Trabajo práctico final
Microarquitecturas y softcores
Carrera de Especialización en Sistemas Embebidos

Contador BCD comandado por instrucciones desde microcontrolador

Alumno: Raúl Emilio Romero

Introducción

En este trabajo se implementa un bloque de hardware digital que consiste en un contador BCD comandado por un microcontrolador integrado en una FPGA. Un contador BCD sigue una secuencia desde 0 incrementando la cuenta de a 1. Al llegar al 9 pasa del 9 al 0 y continúa contando. Para este tipo de contador en sistema binario se requieren 4 bits. El comando del contador se realiza por medio un pequeño set de instrucciones que definen el comportamiento del contador.

Requerimientos

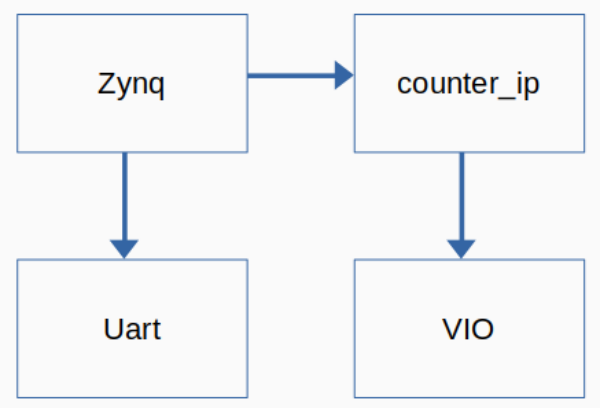
El contador que aquí se describe se diseñó e implementó siguiendo los lineamientos definidos en el documento del trabajo final y considerando los siguientes requerimientos. Estos, se definieron teniendo en cuenta los alcances del curso y el tiempo para la realización de este trabajo.

El diseño deberá:

- El contador BCD se comandará por medio de el envío de comandos desde un microcontrolador integrado en la FPGA.
- El software del microcontrolador deberá permitir realizar una secuencia de comandos automática que permita observar y analizar el comportamiento completo del contador.
- El contador contará de 0-9, es decir será un contador del tipo BCD.
- Al enviar el carácter “r” (reset) el contador deberá reiniciar su cuenta.
- Al enviar el carácter “a” (ascendente) el contador deberá sumar el valor “1” a la cuenta desde la posición actual en que se encuentre el contador.
- Al enviar el carácter “d” (descendente) el contador deberá restar el valor “1” a la cuenta desde la posición actual en que se encuentre el contador.
- Al enviar el carácter “p” (pausa) el contador pausará su cuenta dejando el valor actual en la pantalla.
- Al enviar cualquier otro carácter (diferente a los anteriores) el contador reiniciará su cuenta.
- Luego de un carácter “p” si se presiona “a” o “d” el contador deberá seguir su cuenta normalmente.
- Iniciada la cuenta se realizará indefinidamente.
- La frecuencia de contador deberá ser de 1 Hz para que pueda visualizarse en pantalla.
- El valor actual de la cuenta se mostrará en un visualizador VIO del hardware.

Diagrama en bloques

El siguiente diagrama muestra los bloques y subsistemas generales del dispositivo implementado.

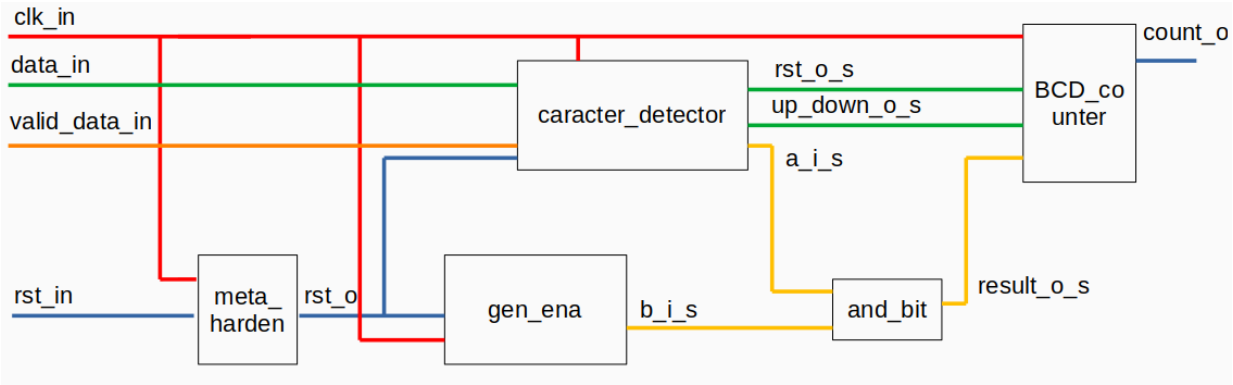


El sistema está formado por cuatro módulos principales. El módulo counter_ip es el contador BCD propiamente dicho integrado en un paquete IP. Este es el desarrollo de hardware principal. Counter_ip es comandado por medio de comandos cargados en registros desde un microcontrolador de Zynq integrado en la FPGA. El desarrollo del software del proyecto se realiza principalmente sobre el microcontrolador. El módulo uart se utiliza para mostrar información por terminal desde el microcontrolador y el módulo VIO se utiliza para visualizar la salida del contador, es decir, para observar el comportamiento del sistema.

El sistema se comanda de forma remota a través de un terminal remoto conectándose por SSH. Dentro del servidor remoto, donde está alojada la placa de desarrollo que contiene la FPGA y la interfaz de comunicación serie, la comunicación con el microcontrolador de la familia Zynq se realiza por medio de minicom. El desarrollo de hardware implementado se programa y se visualiza desde la herramienta de desarrollo Vivado 2018.3. Por otro lado, el desarrollo de software se realiza con la herramienta SDK 2018.3 . Con la misma herramienta se carga el software desarrollado sobre el microcontrolador integrado en la FPGA.

