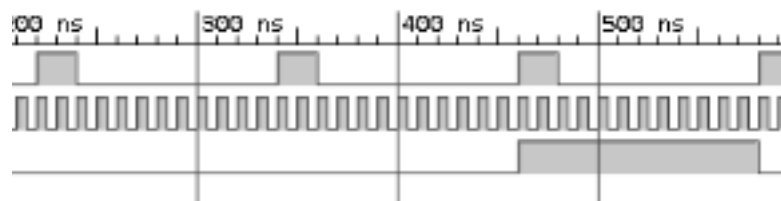


# Trabajo Final

**Microarquitecturas y Softcores**

## Detector de períodos de un tren de pulsos con FPGA ( IP CORE & CORTEX A9)



**Autor: Ing. Fabián Ezequiel Urdaniz**

**Docente: Ing. Nicolás Álvarez**

**Fecha: Agosto 2020**

## Índice:

<b>Descripción de trabajo</b>	<b>3</b>
<b>Objetivos</b>	<b>3</b>
<b>Nuevos elementos de trabajo</b>	<b>3</b>
<b>Actualizando el nuevo código del detector</b>	<b>4</b>
<b>IP DETECTOR</b>	<b>5</b>
Detector ip v1	5
Detector ip v1 S AXI	6
Detector IP - Propiedades	7
<b>ZYNQ A9 sistema procesador</b>	<b>8</b>
Bloque de diseño Zynq	8
ZYNQ - Propiedades	8
<b>Sistema wrapper (integrado)</b>	<b>9</b>
<b>Aplicación Core detector App</b>	<b>9</b>
Verificación del hardware exportado	9
Xilinx SDK	10
Código - Aplicación core detector header	10
Código - Aplicación detector fuente	11
<b>BITSTREAM</b>	<b>12</b>
Fallas en generación de Bitstream	12
Solución en generación de Bitstream	12
<b>Conexión remota FPGA</b>	<b>13</b>
Error en programación remota de FPGA remota	13
Desconexión del puerto ttyUSB	14
<b>Conclusión</b>	<b>15</b>

## Descripción de trabajo

Este mini proyecto busca implementar con IP CORE y CORTEX A9 la detección de un tren de pulsos emitidos por un radar y lograr determinar su periodo de manera sincrónica.

El trabajo es la continuación del proyecto final de la materia Circuitos lógicos programables y cómo se habíamos expresado no tiene relación directa con el proyecto de finalización de carrera pero se desprende de un proyecto mayor que tiene como objetivo el control de señales de un radar con FPGA

## Objetivos

Desarrollar un bloque de hardware digital en lenguaje VHDL (IP Core) que integre un sistema base de procesamiento junto con el micro Cortex A9. El funcionamiento del mismo será implementado a través de un código C. El sistema debe ser capaz de detectar el período de una señal del tipo tren de pulsos y emitir una señal de detección cuya duración es el período calculado. En caso de fallas o errores emitir una señal de falla. Estos eventos deben ser informados en los correspondientes registros.

## Nuevos elementos de trabajo

Se propone implementar un detector con las siguientes entradas, salidas y registros:

### Entradas:

- Clock (reloj interno).
- Señal entrada (secuencia).

### Salidas:

- Señal de detección.
- Señal de falla.

### Lenguaje:

- VHDL.
- C.

### Nuevos registros:

- Detect : Logic\_vector (31 downto 0).
- Error : Logic\_vector (31 downto 0).

Con las capacidades de detectar el tren de pulsos, determinar su periodo y emitir una señal de detección de igual duración que el periodo detectado. Además en caso de fallas o errores se emite una señal de falla. Estos eventos son informados en los correspondientes registros.

El sistema anterior estaba compuesto principalmente por un componente detector de señal y su implementación se basó en::

- Máquinas de estados.
- Conversiones.
- Contadores.
- Combinacionales.

