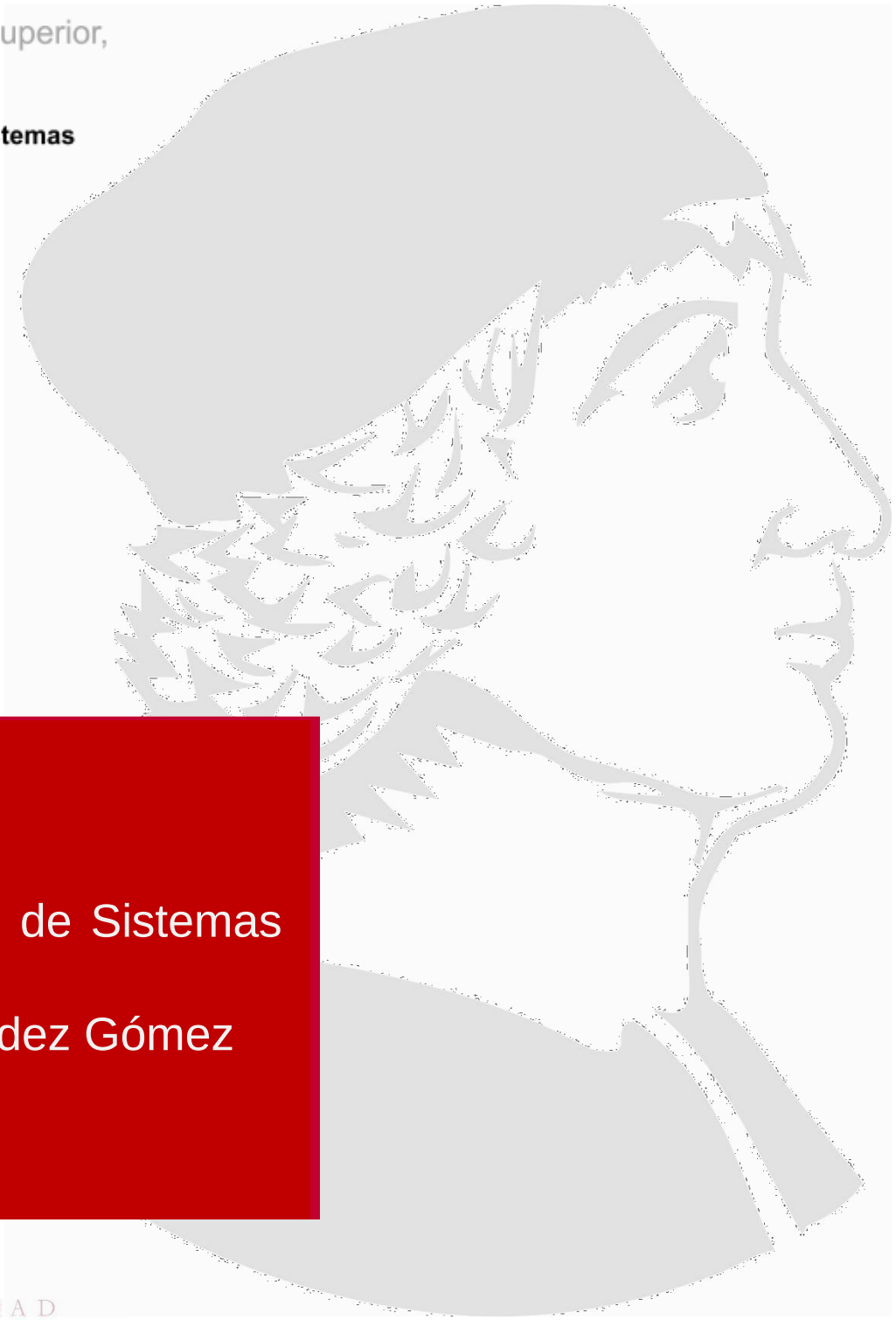
Escuela Politécnica Superior,
Grado en Informática