Diagrama de componentes de counter_ip

El desarrollo que aquí se describe surge del proyecto desarrollado en el trabajo final de la materia Circuitos Lógicos Programables. Los cambios que se realizaron se orientan principalmente a reemplazar el módulo Uart para el envío de los comandos por un microcontrolador que realice la misma función y el empaquetado de toda la lógica programable referida al contador BCD en en una IP de usuario.



Counter_ip cuenta con los siguientes componentes principales:

- and_bit_component: componente and que sirve para sincronizar la frecuencia en la que el contador actualiza su valor y el comando recibido.
- bcd_up_down_counter: contador BCD con posibilidad de cuenta ascendente y descendente.
- caracter_detector: componente que se utiliza para detectar los caracteres a,d,p y r.

- enable_generator: componente para generar a la frecuencia de 1 Hz la habilitación del componente bcd_up_down_counter
- meta_harden: componente que se utiliza para evitar inestabilidades o metas estados en el pin de entrada asicrónica reset.
- counter: componente principal del sistema.

El funcionamiento del sistema es el siguiente:

- Se envía por medio de un registro un comando por ejemplo el carácter “a”.
- La instancia meta_harden reduce la meta-estabilidad en la entrada de rst. De esta manera la entrada de reset de todos los módulos es la señal rst_o. La entrada de reset del módulo BCD_counter (el nombre real se describe arriba, aquí se pone un nombre reducido para simplificar la explicación) viene del módulo caracter_detector y no es la señal rst_o.
- El módulo caracter_detector se encarga de identificar si el dato ingresado corresponde a alguno de los comandos esperados (a, d, p y r). Si es el caso, el módulo genera tres señales de 1 bit que entran componente contador.

caracter_detector (out)	**Reset (r)**	**Ascendente (a)**	**Descendente (d)**	**Pausa (p)**	**Otros caracteres**
reset	1	0	0	0	0
up_down	0	1	0	0	0
detect	0	1	1	0	0

- El módulo BCD_counter se configura con las salidas del módulo caracter_detector, sin embargo como se pretende que la frecuencia de operación del módulo sea de 1 Hz (para que pueda visualizarse adecuadamente) la habilitación se realiza de forma sincronizada por medio del módulo and_bit y el generador de habilitación (gen_ena) a 1 Hz.

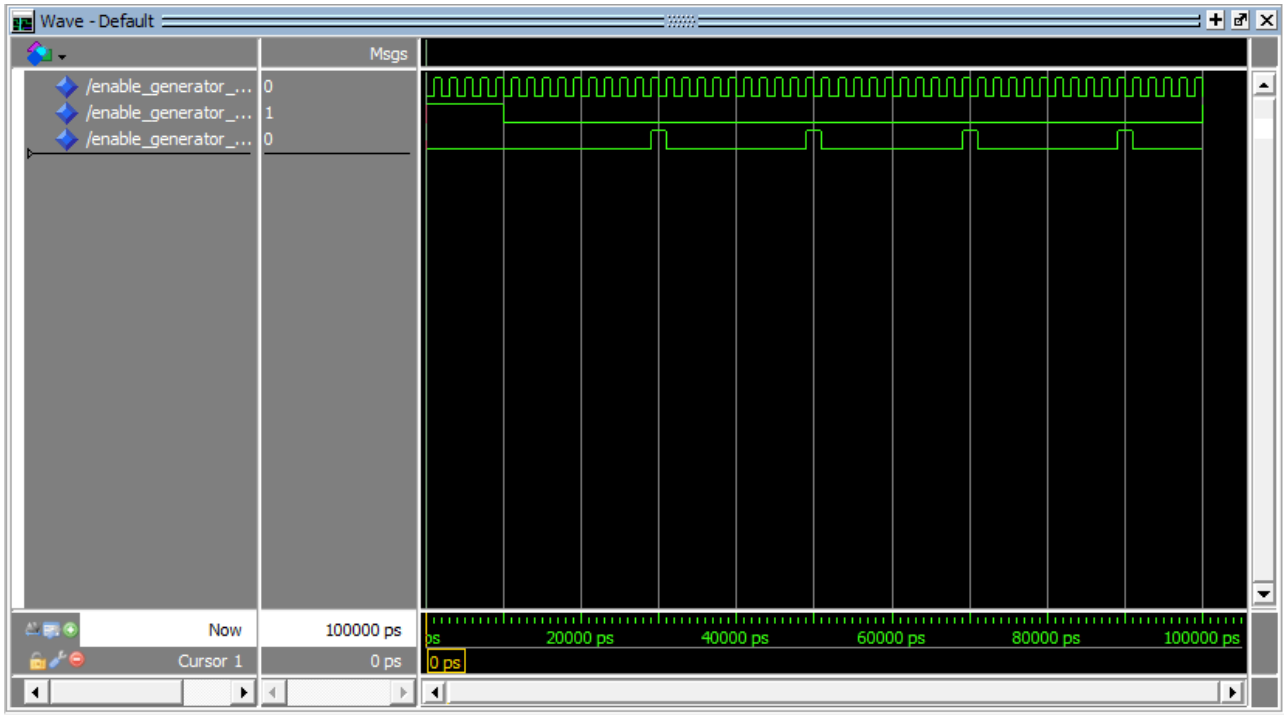
Simulaciones del módulo counter_ip

Se muestran las simulaciones mas relevantes para este trabajo y que no necesariamente se incluyeron en el diseño final:

