

# HERRAMIENTA DE ANÁLISIS DE METAESTABILIDAD EN ZYNQ-7000

Leonardo César, lcvgmez@gmail.com  
Daniel Enrique Zamora Sifredo, dezamora@gmail.com

## RESUMEN / ABSTRACT

Este artículo examina la metaestabilidad en los *flip-flops*, un comportamiento que puede generar señales inestables en los circuitos digitales. Se presenta un Circuito Test para detectar y cuantificar la ocurrencia de fallos debido a la metaestabilidad. Además, se diseña un “circuito para pruebas” para realizar experimentos utilizando el Circuito Test de metaestabilidad. Finalmente, se desarrolla una herramienta basada en el sistema operativo *FreeRTOS*® para estudiar los eventos de metaestabilidad en dispositivos *SoC-FPGA*, especialmente en los dispositivos *ZYNQ* del fabricante *Xilinx*®.

**Palabras clave:** metaestabilidad, *SoC-FPGA*, *FreeRTOS*®.

*This article investigates the phenomenon of metastability in flip-flops, a behavior that can generate unstable signals in digital circuits. A Test Circuit is presented to detect and quantify the occurrence of faults due to metastability. In addition, a “test circuit” is designed to carry out experiments using the Test Circuit for metastability. Finally, a tool based on the FreeRTOS® operating system is developed to study metastability events in SoC-FPGA devices, especially in ZYNQ devices manufactured by Xilinx®.*

**Keywords:** metastability, *SoC-FPGA*, *FreeRTOS*®.

**Title:** “ZYNQ-7000 METASTABILITY ANALYSIS TOOL”.

## 1. -INTRODUCCIÓN

La metaestabilidad es un comportamiento anormal de la salida de un *flip-flop* (*FF*), que afecta el funcionamiento de los circuitos digitales, ya que se generan señales inestables. Si un *flip-flop*, que es el elemento básico para conformar circuitos secuenciales síncronos (*CSS*) es vulnerable a este comportamiento entonces se puede afirmar que cualquier *CSS* es vulnerable. La Figura 1 muestra el comportamiento normal de un *FF* y el comportamiento metaestable. Cuando se cumplen los requerimientos temporales del *flip-flop*: tiempo de *set-up* (*tsu*) y tiempo de *hold* (*th*), su salida *Q* se establece en un valor lógico válido un tiempo *tpd* posterior al frente activo de la señal de reloj. Cuando ocurre metaestabilidad producto de la violación de los requerimientos temporales del *flip-flop*, la salida *Q* se establece en un valor lógico en un tiempo equivalente a (*tpd* + *tMET*).

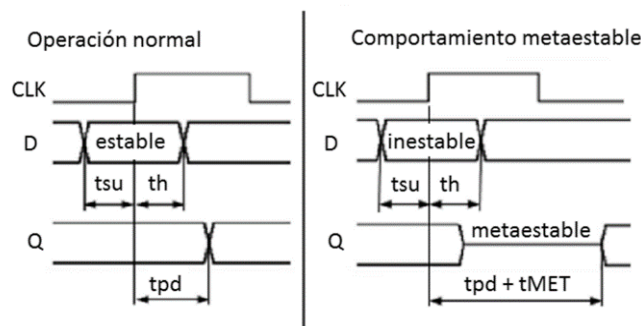


Figura 1: comportamiento normal de un *flip-flop* (a), comportamiento metaestable de un *flip-flop* (b).

La expresión 1 muestra la probabilidad que la condición de metaestabilidad persista más allá de un tiempo  $(tpd + tMET)$  después del frente positivo del reloj; donde  $T$  es una constante de tiempo, determinada por la tecnología de fabricación del FF.

$$P(tMET) = e^{-\frac{tpd+tMET}{T}} \quad (1)$$

De esta ecuación se deduce que mientras más tiempo transcurra, hay menos probabilidad que la condición de metaestabilidad persista, siendo este el principio en el que se basan los sincronizadores. La Figura 2 muestra un sincronizador de dos etapas. A *DFF1* (*DFF*, *flip flop tipo D*) llega una señal asincrónica, por ejemplo, una señal que proviene de otro dominio de reloj; por tanto, no se puede garantizar que esta señal cumpla con los requerimientos temporales de *DFF1* ( $tsu$  y  $th$ ).