El nuevo sistema propuesto está compuesto principalmente por un IP CORE detector y un sistema de procesamiento ZYNQ7

- Registros 32bits
- IP CORE DETECTOR
- ZYNQ - Cortex-A9
- Vivado - SDK

## Actualizando el nuevo código del detector

Adaptamos el código viejo de detector.vhd a los nuevos objetivos del trabajo. Agregamos nuevos registros que tienen como función principal almacenar la detección de periodos y fallas. En la imagen podemos observar los registros.

```

4  ----- entity detector FSM -----
5  library IEEE;
6  use IEEE.std_logic_1164.all;
7  use IEEE.numeric_std.all;
8
9  entity detector is
10     port(
11         ----- i/o -----
12         sig_secuencia_i : in std_logic;           -- signal in.
13         clk_i           : in std_logic;           -- clock.
14         sig_detec_o      : out std_logic;          -- detec signal out.
15         fail_o           : out std_logic;          -- fail.
16         ----- new registros -----
17         detc_o           : out std_logic_vector(31 downto 0);
18         err_o            : out std_logic_vector(31 downto 0);
19     );
20 end detector;

```

La actualización de los registros se realiza dentro del proceso de transiciones en los cambios de la máquina de estado.

```

----- new registros detec -----
if (estado_actual = D) then
    detc_o <= std_logic_vector(to_unsigned(1,32));
else
    detc_o <= std_logic_vector(to_unsigned(0,32));
end if;

----- new registros err_o -----
if (estado_actual = E) then
    err_o <= std_logic_vector(to_unsigned(1,32));
else
    err_o <= std_logic_vector(to_unsigned(0,32));
end if;

```

## IP DETECTOR

Para crear el bloque de hardware detector ip vamos a utilizar como base las plantillas de core que ofrece el vivado. En cada paso se fue modificando y adaptando a las necesidades del proyecto. Los archivos principales son detector\_ip\_v1\_0.vhd y detector\_ip\_v1\_0\_S\_AXI.vhd

### Detector ip v1

Declaramos los puertos del detector que van a conectarse al exterior.

```

entity detector_ip_v1_0 is
    generic (
        C_S_AXI_DATA_WIDTH : integer := 32;
        C_S_AXI_ADDR_WIDTH : integer := 4
    );
    port (
        -- Users to add ports here
        -----
        sig_secuencia_i : in std_logic;
        clk_i           : in std_logic;
        sig_detec_o     : out std_logic;
        fail_o          : out std_logic;
    );
end entity;

```

Declaramos los puertos del componente.

```

architecture arch_imp of detector_ip_v1_0 is
    -- component declaration
    component detector_ip_v1_0_S_AXI is
        generic (
            C_S_AXI_DATA_WIDTH : integer := 32;
            C_S_AXI_ADDR_WIDTH : integer := 4
        );
        port (
            -----
            SIG_SECUENCIA_i : in std_logic;
            CLK_i           : in std_logic;
            SIG_DETEC_o     : out std_logic;
            FAIL_o          : out std_logic;
        );
    end component;
end architecture;

```

Conectamos los puertos de la instancia.

```
-- Instantiation of Axi Bus Interface S_AXI
detector_ip_v1_0_S_AXI_inst : detector_ip_v1_0_S_AXI
generic map (
    C_S_AXI_DATA_WIDTH  => C_S_AXI_DATA_WIDTH,
    C_S_AXI_ADDR_WIDTH  => C_S_AXI_ADDR_WIDTH
)
port map (
    -----
    SIG_SECUENCIA_i => sig_secuencia_i,
    CLK_i => clk_i,
    SIG_DETEC_o => sig_detec_o,
    FAIL_o => fail_o,
    -----
)
```

## Detector ip v1 S AXI

Declaramos los puertos del detector.

```
entity detector_ip_v1_0_S_AXI is
generic (
    C_S_AXI_DATA_WIDTH  : integer    := 32;
    -- Width of S_AXI address bus
    C_S_AXI_ADDR_WIDTH  : integer    := 4
);
port (
    -- Users to add ports here
    -----
    SIG_SECUENCIA_i : in std_logic;
    CLK_i           : in std_logic;
    SIG_DETEC_o     : out std_logic;
    FAIL_o          : out std_logic;
    -----
)
```