- Módulo enable_generador
- Módulo caracter_detector
- Módulo bcd_up_down_counter

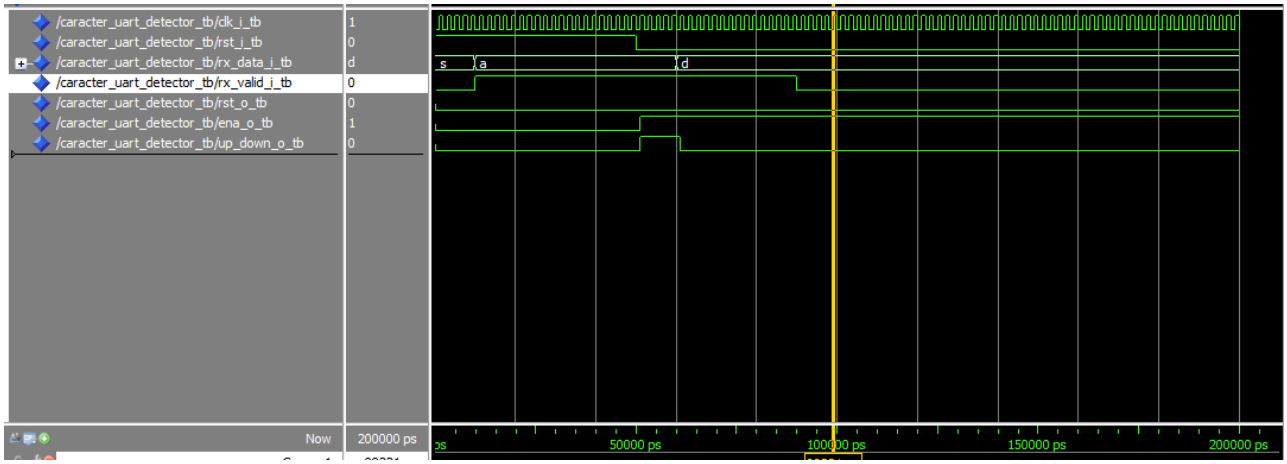
enable_generador

En este caso se puede observar como el periodo de la señal de salida del módulo es 10 veces mas grande que el periodo de la señal de reloj. Además lo que diferencia a un generador de habilitación de un divisor de frecuencias es que el ciclo de actividad del generador de habilitación es de ancho igual a un ciclo de la señal de clock.