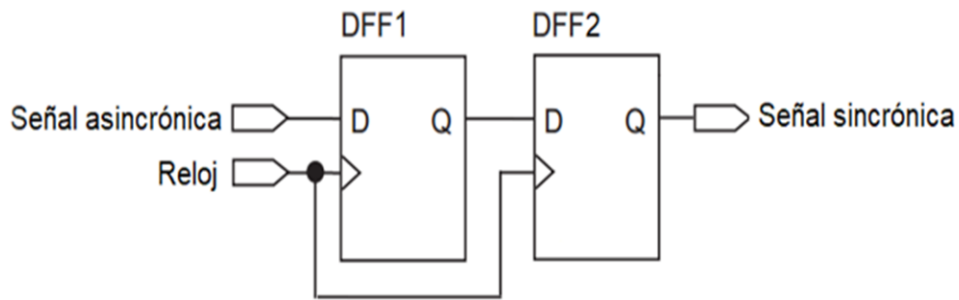


Figura 2: sincronizador de dos etapas.

La señal asincrónica puede ser inestable en la ventana de tiempo  $(tsu + th)$  alrededor del frente activo de la señal de reloj (Comportamiento metaestable en la Figura 1). Si esto ocurre la salida de *DFF1* puede resultar en metaestabilidad; sin embargo, en un tiempo  $(tpd + tMET)$  saldrá de este estado (probablemente) y alcanzará un valor lógico válido sin afectar el comportamiento de *DFF2*, como se muestra en la Figura 3.

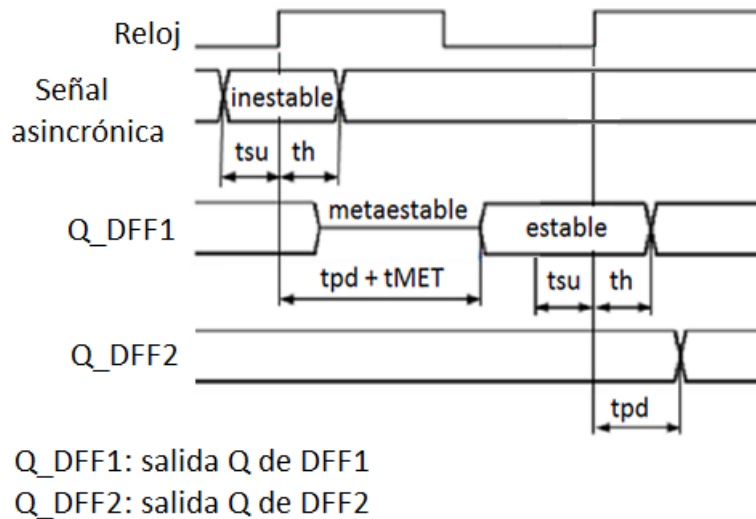


Figura 3: diagrama de tiempo del funcionamiento de un sincronizador de dos etapas.

El comportamiento metaestable de la salida de *DFF1* puede persistir más allá de  $(tpd + tMET)$ . En este caso se dice que el sincronizador falla. Mediante el tiempo medio entre fallos del sincronizador (*MTBF*) se puede

cuantificar la probabilidad de que el sincronizador falle. El *MTBF* es el principal indicador utilizado en los análisis temporales de metaestabilidad, expresa el tiempo promedio entre dos fallos consecutivos del sincronizador (expresión 2). Este indicador se expresa como función de *tMET* y depende de constantes relacionadas con las características eléctricas del *flip-flop*, la frecuencia de la señal de reloj y la frecuencia con que varía la señal de entrada de datos (señal asincrónica).

$$MTBF(tMET) = \frac{\left( e^{\frac{tpd+tMET}{T}} \right)}{w * fclk * fd} \quad (2)$$

Donde:

- **W:** constante de tiempo, determinada por la tecnología de fabricación del flip-flop, se define como la ventana de tiempo susceptible a metaestabilidad, asociada al frente de activo del reloj,
- **fclk:** frecuencia de la señal de reloj y
- **fd:** frecuencia de la señal asincrónica.

