Microarquitecturas y Softcores
CESE 2024
Prof: Ing. Nicolás Álvarez
Alumno: Rubén Mansilla

Resumen del proyecto Sistema PWM IP

Este proyecto toma como base el proyecto desarrollado en CLP: pwmModule cuyo diagrama de bloques es el que se muestra en la figura 1.

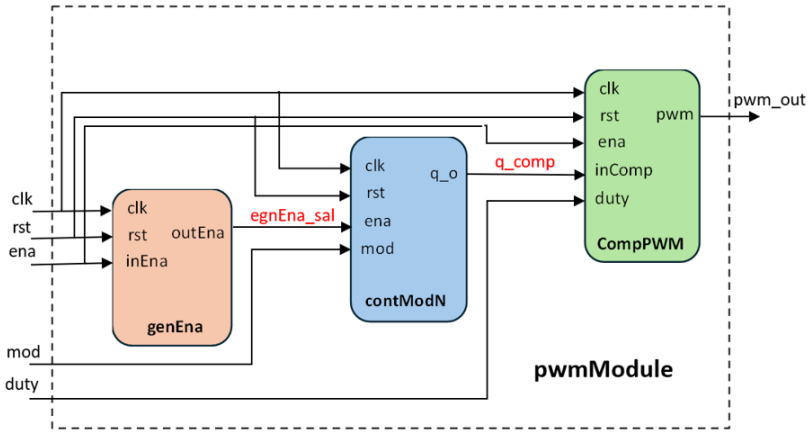

Figura 1

Para este proyecto creamos un *IP Core* con nuestro modulo pwm al que llamaremos *pwm_ip*.

Este IP Core personalizado usa 3 de los 4 registros, del AXI bus, que ofrece Vivado como requerimiento mínimo para la creación de *IP cores*. Estos tres registros se configuran como entrada de datos al IP Core y se usan de la siguiente manera:

Registro 0 → **Mod**: Recibe el valor del **módulo** de la señal pwm.
Registro 1 → **duty**: recibe el valor del **duty cycle** de la señal pwm.
Registro 2 → **ena**: recibe el valor de **enable** del módulo pwm.

En cambio, su salida se saca directamente del *pwm_ip* como una señal de 1 bit llamada **salPwm**.

Las señales de entrada de **clock** y **reset** se conectan directamente con las mismas señales del sistema y se manejan desde el mismo.

Podemos ver como queda el *pwm_ip* en la figura 2.

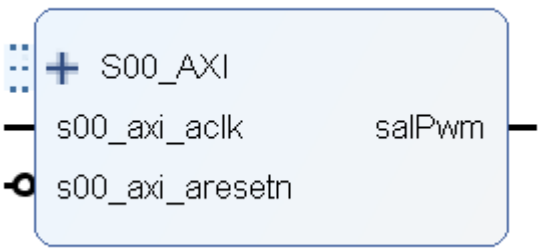

Figura 2

En la siguiente figura vemos algunos detalles de sus puertos en interfaces, figura 3.

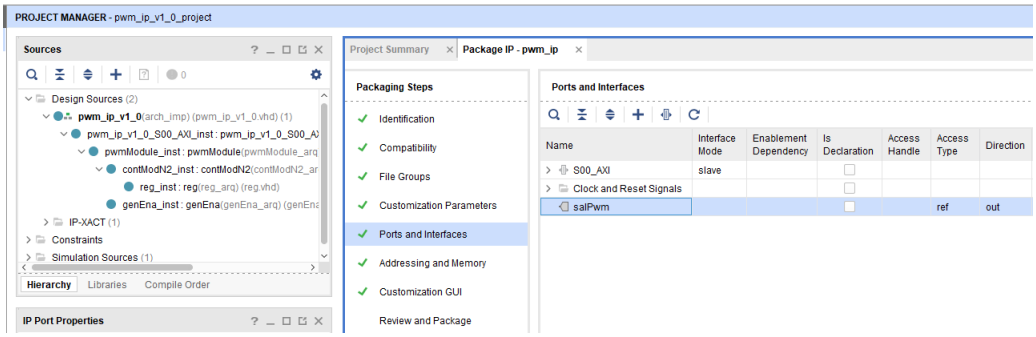

Figura 3

El modulo sintetiza sin errores y se empaqueta exitosamente.