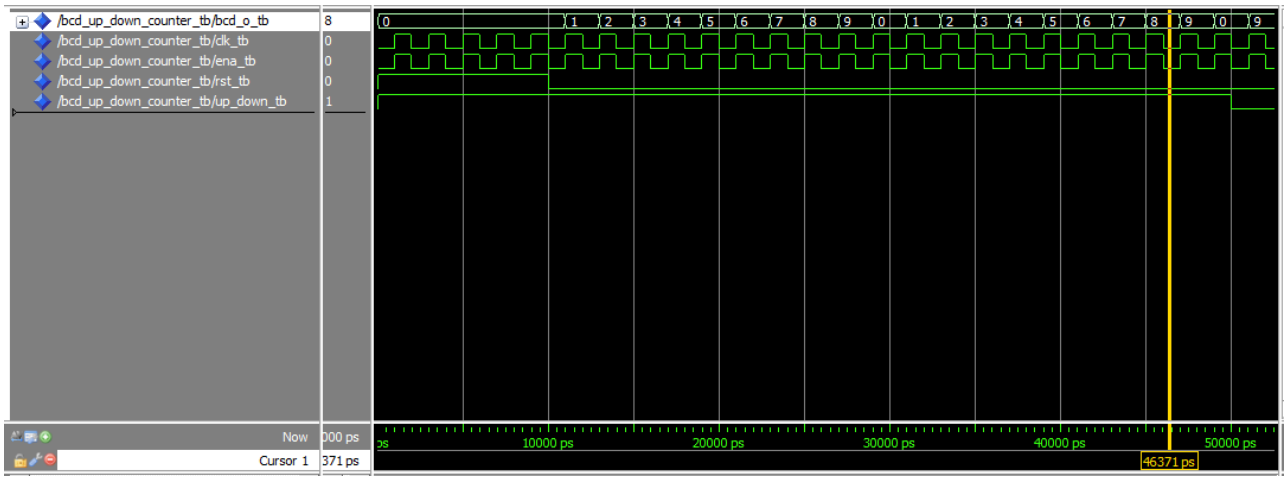
caracter_detector

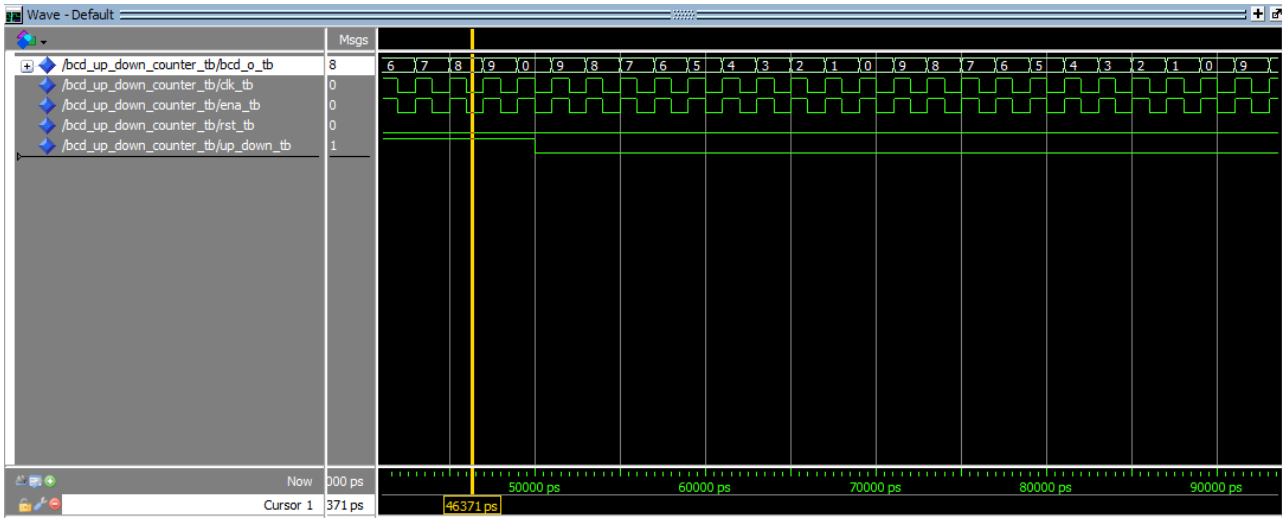
En la siguiente imagen se observa que luego de la habilitación del módulo por la señal de reset se activa la cuenta ascendente por 4 ciclos y luego como la señal up_down cambia las señales de salida del módulo. Queda así, configurado para que el contador quede en cuenta descendente permanentemente.



bcd_up_down_counter

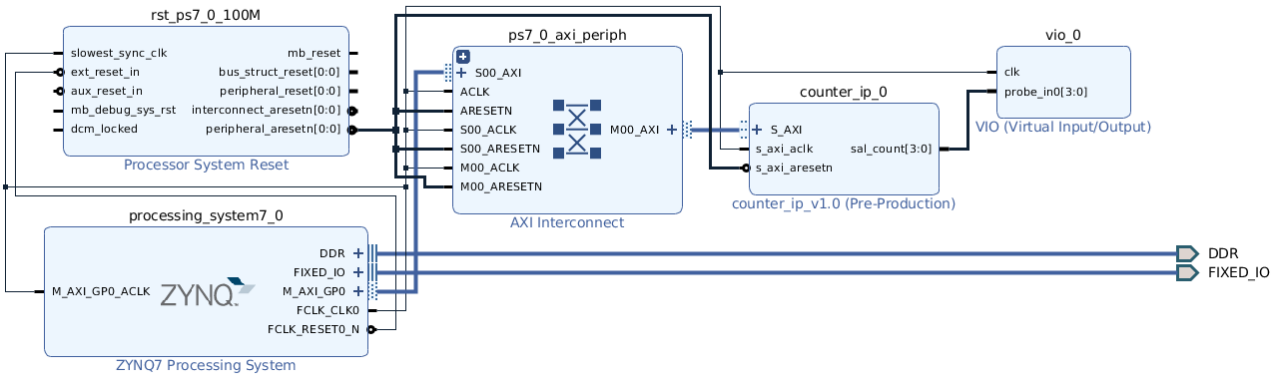
En las imágenes siguientes se muestran como la señal de cuenta se mantiene en cero mientras el reset vale 1 y la señal de up_down esta en modo ascendente (estado 1). Luego en la siguiente imagen se muestra el cambio a cuenta descendente. Se debe aclarar que para la simulación la señal de reloj y la señal de habilitación tienen la misma frecuencia.





Block design

En la figura se muestra el diagrama del sistema que integra el microcontrolador Zunq7, el counter_ip y la salida vio.



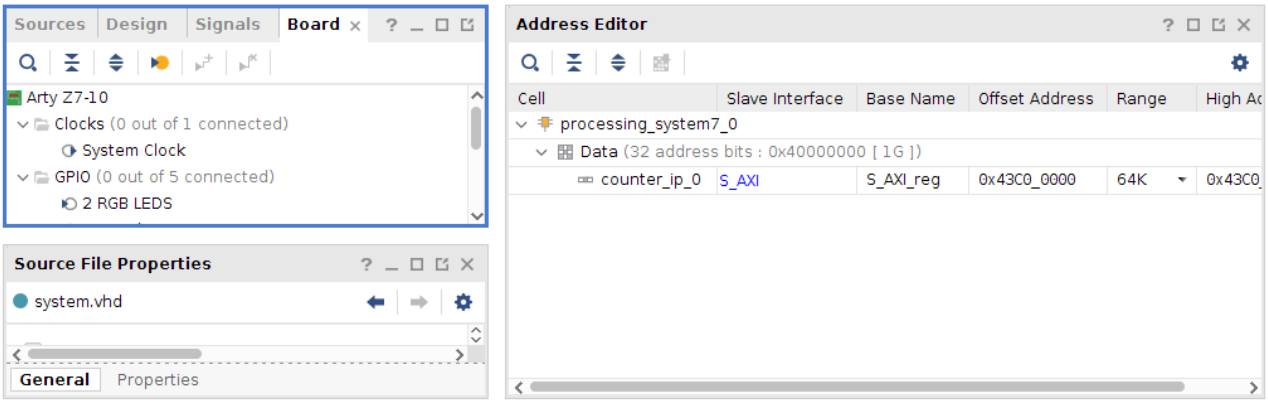
Instancia de counter_ip

En la figura se muestra la declaración de counter_ip donde se muestran las señales y registros involucrados.