## 2.- CIRCUITO TEST DE METAESTABILIDAD

La Figura 4 muestra un ejemplo de Circuito test [1] para detectar comportamiento metaestable, este circuito genera la condición para que ocurra metaestabilidad y cuantifica la ocurrencia de fallos producto del comportamiento metaestable del *FF*.

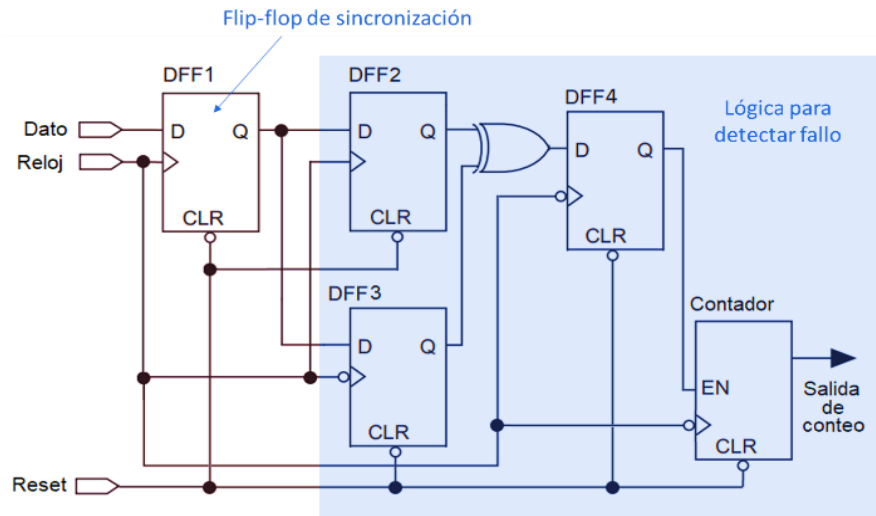


Figura 4: Circuito test de metaestabilidad.

El Circuito test cuenta con una etapa para generar la condición de metaestabilidad y otra para detectar esta condición. *DFF1* recibe una señal asincrónica en la entrada de datos. La salida del *flip-flop DFF1* se conecta a dos *flip-flops* en paralelo (*DFF2* y *DFF3*). *DFF2* y *DFF3* difieren en el frente activo de su señal de reloj, la señal de reloj de *DFF2* es activa con los frentes positivos y la de *DFF3* es activa con los frentes negativos. Cuando el circuito opera de forma normal, los cambios en la señal de datos (Dato) se registran en los *flip-flops DFF1*, *DFF2* y *DFF3* en distintos instantes de tiempo. La lógica conformada por el XOR y *DFF4* es capaz de detectar cuando los *flip-flops DFF2* y *DFF3* registran datos diferentes: condición de error. El contador registra la cantidad de fallos detectados.

La Figura 5 muestra el comportamiento normal del Circuito test. Cuando el circuito opera de forma normal el cambio en la señal de datos se registra con el frente positivo de la señal de reloj posterior al cambio en dato (*t1*)

y se refleja en la salida de *DFF1*, con el frente negativo siguiente ( $t_2$ ) se refleja en la salida de *DFF3* y finalmente en el próximo frente positivo ( $t_3$ ) lo hace en la salida de *DFF2*. El *XOR* compara las salidas de *DFF2* y *DFF3*, estas salidas solo son diferentes dentro del período de tiempo en que aún *DFF2* no ha actualizado su salida (intervalo de  $t_2$  a  $t_3$ ), en este intervalo la salida del *XOR* es '1'. Como *DFF4* registra con los frentes negativos de la señal de reloj, nunca detecta la salida del *XOR* en '1'. Por tanto, cuando el circuito opera de forma normal nunca la salida de *DFF4* vale '1'.