Declaramos el componente detector y las señales auxiliares para los registros.

```
----- signal -----
signal slv_reg0_detc: std_logic_vector(31 downto 0);
signal slv_reg1_err: std_logic_vector(31 downto 0);

----- componente detector -----
component detector is
port(
    ----- i/o -----
    sig_secuencia_i : in std_logic;      -- signal in.
    clk_i           : in std_logic;      -- clock.
    sig_detec_o     : out std_logic;      -- detec sign
    fail_o          : out std_logic;      -- fail.
    ----- Registros -----
    detc_o          : out std_logic_vector(31 downto 0);
    err_o           : out std_logic_vector(31 downto 0)
);
end component;
```

Conectamos los puertos de la instancia.

```
-- Add user logic here
-----
inst_detec:detector
port map (
    ----- i/o -----
    sig_secuencia_i => SIG_SECUENCIA_i,
    clk_i => CLK_i,
    sig_detec_o => SIG_DETEC_o,
    fail_o => FAIL_o,

    ----- Registros -----
    detc_o => slv_reg0_detc,
    err_o => slv_reg1_err
);
-----
-- User logic ends
```

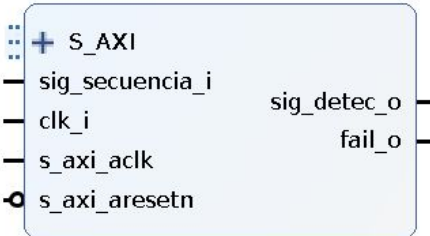
Actualizamos los registros de trabajo.

```
process (slv_reg0_detc, slv_reg1_err, slv_reg2, slv_reg3, axi_araddr, S_AXI_ARESETN, slv_reg_rden)
variable loc_addr :std_logic_vector(OPT_MEM_ADDR_BITS downto 0);
begin
    -- Address decoding for reading registers
    loc_addr := axi_araddr(ADDR_LSB + OPT_MEM_ADDR_BITS downto ADDR_LSB);
    case loc_addr is
        when b"00" =>
            reg_data_out <= slv_reg0_detc;
        when b"01" =>
            reg_data_out <= slv_reg1_err;
        when b"10" =>
            reg_data_out <= slv_reg2;
        when b"11" =>
            reg_data_out <= slv_reg3;
        when others =>
            reg_data_out <= (others => '0');
    end case;
end process;
```

## Detector IP - Propiedades

El bloque de hardware detector ip.

☐ Show disabled ports



Component Name **detector\_ip\_0**

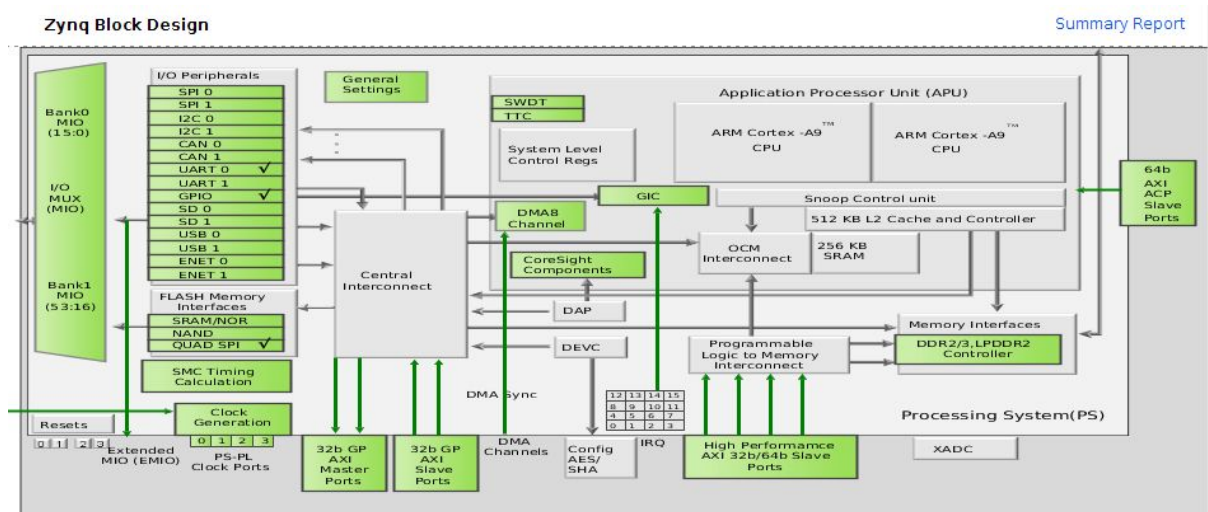
C S AXI DATA WIDTH	32
C S AXI ADDR WIDTH	4
C S AXI BASEADDR	0xFFFFFFFF
C S AXI HIGHADDR	0x00000000