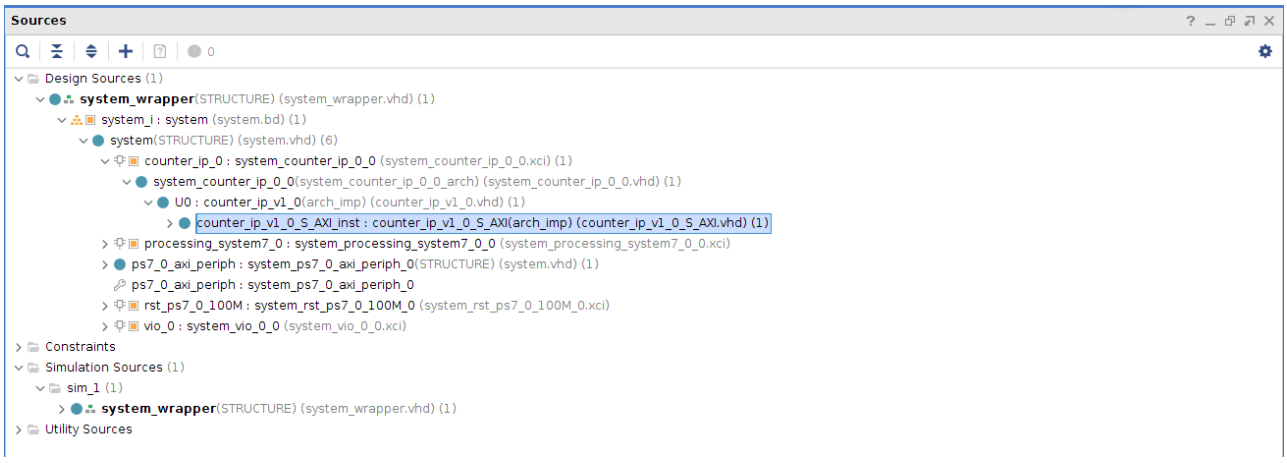
```
397     -- Add user logic here
398     count_inst: counter
399     port map(
400         clk_in        => S_AXI_ACLK,
401         rst_in        => slv_reg0(0),
402         valid_data_in => slv_reg1(0),
403         data_in       => slv_reg2(7 downto 0),
404         count_o       => sal_count
405         );
406     -- User logic ends
407
408 end arch_imp;
409
```

Vista del board y Address Editor

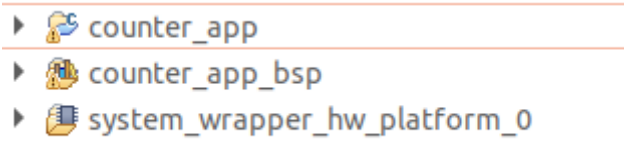
En la figura de muestra la configuración y definición del micro utilizado Arty Z7-10 (con sus periféricos) y la dirección asignada para la instancia counter_ip_0 (0x43C00000).



Vista de jerárquica de módulos y archivos

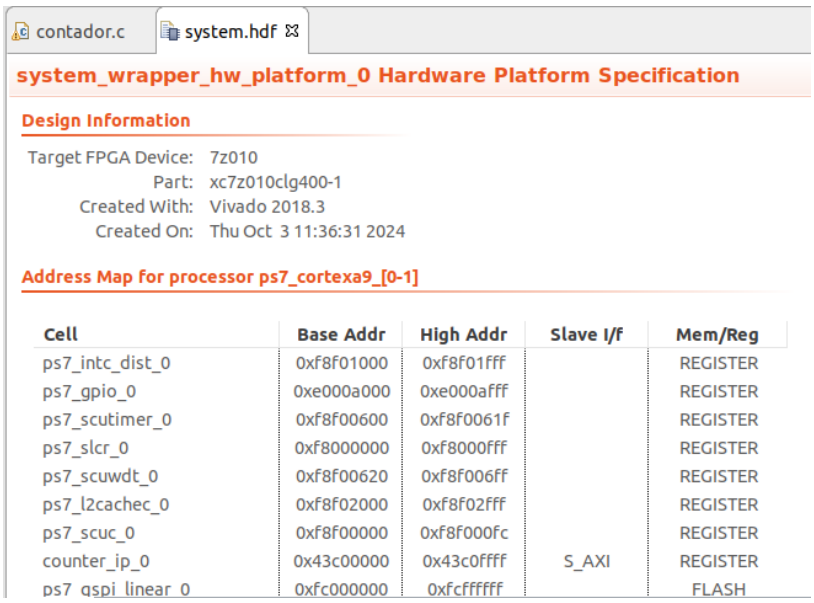


Estructura de archivos en el proyecto de software (en SDK)



Archivo system.hdf

En la figura de la dirección base de counter_ip_0. Esto es importante para poder utilizar las funciones de escritura o lectura de los registros.



contador.c | system.hdf

system_wrapper_hw_platform_0 Hardware Platform Specification

Design Information

Target FPGA Device: 7z010
Part: xc7z010clg400-1
Created With: Vivado 2018.3
Created On: Thu Oct 3 11:36:31 2024

Address Map for processor ps7_cortexa9_[0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
counter_ip_0	0x43c00000	0x43c0ffff	S_AXI	REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcffffff		FLASH

Archivo counter.c.