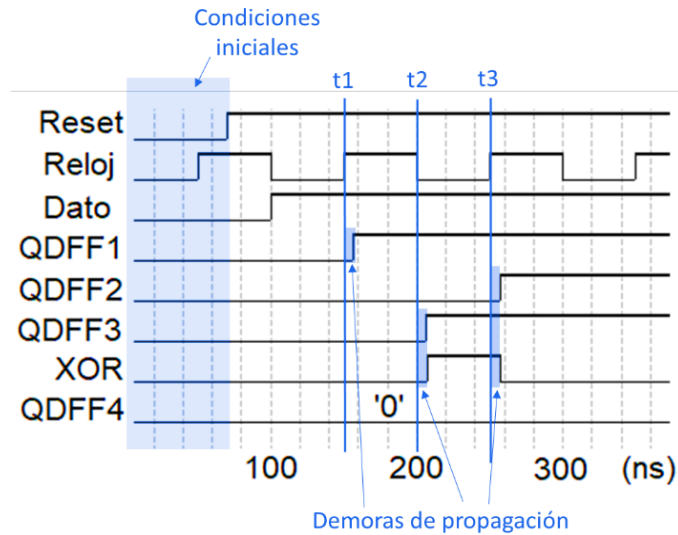


Figura 5: análisis temporal de comportamiento normal del circuito test.

Si la salida de *DFF1* resulta en estado metaestable, esta demorará un tiempo indefinido en alcanzar un valor lógico estable y puede que los *flip-flops* *DFF2* y *DFF3* registren datos diferentes, en este caso *DFF4* registrará un '1' a la salida del *XOR*.

Como muestra la Figura 6, es posible establecer una relación entre el tiempo con que dispone el *flip-flop* *DFF1* para alcanzar un valor estable sin afectar el circuito, si su salida resultó en estado metaestable y el tiempo en '1' de la señal de reloj.

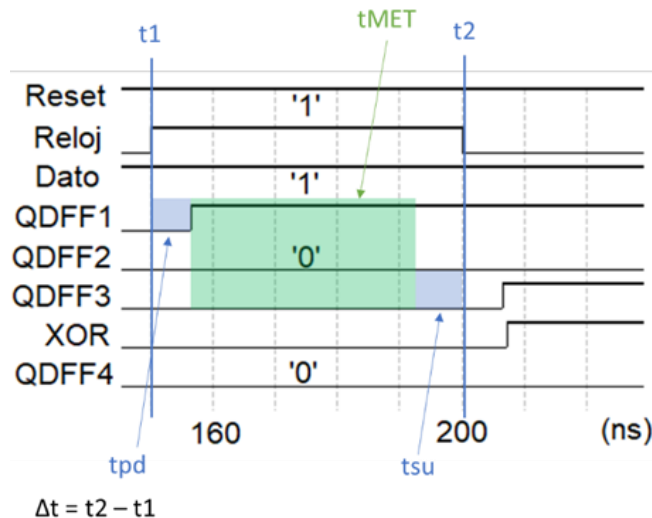


Figura 6: relación entre el tiempo con que dispone el *DFF1* para alcanzar un valor estable sin afectar el circuito, si su salida resultó en estado metaestable y el tiempo en '1' de la señal de reloj.

La relación anterior se ve reflejada en la expresión 3.

$$t_{MET} = \Delta t - (tpd + tsu) \quad (3)$$

Donde:

- **tMET**: tiempo extra a la demora de propagación de las salidas  $Q$  de  $DFF1$  que puede mantenerse la salida del  $DFF1$  inestable sin afectar el funcionamiento del circuito,
- **$\Delta t$** : tiempo en ‘1’ de un período de la señal de reloj,
- **tpd**: demora de propagación de las señales  $Q$  de  $DFF1$  más la demora desde la salida de  $DFF1$  a la entrada de datos de  $DFF3$  y
- **tsu**: tiempo de *set-up* del *flip-flop*  $DFF3$ .