## ZYNQ A9 sistema procesador

El procesador incluido en el kit de Arty Z7-10 y las facilidades de configuración que ofrece el vivado permite un manejo adecuado para las aplicaciones sencillas.

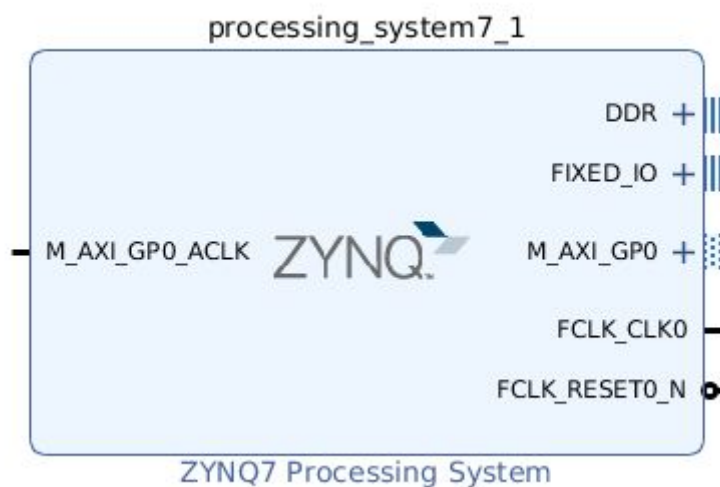
## Bloque de diseño Zynq

Presenta los siguiente configuraciones:



## ZYNQ - Propiedades

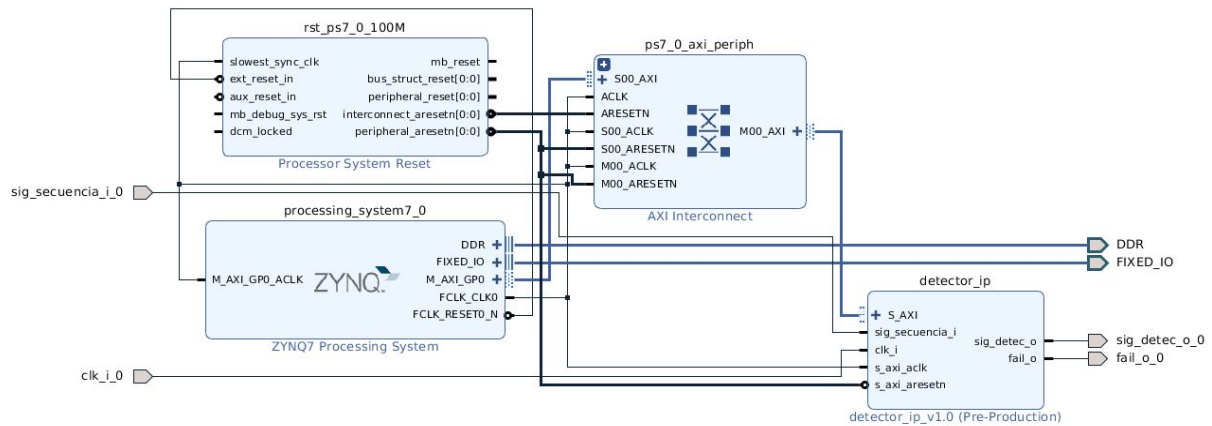
El procesador de trabajo.





## Sistema wrapper (integrado)

Realizadas las interconexiones podemos observar que nuestro sistema está compuesto por, processor system reset, axi interconnect, zynq7 processing system y detector\_ip:



## Aplicación Core detector App

Utilizando las facilidades que brinda el SDK se desarrolló el software embebido que realiza la lectura de los registros e imprime por la terminal los resultados.