contador.c

```c
#include "xparameters.h"
#include "xil_io.h"
#include "counter_ip.h"
// Comandos para trabajar con counter_ip "r", "a", "d", "p"

#define RESET_REG          0  // reset por registro
#define AVAILABLE_DATA     1  // Dato disponible para ser leido
#define NOT_AVAILABLE_DATA 0  // Dato no disponible

#define RESET_COMAND       114 // reset de la cuenta por comando
#define PAUSE              112 // pausa la cuenta
#define ASC_COUNT          97  // habilitación y cuenta ascendente
#define DES_COUNT          100 // habilitación y cuenta descendente
#define INVALID_COMAND     120 // comando inválido
```

```c
int main (void)
{
   int i = 0; // variable auxiliar para recorrer la secuencia de prueba
   int data[5]={ASC_COUNT,DES_COUNT,PAUSE,ASC_COUNT,RESET_COMAND}; // secuencia de prueba
   char ComandDataName[5][14]={"ASC_COUNT","DES_COUNT","PAUSE","ASC_COUNT","RESET_COMAND"};// secuencia de prueba. Etiquetas

   xil_printf("-------------------------\r\n");
   xil_printf("-- START PROGRAM COUNTER --\r\n");
   xil_printf("-- ALUMNO: RAUL ROMERO   --\r\n");
   xil_printf("-------------------------\r\n");

   // INICIALIZACIÓN
   COUNTER_IP_mWriteReg(XPAR_COUNTER_IP_0_S_AXI_BASEADDR,COUNTER_IP_S_AXI_SLV_REG0_OFFSET, RESET_REG); // reset count
   COUNTER_IP_mWriteReg(XPAR_COUNTER_IP_0_S_AXI_BASEADDR,COUNTER_IP_S_AXI_SLV_REG1_OFFSET, NOT_AVAILABLE_DATA);// dato no disponible
   COUNTER_IP_mWriteReg(XPAR_COUNTER_IP_0_S_AXI_BASEADDR,COUNTER_IP_S_AXI_SLV_REG2_OFFSET, INVALID_COMAND); // dato inválido
   xil_printf("-- INICIALIZACIÓN --> RESET_REG --%d\r\n",RESET_REG);
   xil_printf("-- INICIALIZACIÓN --> NOT_AVAILABLE_DATA --%d\r\n",NOT_AVAILABLE_DATA);
   xil_printf("-- INICIALIZACIÓN --> INVALID_COMAND --%d\r\n",INVALID_COMAND);


 while (1)
 {
     if(i>4){
         i=0;
     }
     xil_printf("-------------------------\r\n");
     // Dato válido disponible
     COUNTER_IP_mWriteReg(XPAR_COUNTER_IP_0_S_AXI_BASEADDR,COUNTER_IP_S_AXI_SLV_REG2_OFFSET, data[i]);
     xil_printf("-- Comando enviado-dato: %s --> %d\r\n", ComandDataName[i],data[i]);
     // habilitación de dato disponible
     COUNTER_IP_mWriteReg(XPAR_COUNTER_IP_0_S_AXI_BASEADDR,COUNTER_IP_S_AXI_SLV_REG1_OFFSET, AVAILABLE_DATA);
     xil_printf("-- Dato habilitado --\r\n");
     // disponibilidad del dato por 1 segundos
     sleep(1);
     // dato de entrada no disponible
     COUNTER_IP_mWriteReg(XPAR_COUNTER_IP_0_S_AXI_BASEADDR,COUNTER_IP_S_AXI_SLV_REG1_OFFSET, NOT_AVAILABLE_DATA);
     xil_printf("-- Dato deshabilitado   --\r\n");
     // delay para visualizar el comportamiento del contador
     xil_printf("-- sleep --\r\n");
     sleep(10);
     i++;
 }
```

Archivo counter_ip.h

contador.c | system.hdf | counter_ip.h

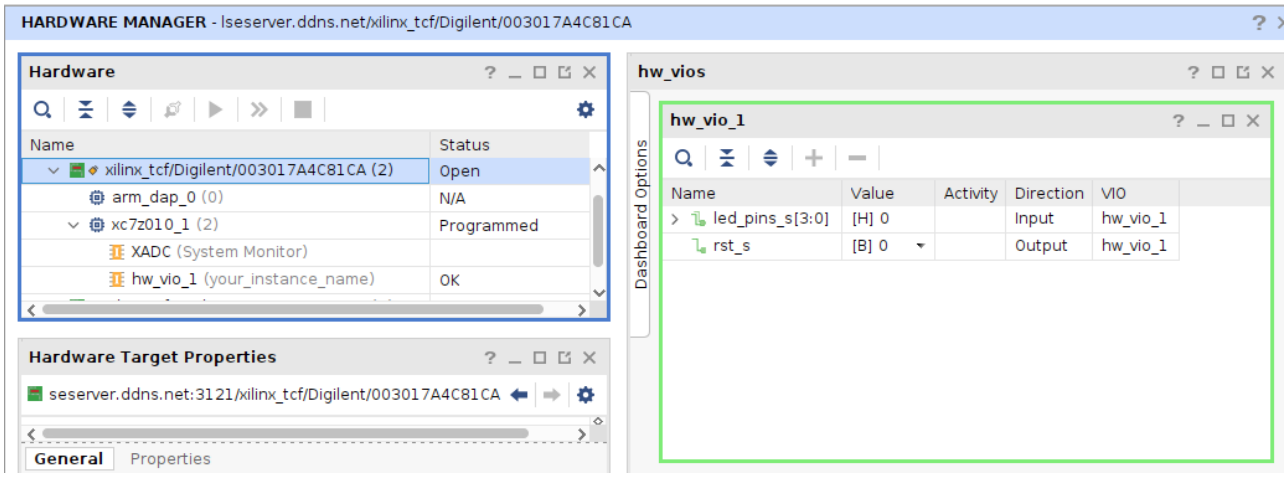
```c
#ifndef COUNTER_IP_H
#define COUNTER_IP_H


/****************** Include Files ********************/
#include "xil_types.h"
#include "xstatus.h"

#define COUNTER_IP_S_AXI_SLV_REG0_OFFSET 0
#define COUNTER_IP_S_AXI_SLV_REG1_OFFSET 4
#define COUNTER_IP_S_AXI_SLV_REG2_OFFSET 8
#define COUNTER_IP_S_AXI_SLV_REG3_OFFSET 12


/**************************** Type Definitions ***********
```