Diseño Automático de Sistemas
Prof: Luis Pérez Miguel



Práctica 4

Diseño Automático de Sistemas Fiables

José María Fernández Gómez
Álvaro Terrón
Gonzalo Vázquez

UNIVERSIDAD NEBRIJA

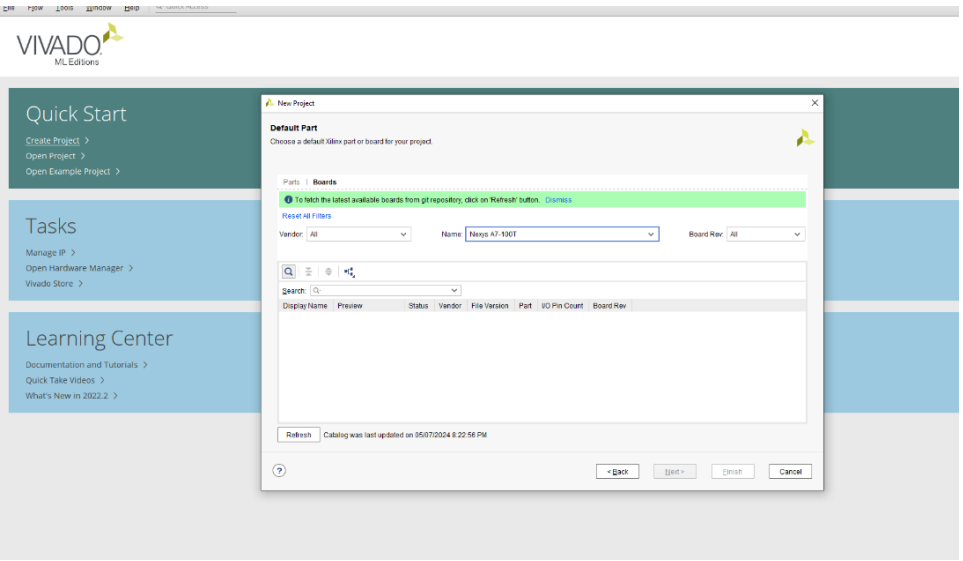
Objetivos De la Práctica

1. Implementar un procesador MicroBlaze en Vivado
2. Desarrollo software para un microcontrolador
3. Uso de la FPGA Nexys A7
4. Manejo de Vitis para desarrollo software

