Generador de señal senoidal con salida PWM y frecuencia variable

Carrera de especialización en sistemas embebidos
Microarquitecturas y softcores

Hernán Gomez Molino
(hernangomezmolino@gmail.com)
Octubre 2022

Introducción

El trabajo final se encuentra basado en el módulo desarrollado para la materia **Circuitos lógicos programables**, que consiste en un controlador por modulación de ancho de pulsos (PWM) capaz de comandar una etapa de potencia (puente H) y generar una salida senoidal.

En la figura 1 se observa el diagrama de bloques del sistema implementado.

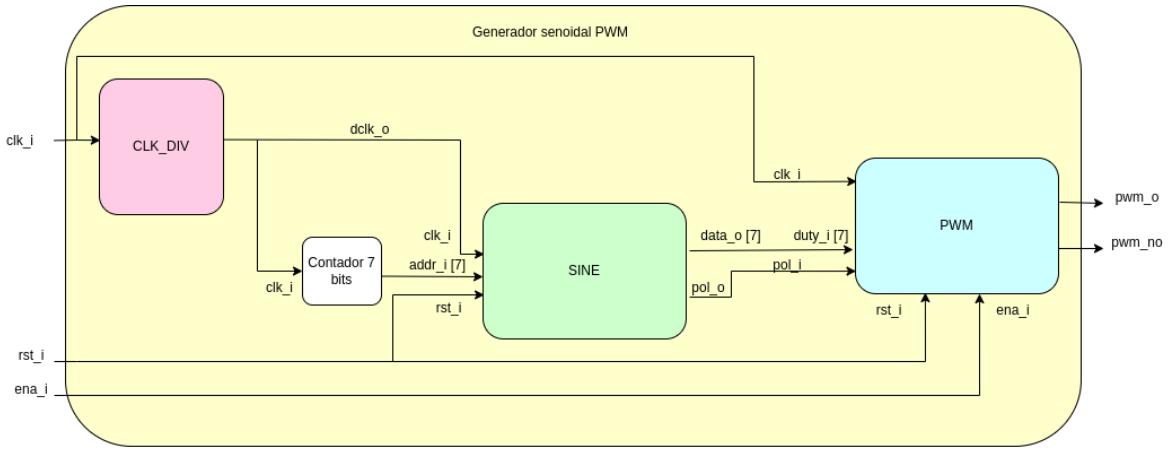

Figura 1: Diagrama en bloques del generador PWM

El objetivo del diseño es utilizar este sistema como bloque constitutivo y agregar los componentes necesarios para comandar las entradas desde el microcontrolador del SOC y manejar su funcionamiento completo, incluyendo la variación de la frecuencia de salida.

Hardware del sistema

Los bloques a agregar se describen a continuación y se muestran en la figura 2:

- VAR-CLK-DIV: Es el componente encargado de dividir la señal de reloj, según el valor recibido en su entrada.
- CONTROL-REG: Es la interfaz con el microcontrolador y posee los registros que se editan desde el software.
- ILA: Es el bloque que se utiliza para visualizar las señales.