## Verificación del hardware exportado

System wrapper permite conocer los cores exportados.

### Design Information

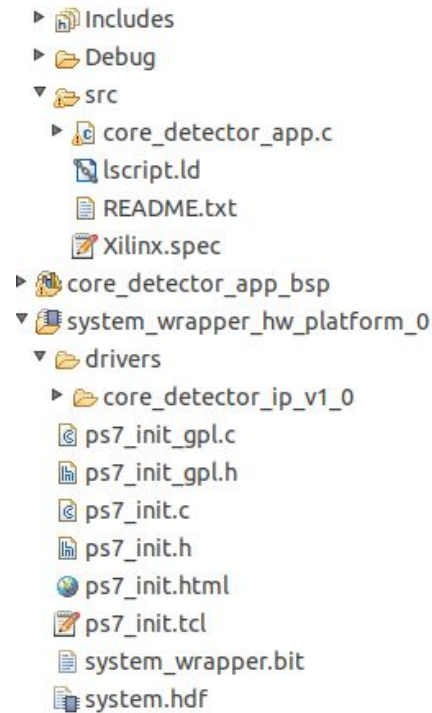
Target FPGA Device: 7z010  
 Part: xc7z010clg400-1  
 Created With: Vivado 2018.1  
 Created On: Mon Aug 24 15:05:37 2020

### Address Map for processor ps7\_cortexa9\_0-1

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
core_detector_ip	0x43c00000	0x43c0ffff	S_AXI	REGISTER
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_intc_dist_1	0xf8f02000	0xf8f02fff		REGISTER

## Xilinx SDK

Partiendo de las plantillas empty que ofrece el SDK se importó el programa base.



### Peripheral Drivers

Drivers present in the Board Support Package.

core\_detector\_ip core\_detector\_ip  
ps7\_afi\_0 generic  
ps7\_afi\_1 generic  
ps7\_afi\_2 generic

## Código - Aplicación core detector header

Al importar el hardware el SDK define las variables y las funciones a usar con el nombre del ip core en nuestro caso "CORE DETECTOR IP". Las funciones obtenidas son de lectura y escritura de registros.

```
#ifndef CORE_DETECTOR_IP_H
#define CORE_DETECTOR_IP_H

#include "xil_types.h"
#include "xstatus.h"

#define CORE_DETECTOR_IP_S_AXI_SLV_REG0_OFFSET 0
#define CORE_DETECTOR_IP_S_AXI_SLV_REG1_OFFSET 4
#define CORE_DETECTOR_IP_S_AXI_SLV_REG2_OFFSET 8
#define CORE_DETECTOR_IP_S_AXI_SLV_REG3_OFFSET 12

#define CORE_DETECTOR_IP_mWriteReg(BaseAddress, RegOffset, Data) \
    Xil_Out32((BaseAddress) + (RegOffset), (u32)(Data))

#define CORE_DETECTOR_IP_mReadReg(BaseAddress, RegOffset) \
    Xil_In32((BaseAddress) + (RegOffset))

XStatus CORE_DETECTOR_IP_Reg_SelfTest(void * baseaddr_p);

#endif
```