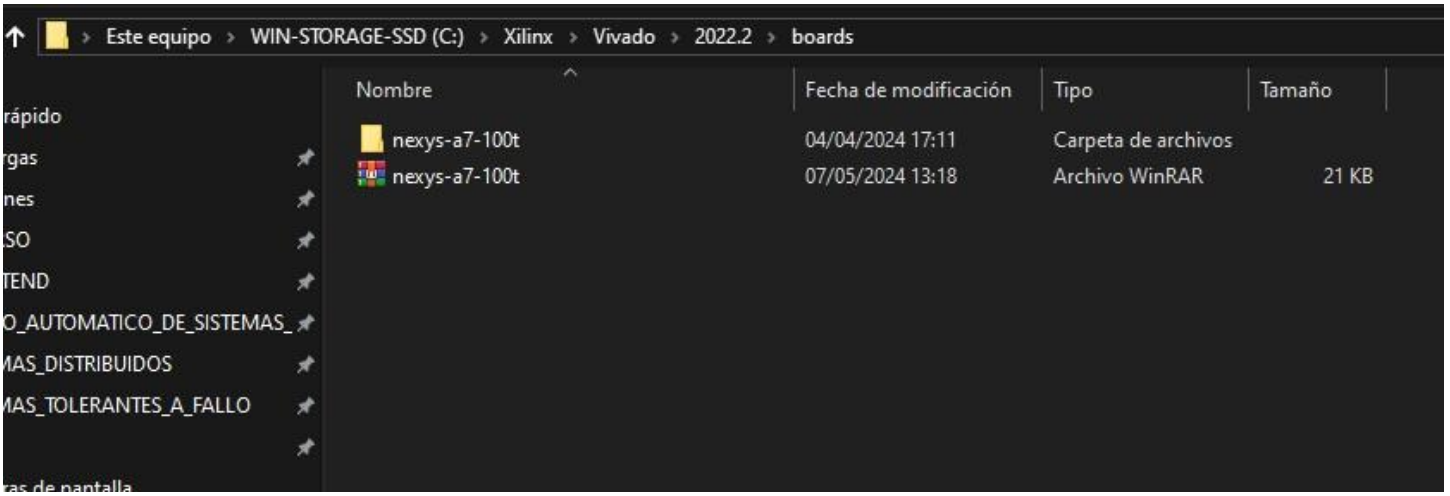
Índice

Creación del Proyecto

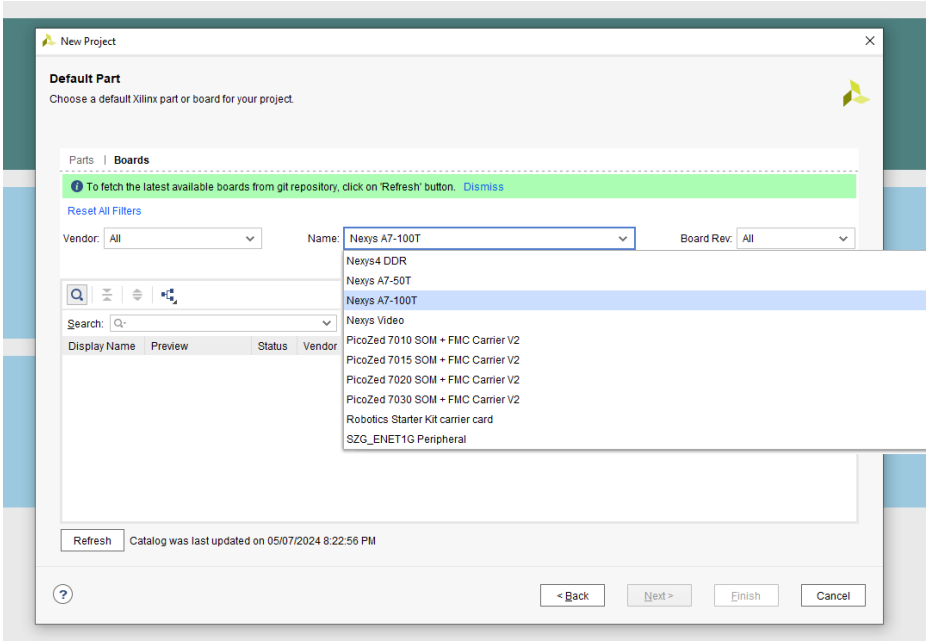
1. Se crea el proyecto de Vivado de manera normal hasta llegar a la pestaña de Default Part, donde tendremos que añadir la placa deseada, en este caso la Nexys A7