A continuación creamos el proyecto el proyecto al que llamamos *sistema_pwmModule*.

Incluimos los IP Cores necesarios para su funcionamiento:

> **pwm_ip**→ llamado pwm_ip_0 del módulo pwm que es el corazón del proyecto.
>
> **Zynq Proccesing System** (micro) →llamado Processing_system7_0 que luego generará los módulos de Interconexion del AXI Bus y el Procesador de Reset del Sistema.
>
> **ILA** (Integrated Logic Analyzer)→ llamado *ila_0* que tomará la salida del modulo pwm y mostrará su forma de onda y variación desde vivado.

En la figura 4 podemos ver como queda conectado y armado el diagrama de bloques del proyecto. Todos los IP Cores se conectan de manera automática, excepto la conexión entre **salPwm** del *pwm_ip* y la entrada de prueba **probe0** del *ila*, que se hace de forma manual. Se puede ver en la figura 4 el diagrama de bloques del proyecto.

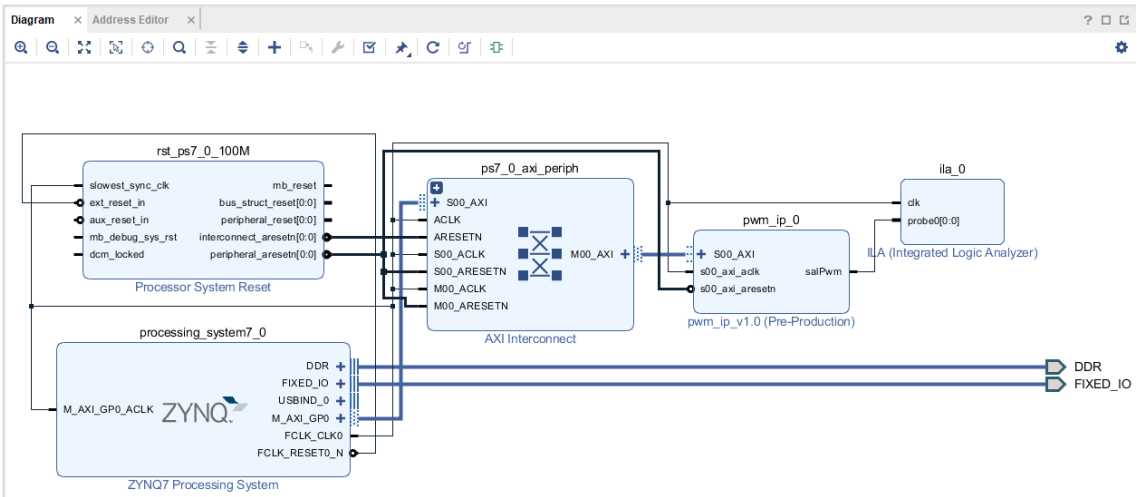

Figura 4

Efectuamos la validación. En **Sources**, y sobre *sistema_pwmModule.bd*, generamos los *productos de salida* para generar los archivos de implementación, simulación y síntesis para el diseño. Luego, también sobre *sistema_pwmModule.bd*, creamos el *wrapper*, para generar el

modelo top-level de VHDL. Se puede ver en la figura 5 como queda la estructura de archivos del proyecto.

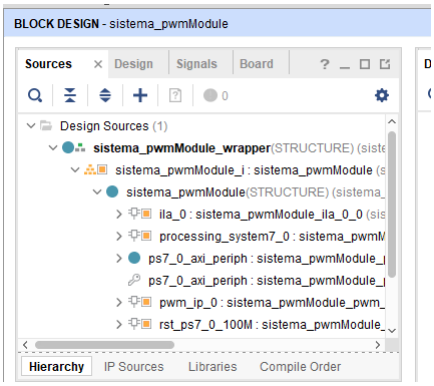

Figura 5

Luego, generamos el **bitstream**. Se crea exitosamente, ver figura 6.