### 3.- CIRCUITO PARA PRUEBAS

Para el desarrollo de experimentos utilizando el Circuito test de metaestabilidad se diseña un “circuito para pruebas”. La Figura 7 muestra un esquema simplificado (a nivel de bloques funcionales) del circuito para pruebas. Este circuito tiene tres dominios de reloj: *RELOJ\_ADTO\_ASINCRONICO*, *RELOJ\_BUS* y *RELOJ\_CIRCUITO\_TEST*.

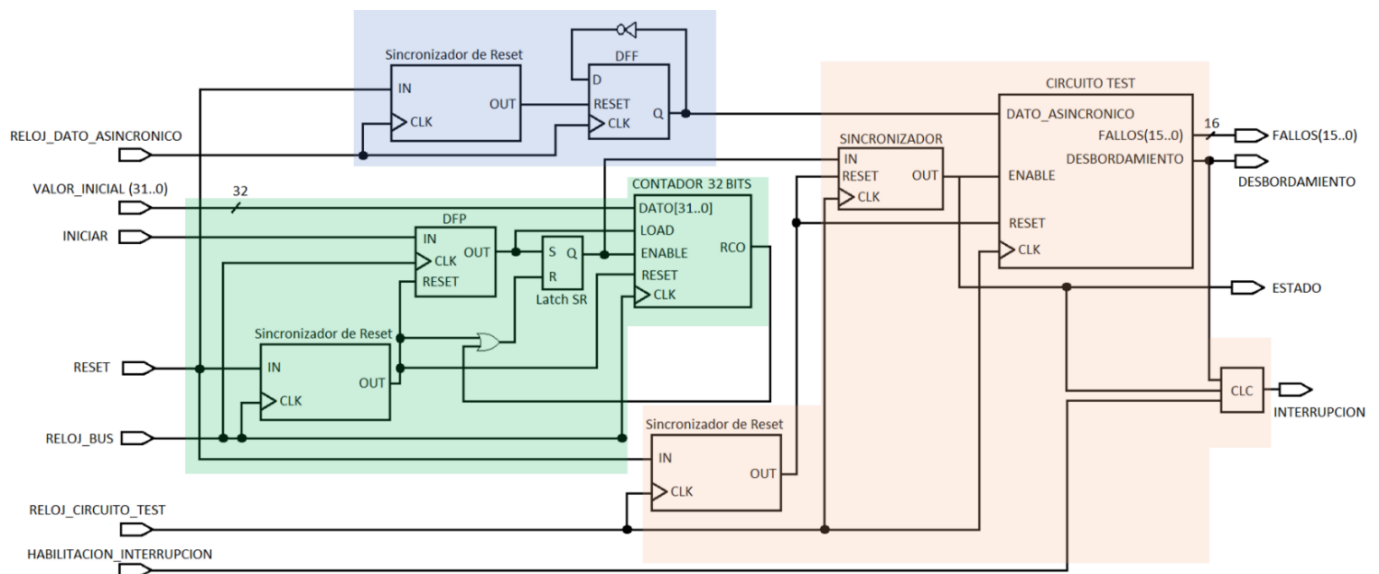


Figura 7: esquema del circuito para pruebas

Las funciones principales de este circuito se resumen a:

1. Generar la señal de datos del circuito test. Esta señal la genera la sección del circuito resaltada en azul. Un *FDD* cambia su estado cada vez que ocurre un frente positivo de la señal de reloj *RELOJ\_DATO\_ASINCRONICO*.
2. Controlar la duración del experimento. La sección del circuito resaltada en verde se encarga de llevar el control del experimento. Cuando la señal *INICIAR* va a ‘1’ se carga el valor inicial del conteo en el contador de 32 bits y luego se habilita el contador. También se habilita el contador de eventos del Circuito test para que registre los eventos metaestables. Cuando el contador de 32 bits se desborda se deshabilita este contador y el contador de eventos del Circuito test quedando registrada en la señal de salida *FALLOS* la cantidad de eventos metaestables detectados.
3. Genera una señal de solicitud de interrupción cuando finaliza el experimento u ocurre desbordamiento de contador de eventos del Circuito test.