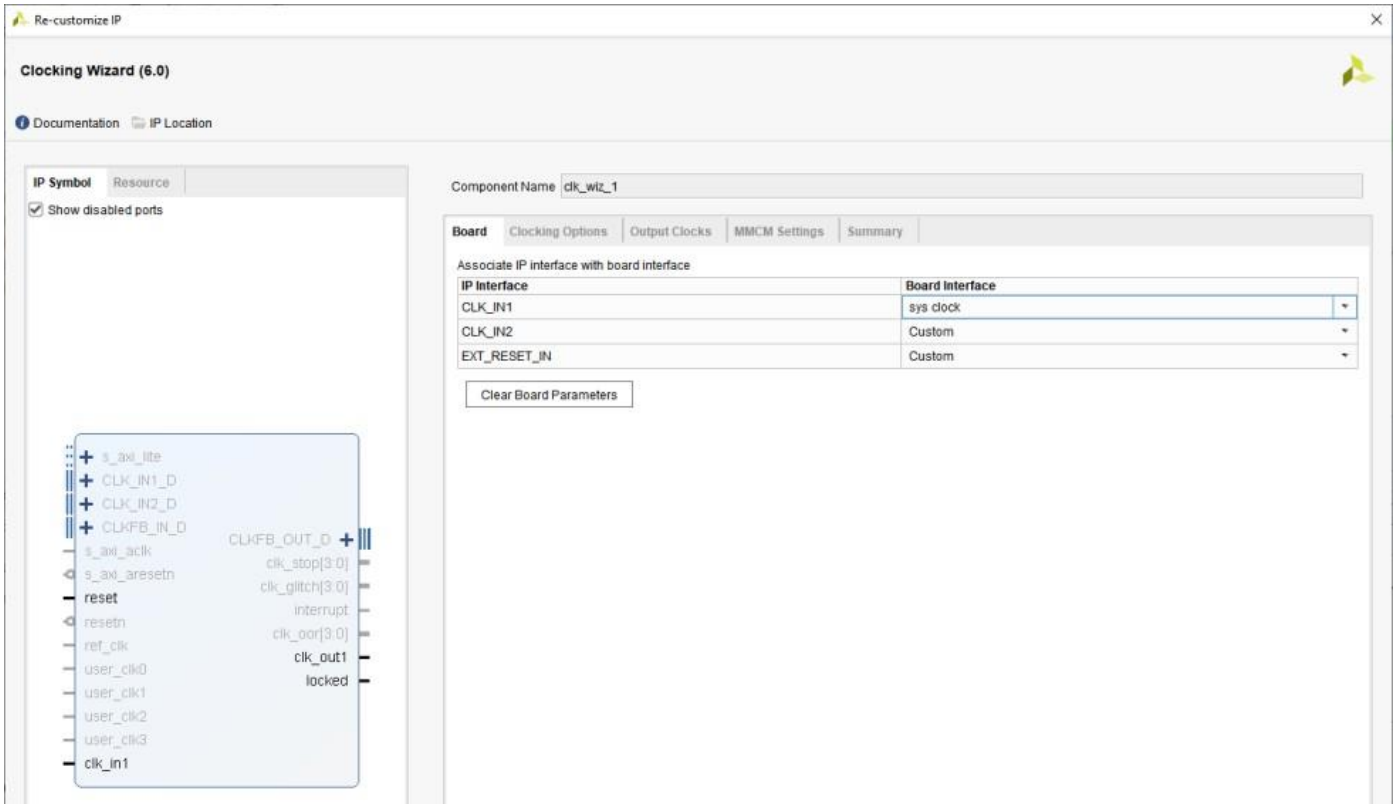
2. Lo más probable es que nuestro vivado no contenga los drivers ni la información de la placa, por lo que es necesario añadirlo manualmente, para esto, hay que descargar el archivo comprimido con la información y código operativo de la placa, y agregarla a la carpeta boards dentro del sistema de archivos de vivado



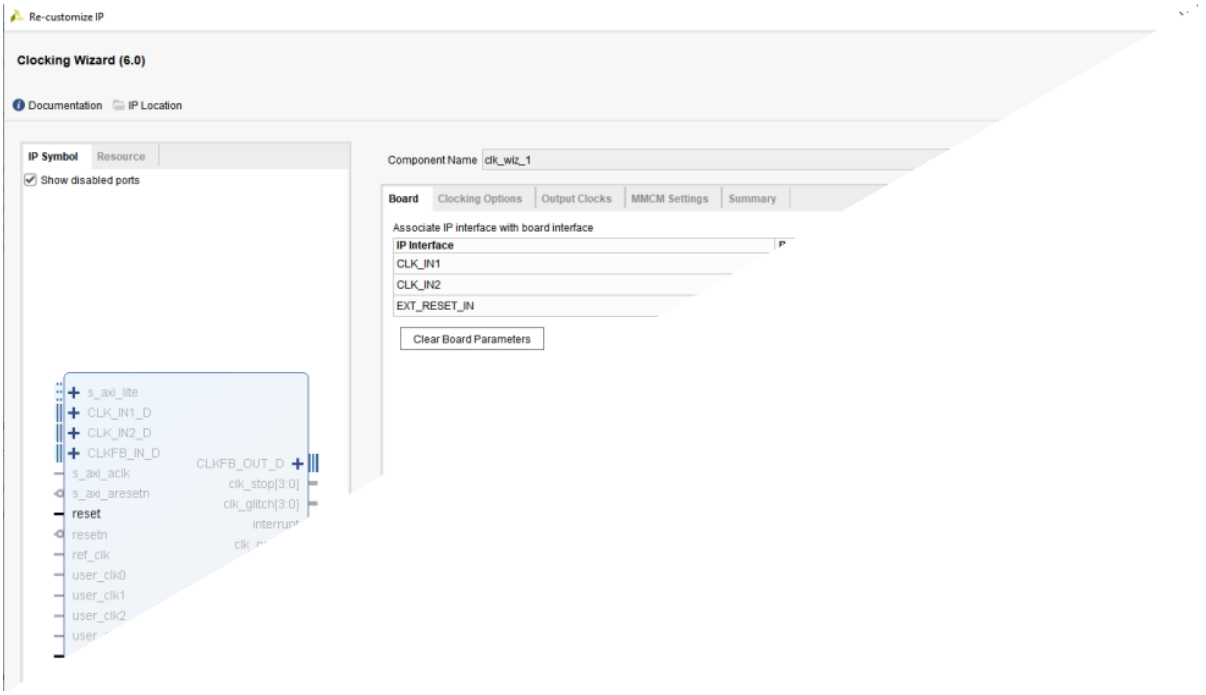
3. Una vez aquí, debería salir la opción de seleccionar nuestra placa en el desplegable Name