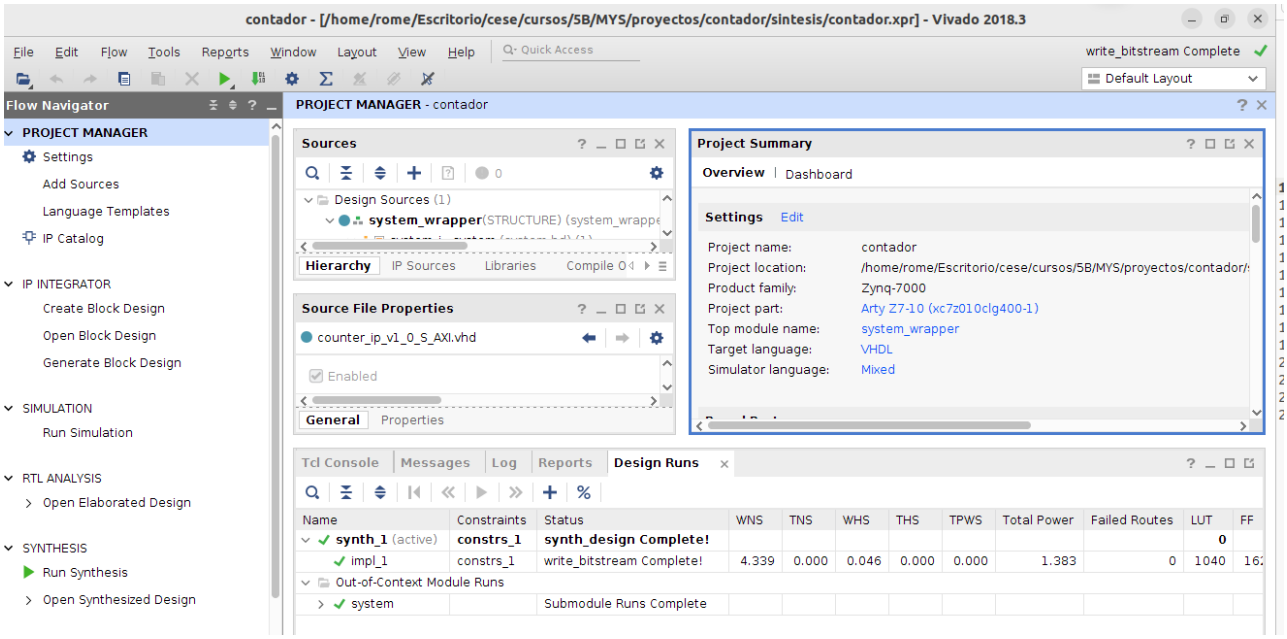
Archivo

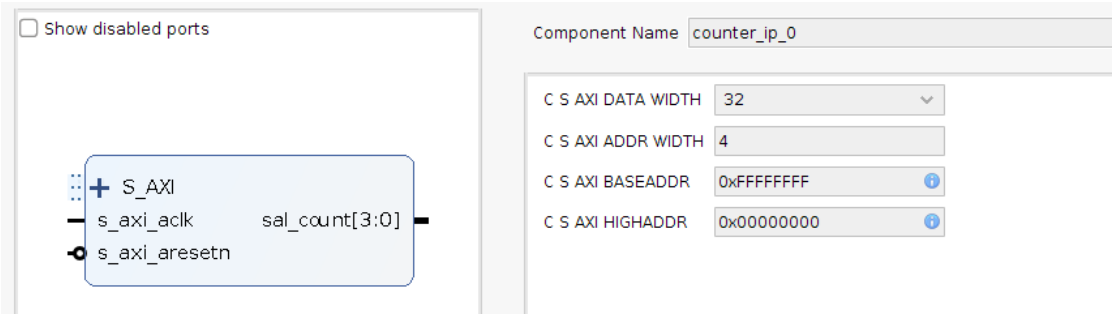
xparameters.h

```
/*****************************************************************/
/* Definitions for driver COUNTER_IP */
#define XPAR_COUNTER_IP_NUM_INSTANCES 1

/* Definitions for peripheral COUNTER_IP_0 */
#define XPAR_COUNTER_IP_0_DEVICE_ID 0
#define XPAR_COUNTER_IP_0_S_AXI_BASEADDR 0x43C00000
#define XPAR_COUNTER_IP_0_S_AXI_HIGHADDR 0x43C0FFFF


/*****************************************************************/
```

Capturas generales de la implementación






Show disabled ports
S_AXI
s_axi_aclk
s_axi_aresetn
sal_count[3:0]
Component Name
counter_ip_0
C S AXI DATA WIDTH
32
C S AXI ADDR WIDTH
4
C S AXI BASEADDR
0xFFFFFFFF
C S AXI HIGHADDR
0x00000000