## Código - Aplicación detector fuente

El programa principal parte de los ejemplos dados en la materia. Comienza imprimiendo por terminal el inicio del programa y luego repite la lectura de los registros usando la función DETECTOR\_IP\_ReadReg cuyos argumentos son la dirección base del core y el offset correspondiente al registro que deseamos leer. Los registros son impresos por terminal en cada ciclo.

```
#include "xparameters.h"
#include "xil_io.h"
#include "core_detector_ip.h"

int main (void) {

    //-- Variables --
    int detec = 99;
    int fail = 99;

    //-- Inicializacion de programa ---

    xil_printf("-- Detector de tren de pulsos con IP detector propio --\r\n");
    while (1){

        //-- Read reg0 -- detec
        detec = CORE_DETECTOR_IP_mReadReg(XPAR_CORE_DETECTOR_IP_S_AXI_BASEADDR,CORE_DETECTOR_IP_S_AXI_SLV_REG0_OFFSET);

        //-- Read reg1 -- fail
        fail = CORE_DETECTOR_IP_mReadReg(XPAR_CORE_DETECTOR_IP_S_AXI_BASEADDR,CORE_DETECTOR_IP_S_AXI_SLV_REG1_OFFSET);

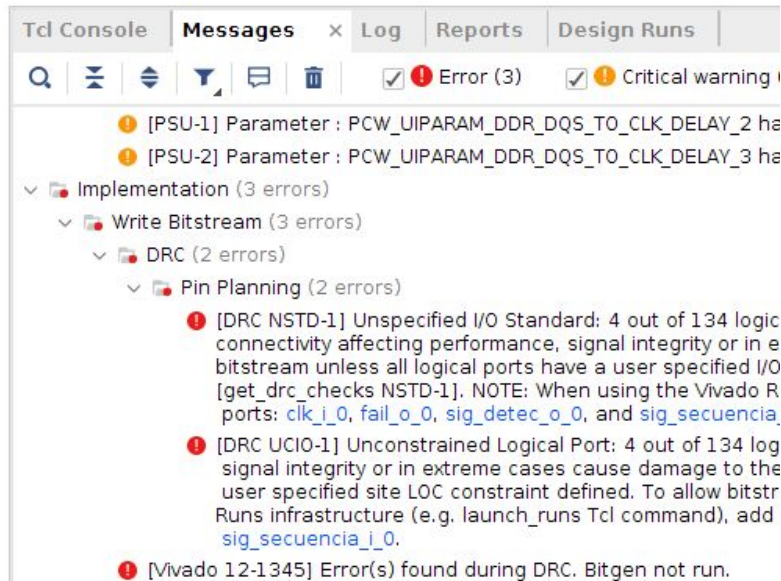
        xil_printf("detector tren de pulso: %d\r\n", detec);
        xil_printf("falla en deteccion: %d\r\n", fail);

        sleep(1);
    }
}
```

## BITSTREAM

### Fallas en generación de Bitstream

El proyecto logra pasar con éxito los procesos de síntesis e implementación, pero al momento de generar el bitstream ocurren errores que lo detienen. El estado indicado por la ventana messages hace referencia a problemas con las entradas y salidas externas del detector ip. Aun después de analizar y corregir algunos errores en los bloques de hardware del detector ip la falla persiste.

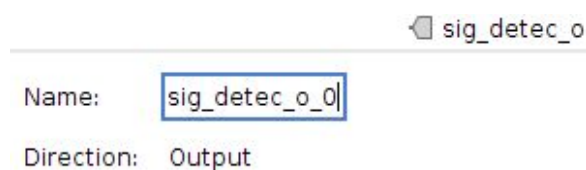


Sin el bitstream no se puede exportar el hardware completo al SDK para darle continuidad y programar la FPGA.

### Solución en generación de Bitstream

El problema como indica el error provenía del auto nombrado que realiza el vivado cuando agregamos un IP nuevo asignando la el “\_0” a cada salida y entrada. La solución fue renombrar las salidas y entradas.