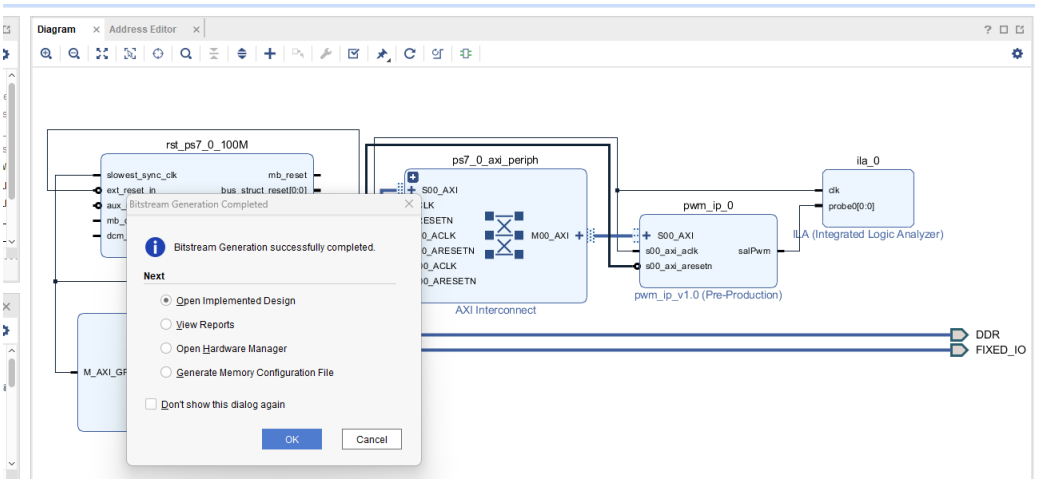

Figura 6

La idea es ahora crear un proyecto en SDK que me permita correr una aplicación en C que cargue los valores de **modulo**, **duty cycle** y **enable** que recibirá el micro de la placa y los pasará al pwm_ip por los registros correspondientes del AXI Bus. Luego, desde Vivado vemos la salida de pwm en la ventana del ILA.

Para lograr esto, exportamos el hardware incluyendo el *bitstream* y lanzamos el SDK. Ver figura 7.

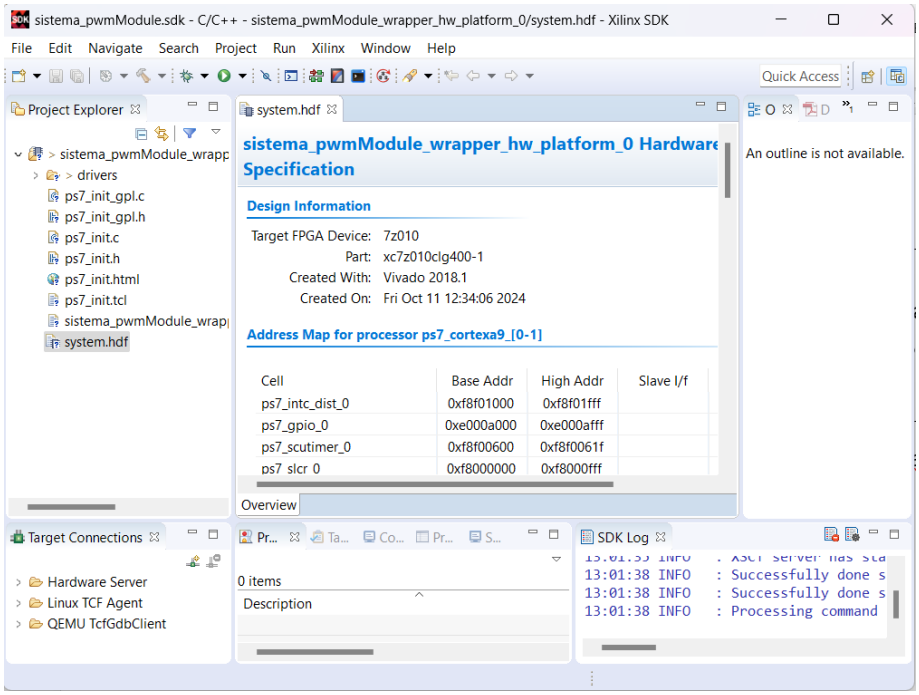

Figura 7

Creamos una nueva aplicación vacía a la que llamamos pwm e importamos el archivo *pwm.c* que tomamos como base de nuestra aplicación. Ver figura 8.

