivo de este trabajo es implementar un contador Up-Down de 12 bits controlado por UART. El sistema digital se implementó en una FPGA Arty Z7-10, utilizando un contador de 12 bits desarrollado en código VHDL y un microcontrolador embebido ARM Cortex A9. Se realizaron simulaciones utilizando el software GHDL y GTKwave. Para el diseño se desarrollo una IP que se vincula al BUS AXI denominado count_ip. La información ingresada por UART, se relaciona con la escritura de registros que controlan el contador. Se realizaron pruebas del desarrollo de manera remota. El contador desarrollado en la materia CLP, resultó ser muy rápido para realizar un control por medio de la escritura de registros

Anexo

Código para correr el ensayo por medio de GHDL y GTKwave.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity cont12b_Lento_tb is
```

```
end;

architecture cont12b_Lento_tb_arq of cont12b_Lento_tb is

	component cont12b_Lento is
		generic(
		N: natural := 12;
		CICLOS: natural := 2
		);
		port(
			clk_i: in std_logic;
			rst_i: in std_logic;
			ena_i: in std_logic;
			step_i  : in std_logic;
			up_down : in std_logic;
			q_o: out std_logic_vector(N-1 downto 0);
			ena_o: out std_logic
		);
	end component;

	constant N_t: natural := 12;
	constant CICLOS_t: natural := 2;
	signal clk_t: std_logic:= '0';
	signal rst_t: std_logic:= '1';
	signal ena_t: std_logic:= '1';
	signal step_t: std_logic:= '0';
	signal up_down_t : std_logic:= '0';
	signal q_t: std_logic_vector(N_t-1 downto 0);
	signal ena_o_t: std_logic;
	begin
		clk_t <= not clk_t after 10 ns;
		rst_t <= '0' after 50 ns;
		ena_t <= '0' after 620 ns, '1' after 800 ns;
		step_t <= '1' after 500 ns,'0' after 1000 ns,'1' after 1500 ns;
		up_down_t <= '1' after 1300 ns;


		int_cont: cont12b_Lento
		generic map(N_t)
		port map(
			clk_i => clk_t,
			rst_i => rst_t,
			ena_i => ena_t,
			step_i => step_t,
			up_down => up_down_t,
			q_o => q_t,
			ena_o => ena_o_t
			);
	end;
```