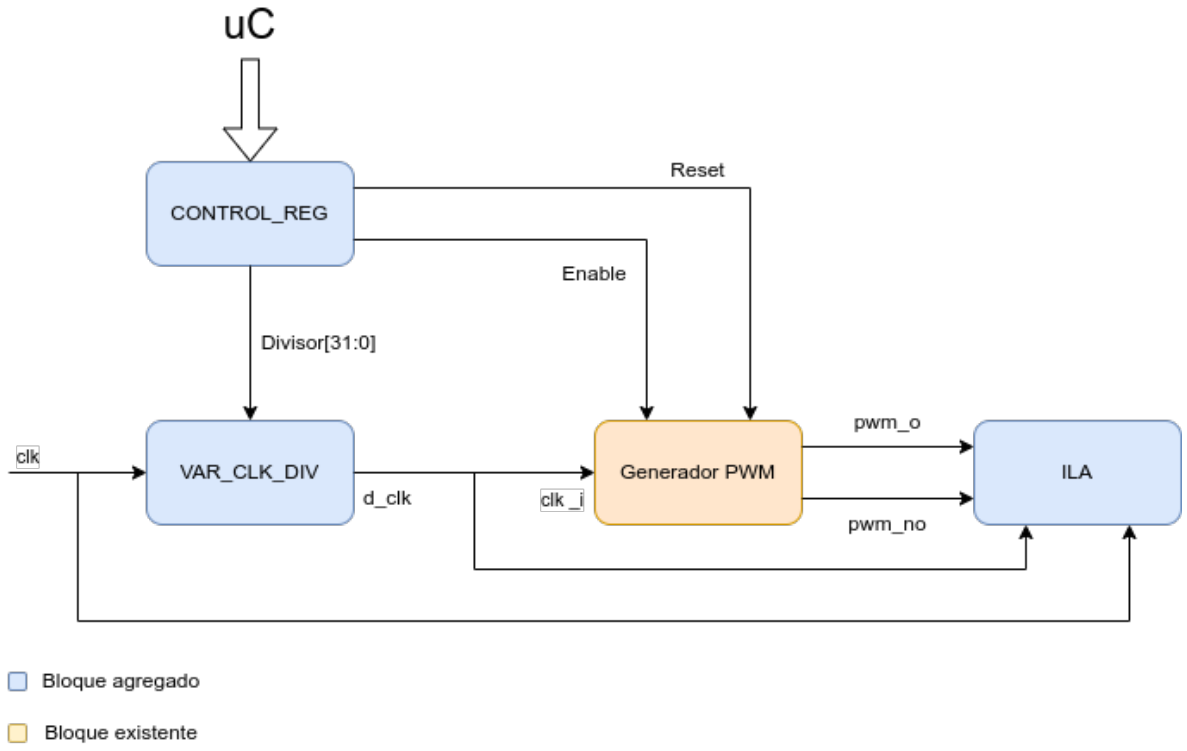

Figura 2: Diagrama en bloques del sistema completo

Bloque VAR-CLK-DIV

Entradas y salidas

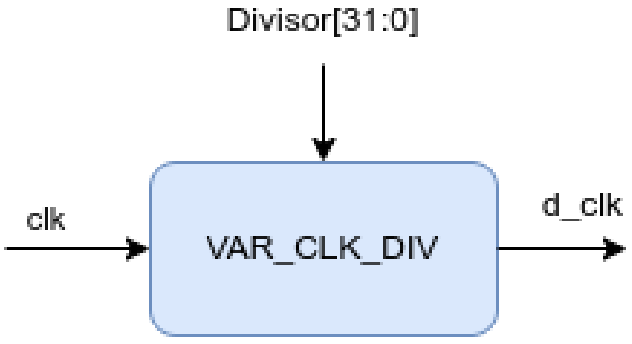

Figura 3: Bloque Divisor de Reloj

Funcionamiento

Este componente cuenta los pulsos de reloj en su entrada (en una señal) y asigna un bit a la salida, dependiendo del valor div-i de entrada. Al cambiar de un bit al siguiente se divide la frecuencia a la mitad.

Descripción VHDL

A continuación se muestra el código VHDL del bloque:

var_clk_div.vhd

/home/hernan/var_clk_div/var_clk_div.srcs/sources_1/imports/Trabajo final/var_clk_div.vhd

```
1  library IEEE;
2  use IEEE.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  entity var_clk_div is
6      port ( clk_i : in  std_logic;
7             div_i : in  std_logic_vector(31 downto 0);
8             dclk_o: out std_logic);
9  end var_clk_div;
10
11 architecture var_clk_div_arch of var_clk_div is
12 signal clk_div_s : std_logic_vector (7 downto 0):="00000000";
13 begin
14     -- clock divider
15     process (clk_i)
16     begin
17         if (clk_i'Event and clk_i = '1') then
18             clk_div_s <= std_logic_vector(unsigned(clk_div_s) + 1);
19         end if;
20
21         if div_i(0) = '1' then  dclk_o <= clk_div_s(1);
22         elsif div_i(1) = '1' then  dclk_o <= clk_div_s(2);
23         elsif div_i(2) = '1' then  dclk_o <= clk_div_s(3);
24         elsif div_i(3) = '1' then  dclk_o <= clk_div_s(4);
25         elsif div_i(4) = '1' then  dclk_o <= clk_div_s(5);
26         elsif div_i(5) = '1' then  dclk_o <= clk_div_s(6);
27         elsif div_i(6) = '1' then  dclk_o <= clk_div_s(7);
28         else dclk_o <= clk_div_s(1);
29         end if;
30
31     end process;
32
33
34
35 --    case to_integer(unsigned(div_i)) is...
45
46 end var_clk_div_arch;
47
```

Figura 4: Código VHDL del Divisor de Reloj

Bloque CONTROL-REG

Entradas y salidas

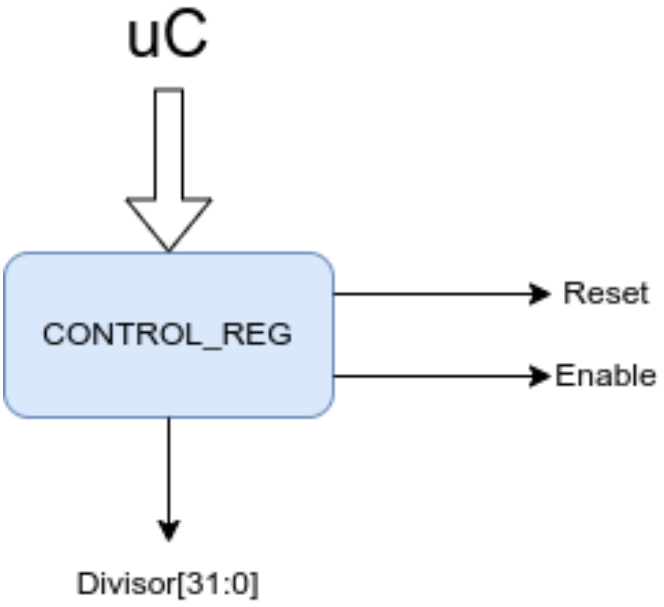

Figura 5: Bloque de Registros de Control

Funcionamiento

El funcionamiento puede resumirse diciendo que el bloque tiene los registros que sirven de entrada a los demás componentes y que los mismos son cargados desde el microcontrolador a través del bus AXI, sirviendo de interfaz entre el software y el hardware.

Descripción VHDL

A continuación se muestra parte del código VHDL del bloque:

```
-- Instantiation of Axi Bus Interface S00_AXI
control_reg_v1_0_S00_AXI_inst : control_reg_v1_0_S00_AXI
    generic map (
        DEFAULT_DIV => DEFAULT_DIV,
        C_S_AXI_DATA_WIDTH  => C_S00_AXI_DATA_WIDTH,
        C_S_AXI_ADDR_WIDTH  => C_S00_AXI_ADDR_WIDTH
    )
    port map (

        div_o => div_o,
        ena_o => ena_o,
        rst_o => rst_o,
        S_AXI_ACLK  => s00_axi_aclk,
```

