IPD432: Diseño Avanzado de Sistemas Digitales
Semestre II-2021
Guía de flujo de diseño usando Vivado y Vitis

17 de Diciembre de 2021

1. Objetivos.

- Complementar los contenidos estudiados en el curso para seguir un flujo de diseño que integra el diseño de hardware en HDL con software en un procesador.

2. Requerimientos

Para esta actividad es requerido lo siguiente:

- Vitis 2021.1.
- Vivado 2021.1.
- Python 3.
- PySerial.
- Soporte de tarjetas Zynq (disponible desde Vivado).

3. Contador de 4 bits.

Esta sección tiene por objetivo ilustrar con un ejemplo sencillo el flujo de diseño en FPGA utilizando diagramas de bloques correspondientes a IPs hechas por un tercero. Para iniciar, se debe clonar el siguiente repositorio: https://github.com/JuanjoV/IPD432_zybo_demo.

3.1. Vivado

Para comenzar, en Vivado creará un nuevo proyecto. Se sugiere usar la carpeta *Demo_zybo_vivado* para alojar el proyecto. No es necesario agregar ningún archivo al proyecto, sólo es importante seleccionar *RTL project* y elegir *Zybo* como tarjeta del proyecto. Note que esta debe aparecer con la

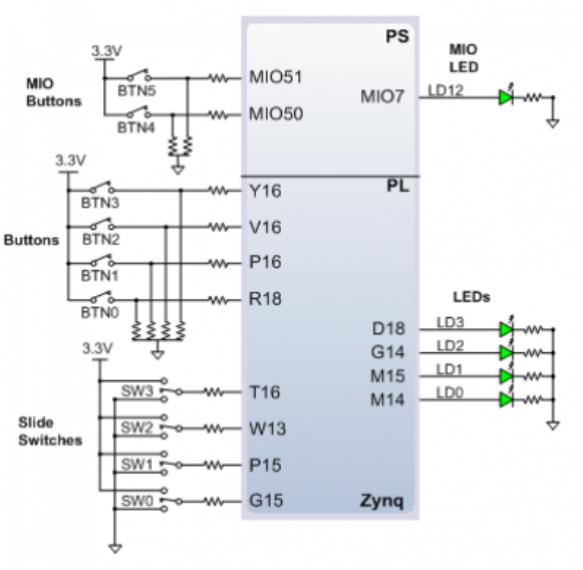

Figura 1: Esquemático de los leds de la tarjeta Zybo

columna *status* como *Installed*, de lo contrario, deberá instalarla haciendo click en el mismo ícono ubicado en esa columna [1].

Luego, en la ventana *Flow Navigator*, se elegirá la opción *Create Block Design*, dentro de *IP Integrator*. Con esto tendremos acceso a un diagrama de bloques vacío. Vivado cuenta con una variedad de bloques IP listos para ser instanciados [2], los que pueden ser explorados con el botón + ubicado en la parte superior.

Se agregará los siguientes bloques:

- 1 x ZYNQ7 Processing System
- 2 x AXI GPIO

A pesar de que el contador se hará por software, es necesario realizar estos pasos en hardware debido a que el procesador de Zybo no tiene acceso a los periféricos directamente, como se ve en la Figura 1, por lo que es necesario que estos pasen por la FPGA.

Los bloques AXI GPIO son controladores para los periféricos, ya que, como muestra el datasheet, no es posible mapear uno a uno los pines correspondientes a leds y botones.

Los bloques instanciados pueden ser configurados haciendo doble click sobre ellos. Renombre los bloques AXI GPIO, se sugiere usar *GPIO_leds* y *GPIO_btns*.

En la parte superior de la ventana, elija la opción *Run Block Automation*. Deje la configuración sugerida por defecto y haga click en OK. Esto realiza las conexiones requeridas para el procesador. Luego, realice la automatización de las conexiones de los bloques haciendo click en *Run Connection Automation* .

[1] [Curiosidad] Este es un cambio relativamente reciente de Vivado, antiguamente debían descargarse manualmente los archivos y copiarse en el directorio de Vivado

[2] Algunos requieren de una licencia adicional

Se mostrará una ventana con una lista de los bloques y sus conexiones configurables por la herramienta. Cada bloque AXI GPIO contiene una interfaz GPIO y una interfaz AXI, configure las interfaces externas de cada uno dejando una como *btns_4bits* y la otra como *leds_4bits*. Sea consistente con el nombre que asignó a los bloques. Luego, marque todas las checkboxes y haga click en OK.