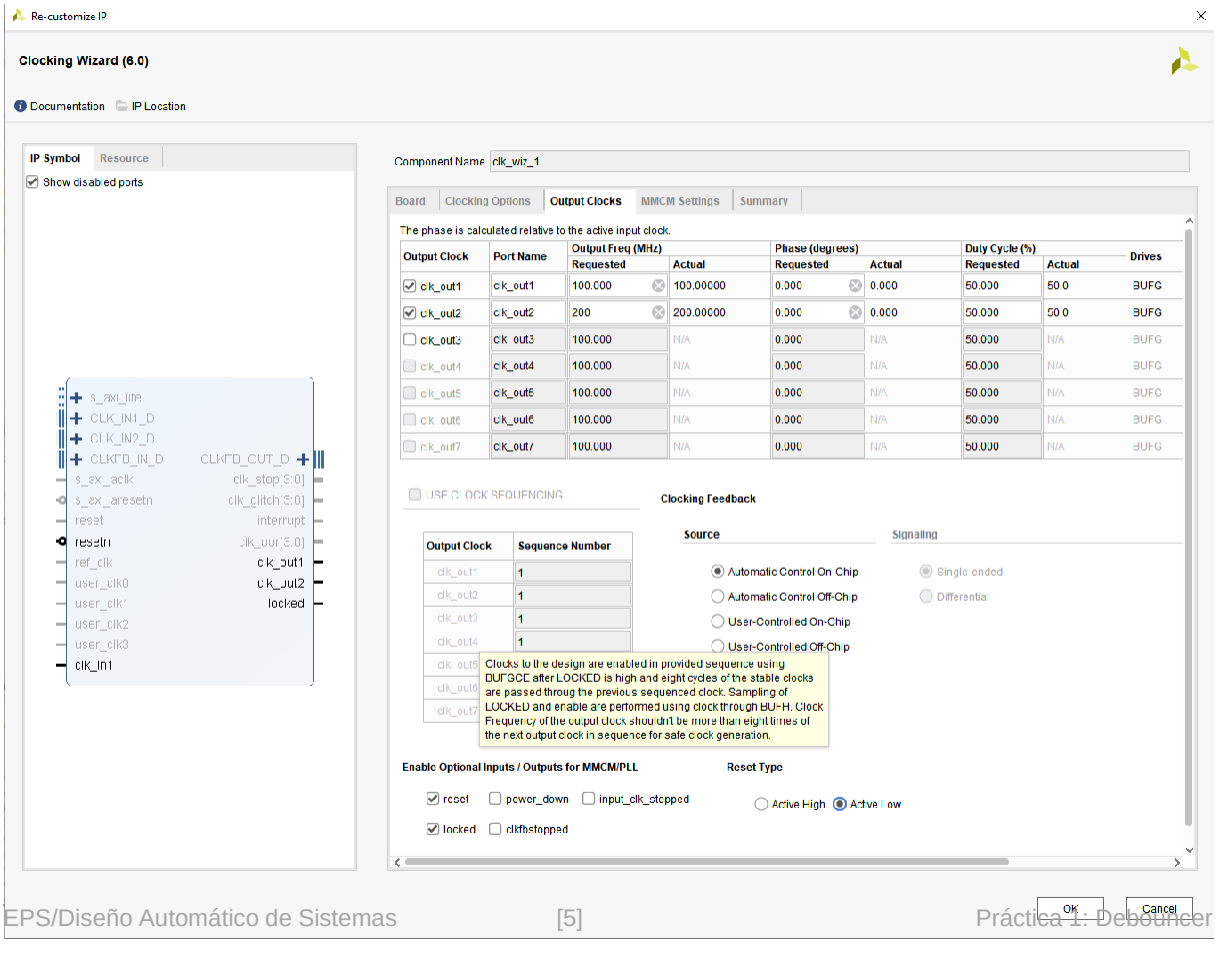


Modificar un bloque: clk_wiz_1



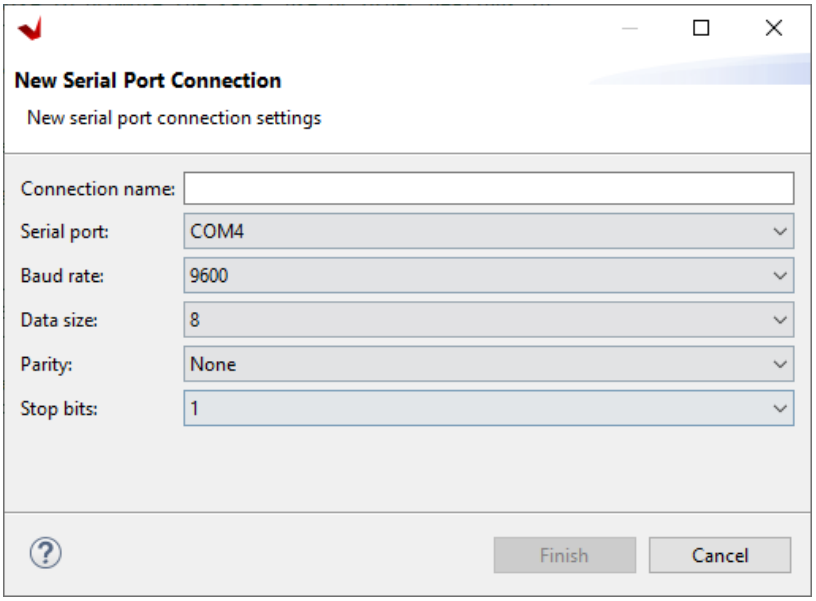
Configuración del bloque que genera las señales de reloj





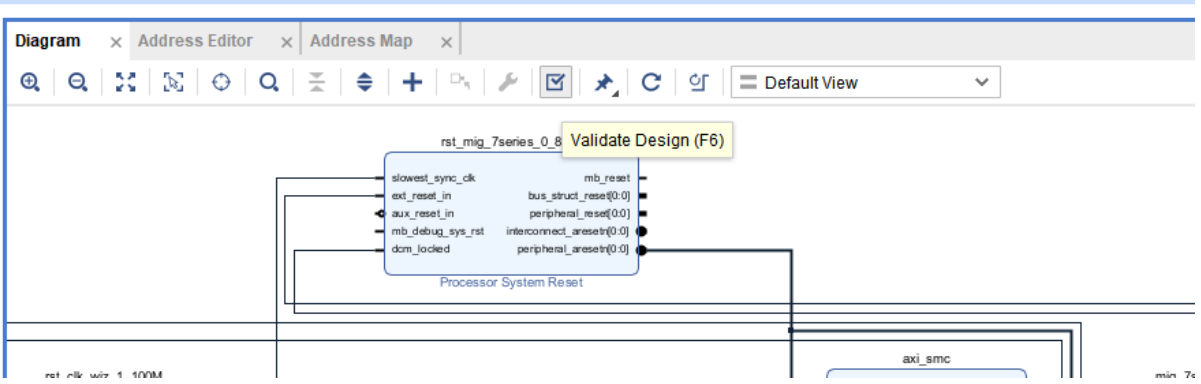
Interfaz UART

La interfaz UART (UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER) es una interfaz de comunicaciones serie, que se ha de configurar con una velocidad de transmisión en baudios, esta vez se ha configurado en 9600, pero también hay opciones como 4800, esto es dependiente de la frecuencia de nuestro sistema y como queramos configurarla, Vitis nos proporciona esta interfaz para poder configurarlo de manera directa.



Validar el diseño HW

Para poder hacer este paso, lo primero que hay que hacer es pasar la validación del diseño, que es un paso que implementa vivado antes de realizar cualquier carga de software en un sistema, para garantizar que este diseño es viable.