Figura 6: Parte de Código VHDL - Bloque de Registros de Control

Bloque ILA (Integrated logic Analyzer

Entradas y salidas

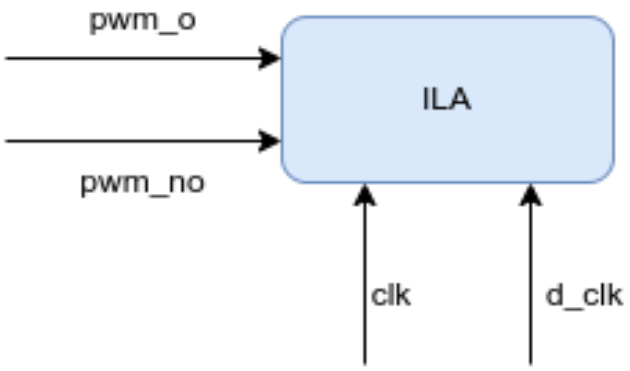

Figura 7: Bloque ILA

Funcionamiento

Este bloque se genera automaticamente y sirve para monitorear señales a traves del tiempo. En este caso particular se ultiliza para visualizar las señales de reloj, reloj dividido y ambas salidas PWM.

Configuración

A continuación se muestra la configuracón del bloque (figura 8):

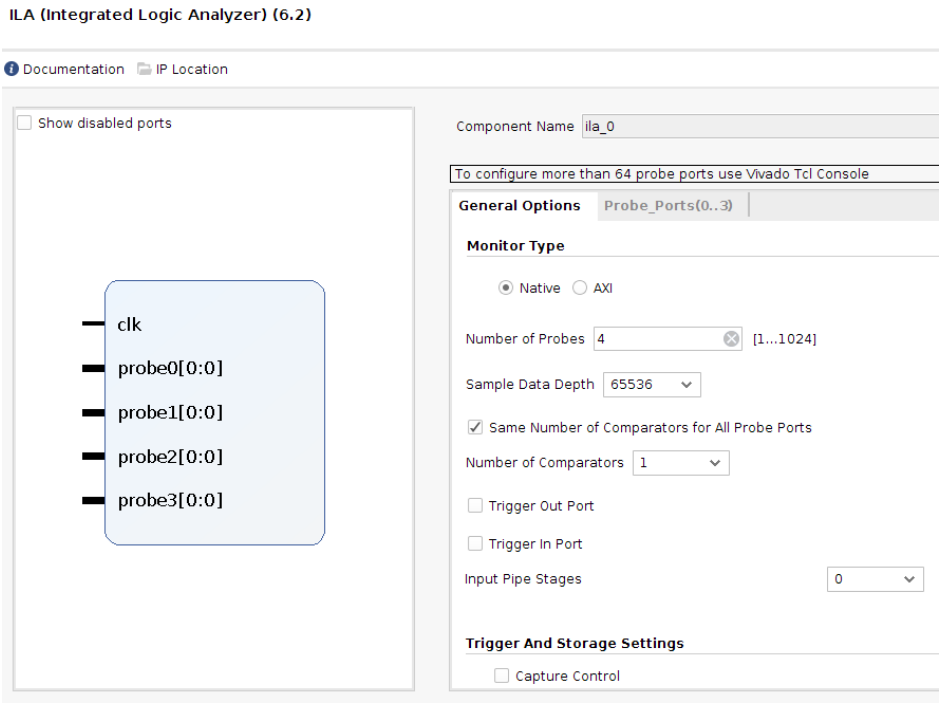

Figura 8: Configuración Bloque ILA

Sistema Implementado

A continuación se muestra el diseño de bloques implementado con el software Vivado (figura 9):