Error nombres



Correcto nombres



Luego de la corrección se puede generar el .bit

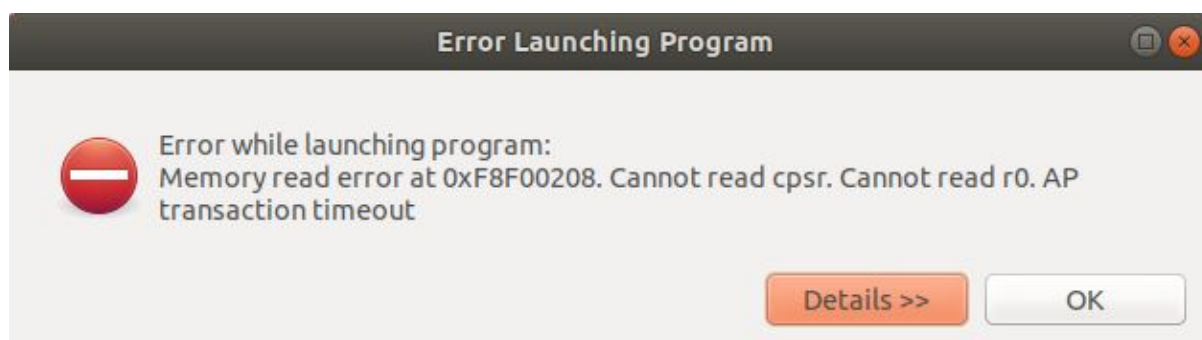
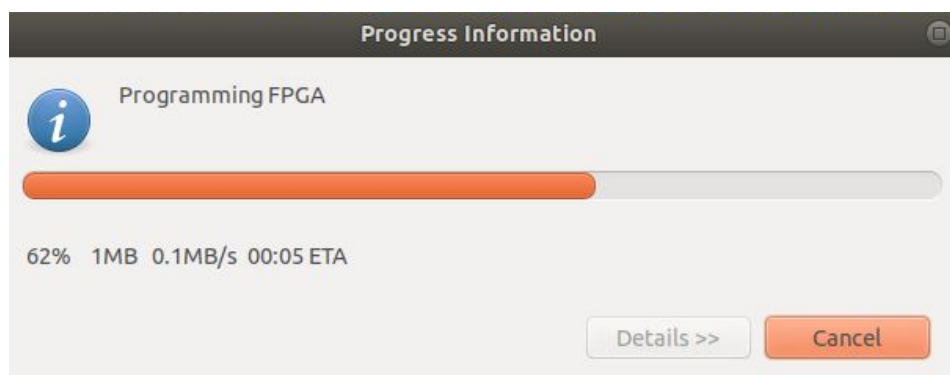
Name	Constraints	Status
✓ <b>synth_1</b> (active)	<b>constrs_1</b>	<b>synth_design Complete!</b>
✓ impl_1	constrs_1	write_bitstream Complete!
Out-of-Context Module Runs		
✓ system		Submodule Runs Complete
✓ system_processing_system7_0_0_synt...	system_pr...	synth_design Complete!
✓ system_core_detector_ip_0_0_synth_1	system_c...	synth_design Complete!
✓ system_rst_ps7_0_100M_0_synth_1	system_rs...	synth_design Complete!
✓ system_auto_pc_0_synth_1	system_a...	synth_design Complete!

## Conexión remota FPGA

La conexión remota con el servidor se realiza con éxito. Pero no logra la conexión con el ttyUSB del kit.

## Error en programación remota de FPGA remota

Visualización del proceso de programación. Después de varios intentos y solicitudes de programar con SDK la falla persiste.



## Desconexión del puerto ttyUSB

Falla al conectarse al ttyUSB1

```
artyz7-user00@lse-server-pc:~  
Archivo Editar Ver Buscar Terminal Ayuda  
[artyz7-user00@lse-server-pc ~]$ minicom -D /dev/ttyUSB1 -b 115200  
Cannot create lockfile for /dev/ttyUSB1: No existe el fichero o el directorio  
[artyz7-user00@lse-server-pc ~]$
```

Éxito al conectarse al ttyUSB0

```
artyz7-user00@lse-server-pc:~  
Archivo Editar Ver Buscar Terminal Ayuda  
[artyz7-user00@lse-server-pc ~]$ minicom -D /dev/ttyUSB0 -b 115200
```

Desde el minicom

```
artyz7-user00@lse-server-pc:~  
Archivo Editar Ver Buscar Terminal Ayuda  
  
Welcome to minicom 2.7.1  
  
OPCIONES: I18n  
Compilado en Jul 25 2019, 00:00:00.  
Port /dev/ttyUSB0, 15:48:02  
  
Presione CTRL-A Z para obtener ayuda sobre teclas especiales  
  
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Desconectado | 0
```

No hay conexión con el kit.

## **Conclusión**

Se completaron todas las etapas del proyecto desde la creación y adaptación de un IP CORE propio hasta la interacción con el procesador CORTEX A9 del ZYNQ. Si bien la funcionalidad del sistema completo depende de las entradas y salidas externas de la placa. Los registros de eventos son detectados e impresos por la terminal.

Se generó el BITSTREAM.

Se logró establecer la conexión remota pero no se puede acceder al KIT FPGA.

Los resultados obtenidos son los esperados. Continúa la depuración de errores para grabar la FPGA en forma remota y probar el sistema completo.