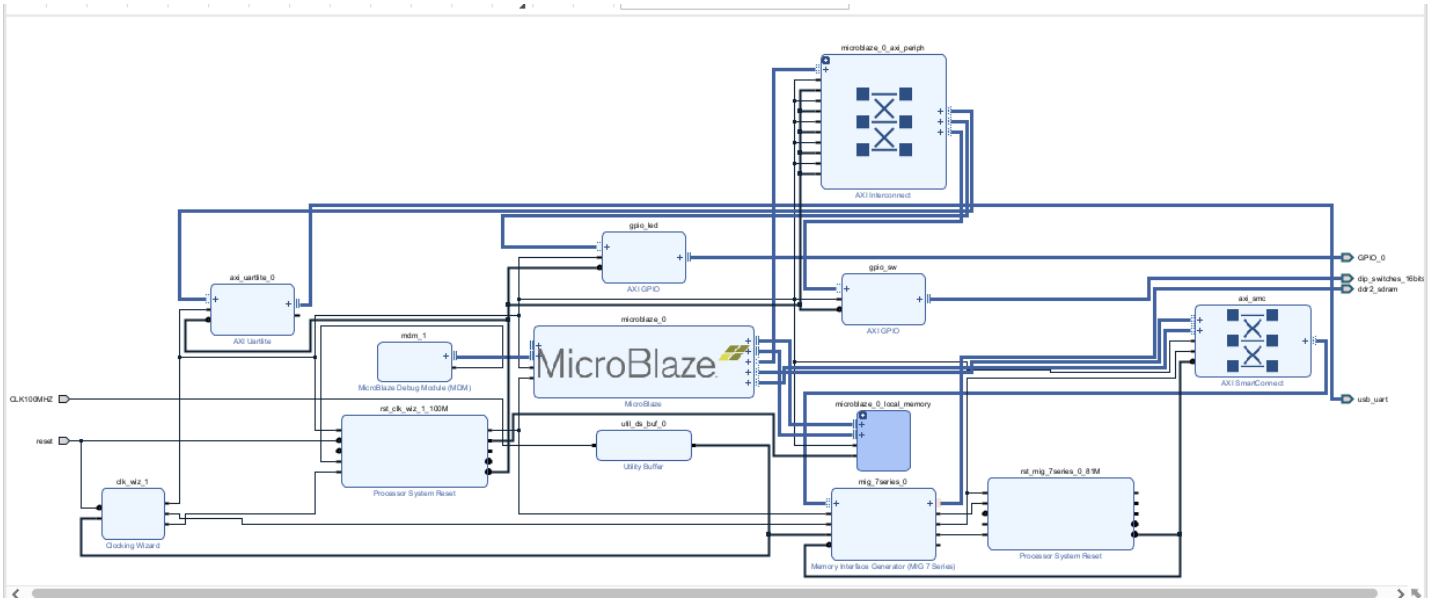
Tras esto, habrá que crear el HDL wrapper que se hace simplemente escogiendo la opción del desplegable que se genera al hacer clock derecho sobre el archivo de design source.

Generación de bitstream y exportar proyecto

Ahora, se va a generar el paso final para poder realizar esta simulación, se va a generar el bitstream, que es la forma que tiene vivado de describir nuestro proyecto para que pueda ser cargado en FPGAS reales y simulaciones. Este paso es el más pesado a nivel computacional ya que se están generando los componentes y conexiones simuladas, que luego podrán ser cargados en la FPGA.

Diseño final

Así quedaría el diseño final una vez se han automatizado las conexiones entre sí después de haber ejecutado la instrucción de run block automation.