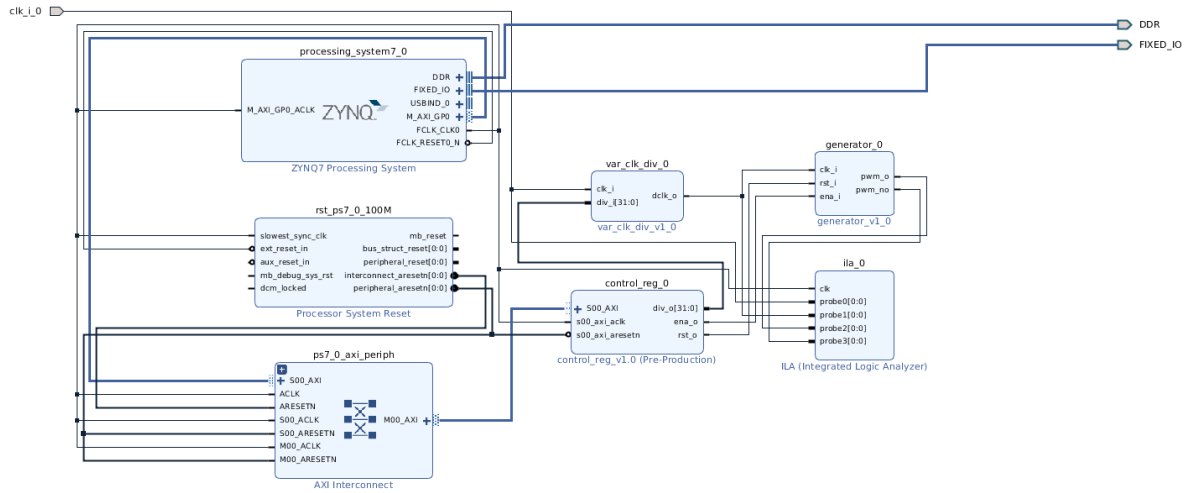

Figura 9: Diseño de Bloques

Software del sistema

Descripción

Se trata de programa desarrollado en lenguaje C, que utiliza los módulos y drivers generados automaticamente por la herramienta Vivado.

Esta desarrollado siguiendo el patrón conocido como superloop y su función principal es la de actualizar los registros de control de acuerdo a los parámetros recibidos via UART. Estos condicionan el funcionamiento del hardware descripto del sistema.

Código C

```c
int main()
{
    int byte = 0;
    int enable = 0;
    int reset = 0;
    int divisor = 0;

    init_platform();

    print("Generador PWM\n\r");
    print("\n\r");
    print("1. -> Activar ENABLE\n\r");
    print("2. -> Desactivar ENABLE\n\r");
    print("3. -> Activar RESET\n\r");
    print("4. -> Desactivar RESET\n\r");
    print("5. -> Dividir frecuencia\n\r");

    while(1)
    {
        byte = XUartPs_RecvByte(XPAR_PS7_UART_0_BASEADDR);
        if (byte != 0)
        {

            switch(byte)
            {
                case ASCII_1:
                    CONTROL_REG_mWriteReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG1_OFFSET, 1);
                    break;

                case ASCII_2:
                    CONTROL_REG_mWriteReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG1_OFFSET, 0);
                    break;
```

Figura 10: Software - Parte 1

```
                case ASCII_3:

                    CONTROL_REG_mWriteReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG2_OFFSET, 1);
                    break;

                case ASCII_4:
                    CONTROL_REG_mWriteReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG2_OFFSET, 0);
                    break;

                case ASCII_5:
                    divisor = CONTROL_REG_mReadReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG0_OFFSET);
                    if(divisor < 8)
                        divisor *=2;
                    else
                        divisor = 1;
                    CONTROL_REG_mWriteReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG0_OFFSET, divisor);
                    break;

                default:

                    enable = CONTROL_REG_mReadReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG1_OFFSET);
                    reset = CONTROL_REG_mReadReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG2_OFFSET);
                    divisor = CONTROL_REG_mReadReg(XPAR_CONTROL_REG_0_S00_AXI_BASEADDR, CONTROL_REG_S00_AXI_SLV_REG0_OFFSET);
                    printf(" ENABLE = %d - RESET = %d - DIVISOR = %d\n\r", enable, reset, divisor);
            }
            byte = 0;
        }
    }

    cleanup_platform();
    return 0;
}
```

Figura 11: Software - Parte 2

Test

En la captura de pantalla se puede observar la salida graficada por el bloque ILA.

En la parte superior de la imagen se ven las dos señales de reloj (normal y dividida). Debajo se observa la señal PWM perteneciente a cada uno de los grupos de transistores de potencia, variando su ancho en seguimiento de la amplitud de la señal seno.

Figura 12: Sistema en funcionamiento