Note que además del bus AXI, se conectó también una señal de reloj y un reset a cada bloque. El reloj conectado, por defecto es de 100 MHz y puede modificarse dentro de la configuración del procesador.

Ahora, habilite una señal de interrupción desde el bloque que controla los botones, entrando a la configuración de este y habilitando la opción *Enable Interrupt*. Luego, habilite las interrupciones desde el procesador, entrando a la configuración de este y luego al menú Interrupts. Habilite *Fabric Interrupts* y luego en *PL-PS Interrupt Ports* habilite *IRQ_F2F*. Finalmente, conecte manualmente los pines que aparecieron producto de ambas configuraciones.

Después de los pasos indicados, su diagrama de bloques debería verse como la Figura 2.

En la parte superior, además es posible editar las direcciones de memoria donde se ubicarán los controladores en la pestaña *Address Editor*. Verifique la consistencia del diagrama con la herramienta *Validate Design*, ubicada en el menú superior.

Para terminar el diseño, se creará el contenedor del diagrama de bloques entrando a la ventana *Sources* y seleccionando la opción *Create HDL Wrapper...* que aparecerá luego de hacer click derecho sobre el diagrama de bloques.

Ejercicio propuesto: Identifique en los archivos generados la señal de interrupción configurada en el bloque AXI GPIO asignado a los botones.

Para finalizar la parte correspondiente a Vivado, se debe generar el bitstream y luego exportar el hardware, incluyendo el bitstream. Como producto de esta acción, debe obtener un archivo `.xsa`.

Para la siguiente parte, puede iniciar Vitis desde la Vivado, haciendo click en *Tools* y luego en *Launch Vitis IDE*.