Diseño de la aplicación SW

A continuación, se muestra el código que se ha diseñado y se cargará en el sistema a través de Vitis.

```c
#include <stdio.h>
#include "platform.h"
#include "xil_printf.h"
#include "xgpio.h"


XGpio sw_gpio , led_gpio;

void init_driver(){
    int status;
    status = XGpio_Initialize(&sw_gpio,XPAR_GPIO_SW_DEVICE_ID);
    if(status!=XST_SUCCESS){
        print("Failed to initialize switch gpio. \n\r");
    }else{
        print("Initialized switch gpio\n\r");
        XGpio_SetDataDirection(&sw_gpio,1,1);
    }

    status = XGpio_Initialize(&led_gpio,XPAR_GPIO_LED_DEVICE_ID);
    if(status != XST_SUCCESS){
        print("Failed to initialize led gpio. \n\r");
    }else{
        print("Initialized led gpio\n\r");
        XGpio_SetDataDirection(&led_gpio,1,0);
        XGpio_DiscreteWrite(&led_gpio,1,0);
    }
}
```

```c
int main()
{
    int sw_val;


    init_platform();
    init_driver();
    print("Hello World\n\r");
    print("Successfully ran Hello World application");
    while(1){
        sw_val=XGpio_DiscreteRead(&sw_gpio, 1);
        switch(sw_val){
            case 0:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x0);
                break;
            case 1:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x1);
                break;
            case 2:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x2);
                break;
            case 3:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x3);
                break;
            case 4:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x4);
                break;
            case 5:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x5);
                break;
            case 6:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x6);
                break;
            case 7:
                XGpio_DiscreteWrite(&led_gpio, 1, 0x7);
                break;
        }
    }
    cleanup_platform();
    return 0;
}
```

MICROBLAZE

MicroBlaze es un microprocesador soft-core de 32 bits desarrollado por Xilinx. Se utiliza principalmente en FPGAs (Field Programmable Gate Arrays) de Xilinx, permitiendo a los diseñadores de sistemas integrar una CPU personalizable en sus diseños de FPGA sin la necesidad de un procesador dedicado.

Algunas de sus características son:

- **Configurable:** MicroBlaze es altamente configurable, permitiendo a los diseñadores seleccionar solo las características y periféricos necesarios, lo que puede ayudar a minimizar el uso de recursos y energía.
- **Compatibilidad con herramientas**: Se integra bien con el ecosistema de herramientas de Xilinx, incluyendo Vivado y Vitis, lo que facilita el desarrollo, la depuración y la implementación.
- **Soporte para varios sistemas operativos**: MicroBlaze soporta una variedad de sistemas operativos en tiempo real, incluyendo FreeRTOS y Linux, lo cual es ideal para aplicaciones embebidas
- **Arquitectura**: Ofrece opciones para configuraciones con o sin gestión de memoria, pipelining, instrucciones de punto flotante, y más.

Aparte entre sus aplicaciones y usos destacan 2:

- **Sistemas Embebidos:** Es ideal para aplicaciones embebidas donde el espacio, el costo y el consumo de energía son críticos.
- **Prototipado Rápido:** Permite a los desarrolladores prototipar rápidamente ideas y conceptos sin la necesidad de hardware dedicado.

HARD-CORE VS SOFT-CORE

Soft-core Processor: Un procesador soft-core, como MicroBlaze, es un diseño de microprocesador implementado en la lógica programable de un FPGA. No está físicamente presente en el silicio del hardware; en su lugar, se sintetiza dentro del FPGA a partir de una descripción en HDL (Hardware Description Language).

Hard-core Processor: Un procesador hard-core está físicamente implementado en el silicio de un chip. Ejemplos de esto incluyen los procesadores ARM Cortex que se encuentran en los SoCs (System on Chips) de Xilinx, como la serie Zynq.

Depende de las necesidades de cada caso hay ventajas de uso de cada uno:

Ventajas de Soft-core sobre Hard-core:

- **Flexibilidad**: Los soft-cores pueden ser personalizados para ajustarse exactamente a las necesidades del proyecto, incluyendo solo las características necesarias.
- **Escalabilidad**: Se puede escalar el diseño cambiando la configuración del soft-core sin necesidad de cambiar el hardware físico.
- **Costo**: En ciertos casos, el uso de soft-cores puede reducir los costos de desarrollo y producción al utilizar FPGAs genéricos en lugar de SoCs específicos.

Ventajas de Hard-core sobre Soft-core:

- **Rendimiento**: Generalmente, los hard-cores ofrecen mejor rendimiento y mayor eficiencia energética que los soft-cores, ya que están optimizados durante el proceso de fabricación del silicio.
- **Integración:** Los hard-cores permiten una mayor integración de componentes, lo que puede simplificar el diseño del sistema y reducir la complejidad general.
- **Menor Latencia**: Los procesadores hard-core pueden ofrecer menores latencias en comunicación entre diferentes bloques del SoC, lo cual es crucial para ciertas aplicaciones en tiempo real.