La siguiente tabla muestra una descripción detallada de las entradas y salidas del circuito para pruebas de metaestabilidad:

*Tabla 1: entradas y salidas de circuito para pruebas.*

	Señales	Descripción
Entradas	RELOJ DATO ASINCRONICO	Reloj para generar la señal de dato asincrónico del Circuito test
	VALOR_INICIAL	Valor que carga el contador de tiempo cuando la señal <i>INICIO</i> pasa de ‘0’ a ‘1’.
	INICIAR	Control del experimento. Cuando esta señal pasa de ‘0’ a ‘1’ se inicia un experimento. Se establecen las condiciones iniciales y se habilitan los contadores de tiempo y eventos.
	RESET	<i>Reset</i> global del circuito.
	RELOJ_BUS	Reloj de la sección del circuito encargada del control del experimento.
	RELOJ_CIRCUITO_TEST	Reloj del Circuito test.
Salidas	HABILITACION_INTERRUPTCION	Habilitación de la solicitud de interrupción del circuito. ‘0’: Interrupción deshabilitada. ‘1’: Interrupción habilitada.
	FALLOS	Indica la cantidad de fallos detectados cuando finaliza el experimento.
	DESBORDAMIENTO	Indica si el contador de eventos se desbordó durante el experimento. ‘0’: No ocurrió desbordamiento. ‘1’: Ocurrió desbordamiento.
	ESTADO	Indica el estado del circuito. ‘0’: Listo para realizar el experimento. ‘1’: Ocupado. Cuando esta señal pasa de ‘1’ a ‘0’ indica que finalizó el experimento.
	INTERRUPCION	Solicitud de interrupción. ‘0’: Desactivada. ‘1’: Activada.

## 4.- DISEÑO DE HERRAMIENTA PARA EXPERIMENTOS DE METAESTABILIDAD

Se decide diseñar una herramienta para el estudio de los eventos de metaestabilidad en dispositivos *SoC-FPGA*, especialmente en los dispositivos *ZYNQ* del fabricante *Xilinx* utilizando los circuitos anteriormente descritos. Esta herramienta debe ser capaz de permitir al usuario configurar de manera simple todos los campos necesarios para realizar un experimento de estrés en las celdas del de la zona reconfigurable del dispositivo. Además, la herramienta debe ser capaz de iniciar uno o un grupo de experimentos consecutivos, almacenar los resultados de dichos experimentos y generar los informes correspondientes a los resultados obtenidos, para su posterior análisis.

Para lograr esto, evidentemente se necesita un medio de comunicación con el dispositivo que facilite una interfaz amigable para el usuario. En este sentido, se utiliza el sistema de procesamiento *ARM* integrado en esta familia de *SoC-FPGA* junto con una de las interfaces *UART* para implementar la interfaz de comunicación y el sistema de configuración y gestión de experimentos. La Figura 8 muestra un esquema simplificado de la estructura interna de la herramienta a nivel de bloques funcionales.