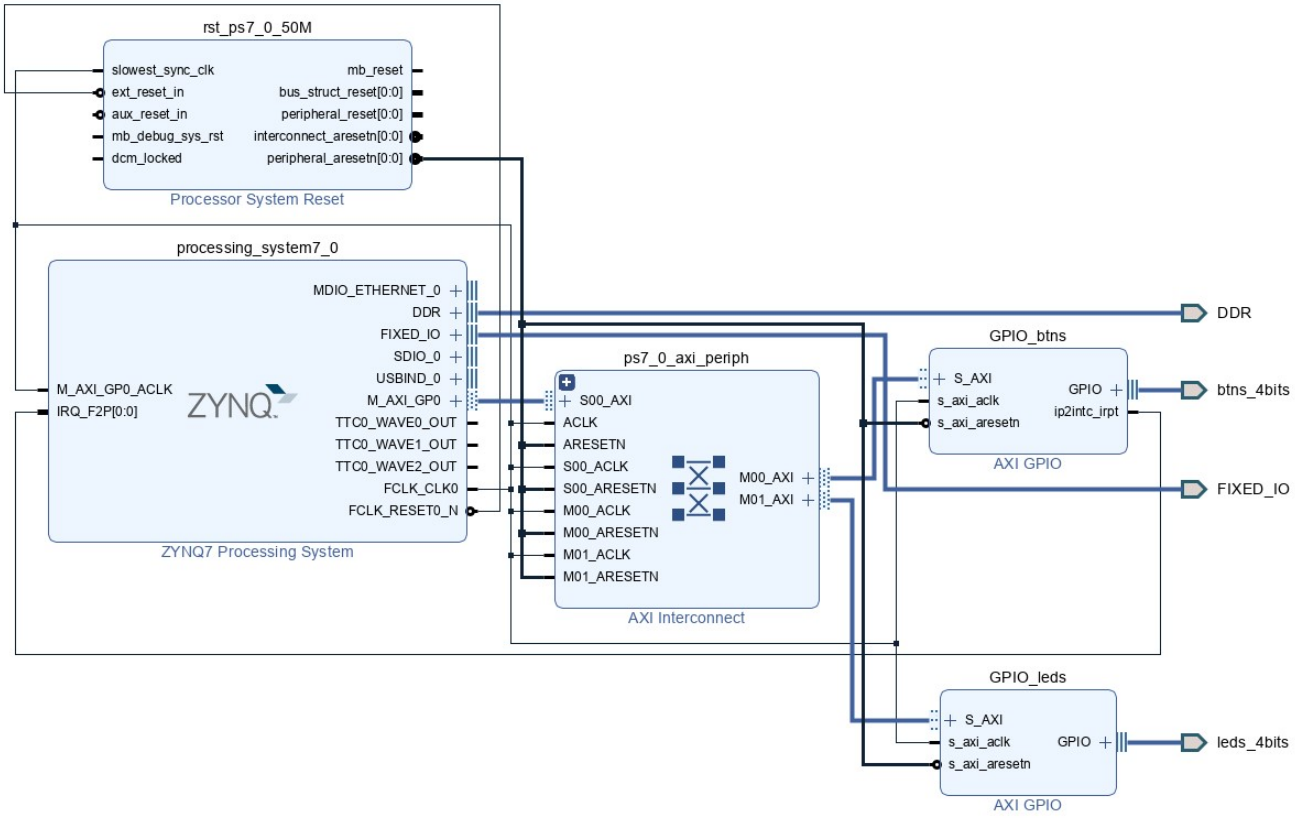

Figura 2: Diagrama de bloques para contador de 4 bits en Zynq

3.2. Vitis

Al iniciar Vitis, solicitará indicar un Workspace, a lo cual se debe elegir la carpeta dispuesta para Vitis en el repositorio *Demo_zybo_vitis*. En la ventana inicial, se elegirá la opción *Create Application Project*.

En la configuración del proyecto, busca el archivo `.xsa` generado en el punto anterior, dentro de la pestaña *Create a new platform from hardware*. Haz click en siguiente y luego indica un nombre para la aplicación. Posteriormente, en la sección de Dominio, el sistema operativo quedará con la opción *Standalone*. Para finalizar, elegiremos el template *Empty Application (C)*.

Una vez generado el proyecto, en el menú de proyecto, si se despliega el submenú con el nombre dado a la plataforma (por defecto, el nombre del `.xsa`), en el archivo *platform.spr* se puede ver la lista de los drivers disponibles para cada componente del diseño. Además, se encuentran links a las documentaciones y ejemplos de cada uno. Además, dentro de la carpeta *hw* se encuentra el archivo `.xsa` exportado de Vivado el cuál contiene las direcciones de memoria asignadas a cada componente, junto con una descripción de los registros de los IP del diseño, en este caso, los AXI GPIO.

Dentro del submenú de la aplicación, en la carpeta *src* haz click derecho e importa el archivo *interrupt_counter.c*. Luego, compila el proyecto.

Para terminar, se debe cargar el bitstream en la FPGA y luego el software. Lo primero se pue-

de hacer desde Vivado o desde Vitis, para esta último en el menú de aplicación se debe entrar en *Xilinx/Program Device*, dejar todo por defecto y hacer click en *Program*. Para cargar el software, en el submenú de aplicación (no sistema), se debe hacer click derecho y elegir *Run as* y luego seleccionar *Launch Hardware*.

4. Añadiendo IP personalizado

4.1. Exportado IP desde Vitis HLS

Se generará un IP personalizado sencillo utilizando Vitis HLS, en la carpeta *Demo_zybo_hls/*. Para crear el proyecto use el part number xc7z010-clg400-1 y considere los archivos *hls_main.c* y *hls_config.h* como archivos de diseño, y *testbench.c* y *goldenReference.dat* como archivos de testbench. El IP a diseñar es un sumador y promediador de elementos de vectores de largo N y de tipo `data_t`, ambos definidos en *hls_config.h*.

El archivo *goldenReference.dat* contiene 1000 vectores con su suma y promedio. La constante `TRIALS` en *testbench.c* indica cuantos de esos vectores se utilizan para realizar el test.

Realice el flujo completo de HLS, esto es, Simulación en C, luego Síntesis, Co-Simulación[3], terminando con *Export RTL*. En este último paso, se puede indicar la ubicación del IP exportado, por defecto *solution1/impl* con nombre *export.zip*. Este archivo se debe copiar en el directorio de Vivado y extraerse dentro de una carpeta.

4.2. Importando IP en Vivado

Considere el diseño realizado en la sección 3.1. Ingrese al *IP Catalog*, luego click derecho en la ventana mostrada y seleccione *Add Repository*. En la ventana emergente, seleccione la carpeta donde se extrajo el IP generado con HLS. Aparecerá el bloque diseñado con el nombre indicado al momento de exportarlo.

Ahora, vuelva al diagrama de bloques y agregue el IP personalizado. Luego de esto, deberá volver a correr *Run Connection Automation* y conectar el pin de interrupción del nuevo bloque. Para esto, se debe agregar un bloque *Concat*. El diagrama de bloques debe resultar como en la figura 3.

4.3. Añadiendo un pin externo

Para medir el tiempo que tarda el IP entre que se le da la señal *Start* y que entrega el dato, se puede hacer de forma externa señalizando el trabajo a través de un pin.

Para esto, se debe agregar un tercer AXI GPIO y señalar su salida como *custom*. Aparecerá un puerto marcado como *gpio_rtl*.