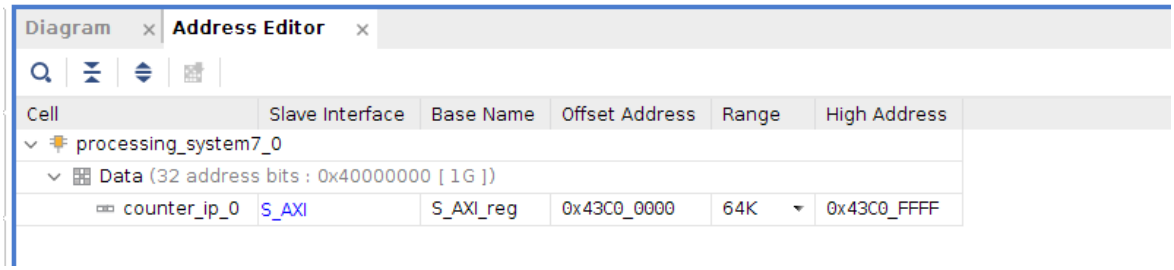
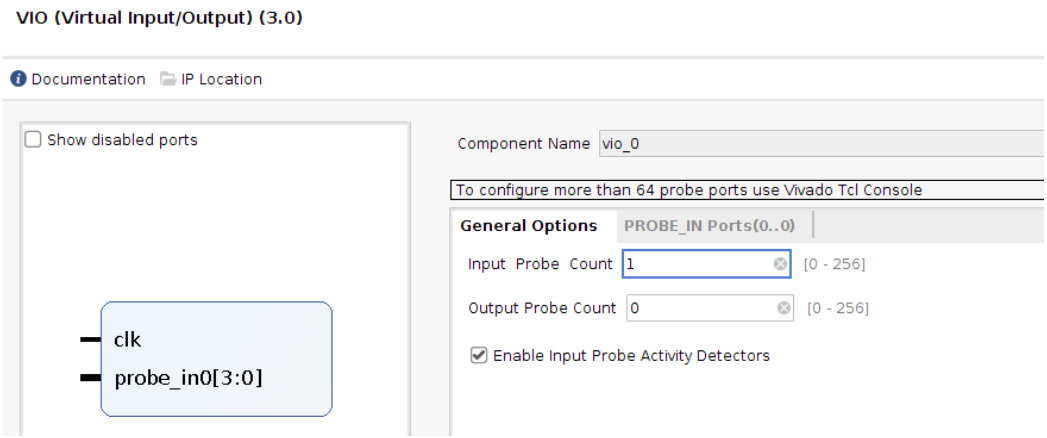
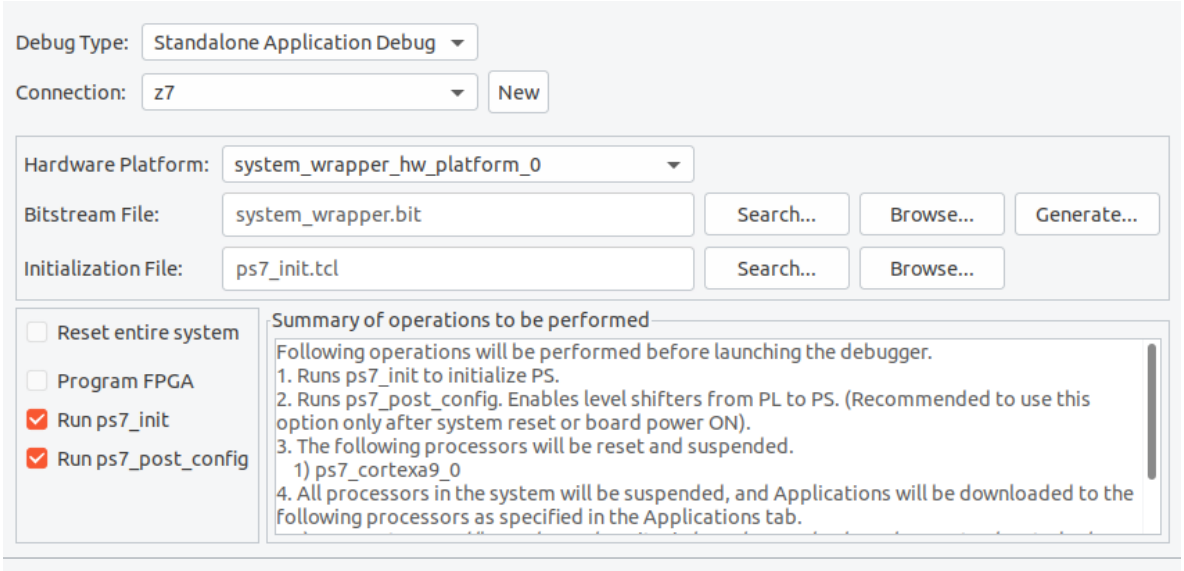


Diagram
Address Editor
Cell
Slave Interface
Base Name
Offset Address
Range
High Address
processing_system7_0
Data (32 address bits : 0x40000000 [1G])
counter_ip_0
S_AXI
S_AXI_reg
0x43C0_0000
64K
0x43C0_FFFF


VIO (Virtual Input/Output) (3.0)
Documentation
IP Location
Show disabled ports
clk
probe_in0[3:0]
Component Name
vio_0
To configure more than 64 probe ports use Vivado Tcl Console
General Options
PROBE_IN Ports(0..0)
Input Probe Count
1
[0 - 256]
Output Probe Count
0
[0 - 256]
Enable Input Probe Activity Detectors


Debug Type:
Standalone Application Debug
Connection:
z7
New
Hardware Platform:
system_wrapper_hw_platform_0
Bitstream File:
system_wrapper.bit
Search...
Browse...
Generate...
Initialization File:
ps7_init.tcl
Search...
Browse...
Reset entire system
Program FPGA
Run ps7_init
Run ps7_post_config
Summary of operations to be performed
Following operations will be performed before launching the debugger.
1. Runs ps7_init to initialize PS.
2. Runs ps7_post_config. Enables level shifters from PL to PS. (Recommended to use this option only after system reset or board power ON).
3. The following processors will be reset and suspended.
1) ps7_cortexa9_0
4. All processors in the system will be suspended, and Applications will be downloaded to the following processors as specified in the Applications tab.