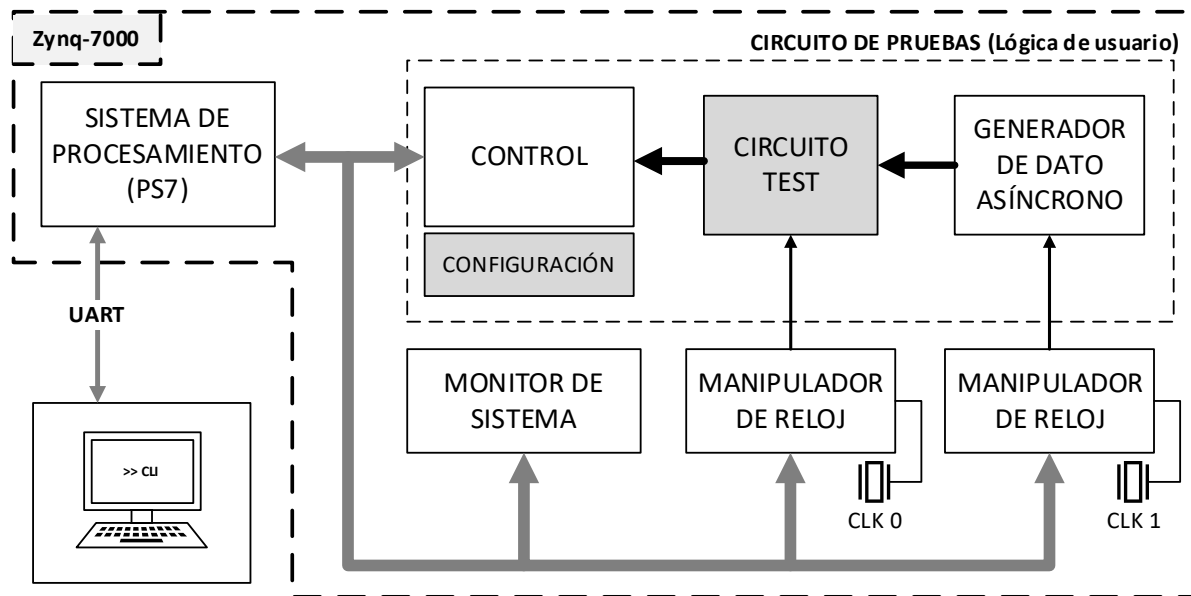


Figura 8: diagrama de bloques funcionales del sistema.

En este esquema se implementa el circuito de pruebas encapsulado en un módulo de propiedad intelectual (IP) que requiere de dos entradas de reloj independientes (una para el circuito test y otra para el generador de datos asíncronos). Este circuito también posee una interfaz *AXI-Lite* (como esclavo) para ser configurado por un sistema de procesamiento externo (en este caso el Zynq-7000) y facilita un mecanismo de interrupción que identifica el fin de un experimento [2].

El acceso a las señales de entrada y salida del módulo IP desarrollado se realiza a través de tres registros: Control, Carga y Estado. En la Figura 9 se muestra la descripción de los registros Control y Estado, en el registro Carga se escribe el valor inicial del contador de tiempo. La salida *INTERRUPCION* del módulo IP se conecta a un puerto de salida del módulo para ser conectada a una de las entradas de interrupción del sistema de procesamiento.

Registro Control					
bit 32	(...)	bit 1	bit 0		
X	X	HABILITACION_INTERRUPCION	INICIAR		

Registro Estado					
bit 32	(...)	bit 18	bit 17	bit 16	bits [15..0]
X		ESTADO	INTERRUPCION	DESBORDAMIENTO	FALLOS

Figura 9: descripción de los registros de control y estado.

El módulo IP puede ser atendido por encuesta o por interrupción. La figura 10 y 11 muestran los diagramas de flujo del segmento de programa principal encargado de realizar un experimento y de la subrutina de atención a la interrupción del Circuito para pruebas.