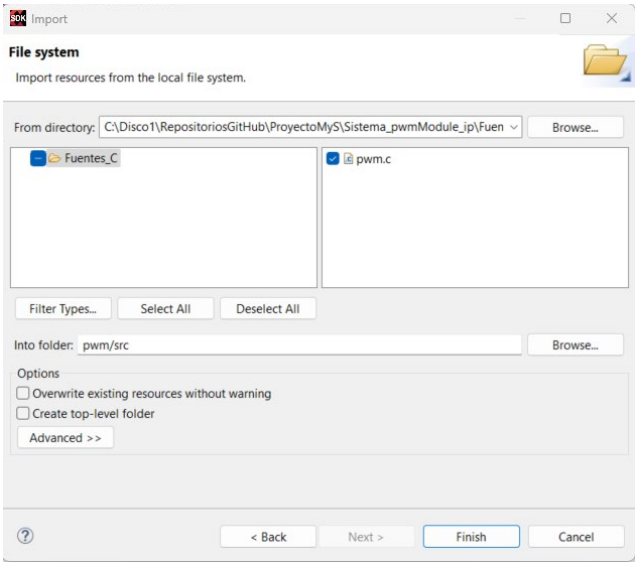

Figura 8

Esta aplicación se tomó de otra desarrollada en clase y se la ajusto a los parámetros usados en nuestro proyecto consultando las etiquetas que usa como BASE ADDRES y OFFSET de cada registro del AXI Bus que usaremos y el nombre de las funciones de lectura de estos puertos. Esta info se encuentra en los archivos incluidos al principio:

Xparameters.h
Xil_io.h
Pwm_ip.h

El archivo pwm.c queda de como se ve en la figura 9.

system.hdf | system.mss | pwm.c | xparameters.h | pwm_ip.h | xil_io.h

```c
#include "xparameters.h"
#include "xil_io.h"
#include "pwm_ip.h"


//================================================

int main (void) {

    //int pwm_out;
    int modulo = 15;
    int duty = 7;
    int enable = 1;
    //int i;

    xil_printf("-- Inicio del programa para validar el uso de IP cores propios --\r\n");

    PWM_IP_mWriteReg(XPAR_PWM_IP_0_S00_AXI_BASEADDR, PWM_IP_S00_AXI_SLV_REG0_OFFSET, modulo);
    PWM_IP_mWriteReg(XPAR_PWM_IP_0_S00_AXI_BASEADDR, PWM_IP_S00_AXI_SLV_REG1_OFFSET, duty);
    PWM_IP_mWriteReg(XPAR_PWM_IP_0_S00_AXI_BASEADDR, PWM_IP_S00_AXI_SLV_REG2_OFFSET, enable);
    //pwm_out = PWMMODULE_IP_mReadReg(XPAR_PWM_IP_0_S00_AXI_BASEADDR, PWMMODULE_IP_S_AXI_SLV_REG3_OFFSET);

    xil_printf("modulo, duty, enable: %d , %d , %d\r\n", modulo, duty, enable);


}
```

Figura 9

La aplicación escribe valores, “harcodeados”, sobre los registros del AXI Bus ingresando al módulo pwm los datos de **modulo**, **duty cycle** y **enable** únicamente ya que la salida pwm se podrá observar desde vivado en el ILA.

Los pasos que siguen, que serían la programación de la placa FPGA, correr la aplicación en el micro de la placa, ver que esta funciona y luego abrir el target desde Vivado para ver la salida pwm en el ILA se pueden seguir en el video cuyo lin se publica mas abajo.

Link al video de demostración:
https://drive.google.com/file/d/16E4peLYJTnFlt9qTe8HscNaMTx0JGNo3/view?usp=sharing