Para realizar la configuración, se debe entrar en *Open Elaborated Design* y luego en *IO Planning*. Esta mostrará una versión de alto nivel del archivo de constraints. Desde acá se puede asignar manualmente cada bit del puerto *gpio_rtl*.

[3]Para evitar esperar demasiado, se sugiere usar un `TRIALS <10`

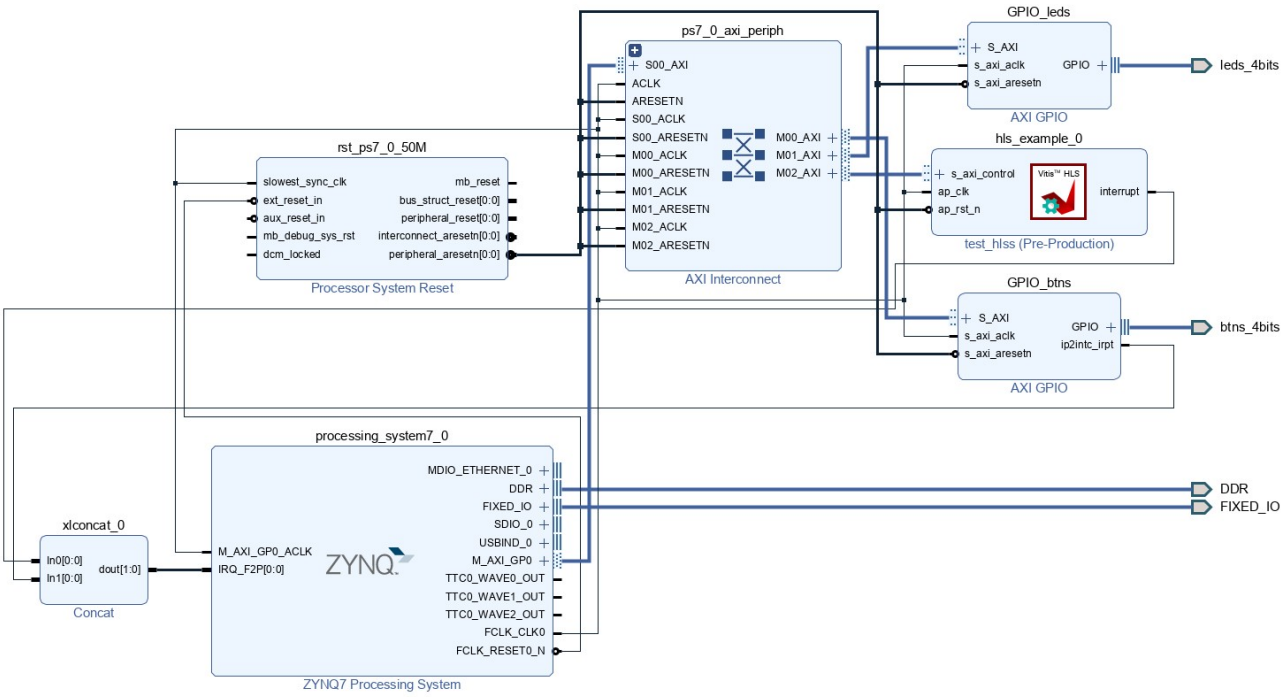

Figura 3: Diagrama de bloques con bloque IP personalizado.

Una vez lista la asignación, se debe guardar los constraints generados y volver a generar el archivo `.xsa`.

4.4. Usando el IP desde Vitis

Para esta sección se debe crear un nuevo proyecto de Vitis, de igual forma que el anterior, pero usando el nuevo archivo `.xdc`.

Esta vez, en Vitis, dentro del mismo directorio donde se encuentra el archivo `.xsa` aparecerá una carpeta llamada *drivers*, que contiene los drivers autogenerados por HLS para comunicarse con el IP creado. Basta con agregar estos drivers al código para poder usar el IP. Dentro del repositorio se añadirá el archivo *main.c*, que contiene el código para realizar la prueba del IP.

El test consta de dos códigos, uno cargado en la Zynq, que recibe y transmite datos por UART y el segundo, en PC que lee el archivo *goldenReference.dat*, envía los datos, recibe los resultados y los compara.

Programe la tarjeta Zybo y luego corra el script *Serialcmd.py*. Asegurese de editar este último configurando el puerto serial donde está la tarjeta conectada y la ubicación del archivo con los datos de referencia.

Use el analizador lógico para ver cuanto tiempo transcurre entre que se envía la señal de inicio y se reciben los datos.