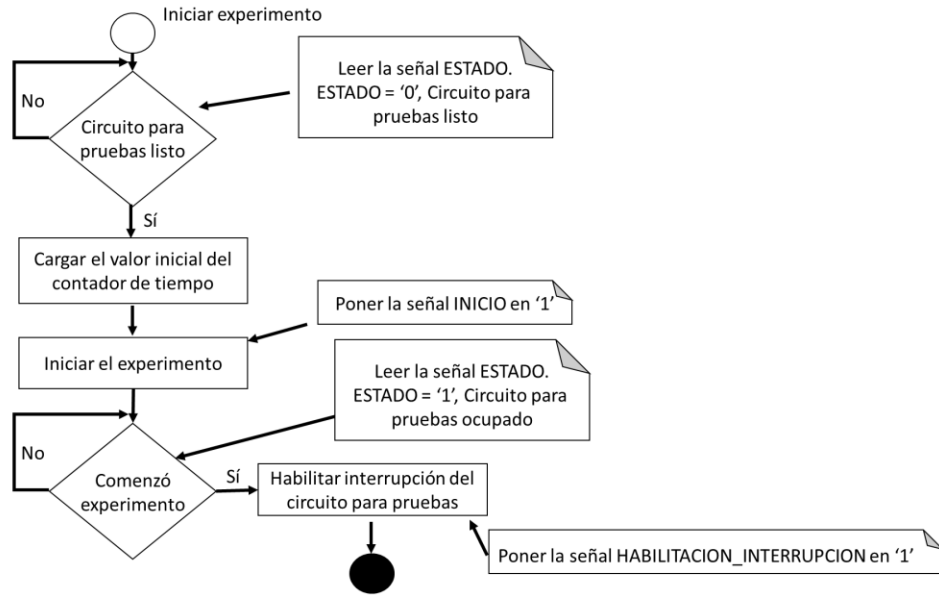


Figura 10:diagrama de flujo para realizar un experimento con el módulo IP diseñado.

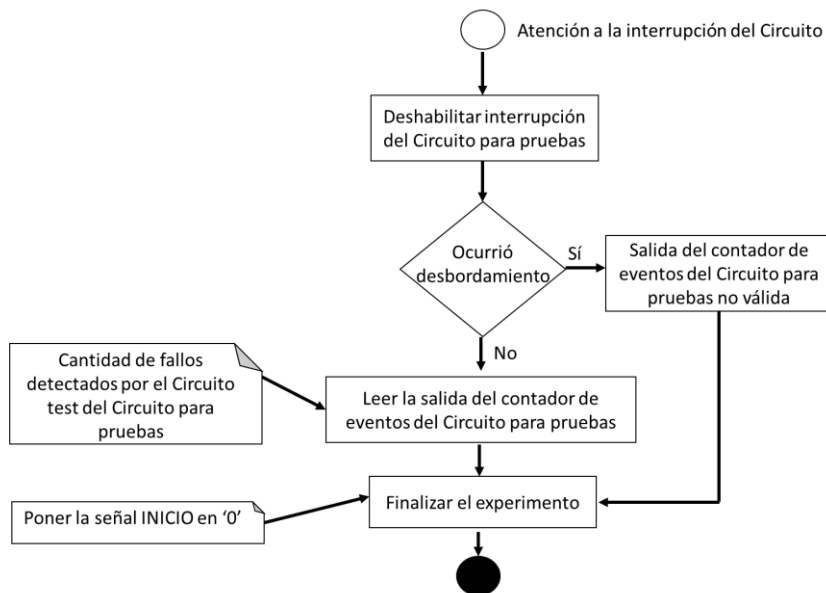


Figura 11:diagrama de flujo de subrutina de atención del módulo IP por interrupción.

El sistema de procesamiento además de proporcionar la interfaz de comunicación con el usuario, también es el encargado de ajustar los circuitos de manipulación de reloj que se conectan a la lógica de usuario (módulos de propiedad intelectual de *Xilinx*) [3]. EL sistema de procesamiento debe atender constantemente el estado de la temperatura y las fuentes de alimentación del sistema mediante el módulo *XADC* empotrado en el *SoC-FPGA* [4].

## 4.- DESARROLLO DE FIRMWARE

Con el objetivo de aprovechar la capacidad de procesamiento disponible en estos dispositivos se decide abordar una solución de firmware basada en el sistema operativo *FreeRTOS®*. Además, este enfoque permite modularizar las tareas del firmware y lograr una solución fácilmente portable a otras plataformas de hardware al ser un sistema operativo en tiempo real de código abierto y ampliamente utilizado.



Como se utiliza una interfaz *UART* como medio de comunicación con el sistema, es necesario implementar un protocolo de comunicación para la interacción desde una aplicación en un terminal. Ya que se cuenta con un sistema de procesamiento con amplias capacidades y de alta velocidad se implementa como protocolo una interfaz de línea de comandos basada en la forma de operar de intérpretes conocidos como *bash* o *powershell* (ver figura 12).

```

PS in D:\
> terminal-s
--- COM7 is connected. Press Ctrl+] to quit ---

[Timing Diagram]

>> version
MT-TOOL 0.0.1 (ZYNQ-VERSION)
>>

```

Figura 12: captura de pantalla de terminal de comandos de la herramienta

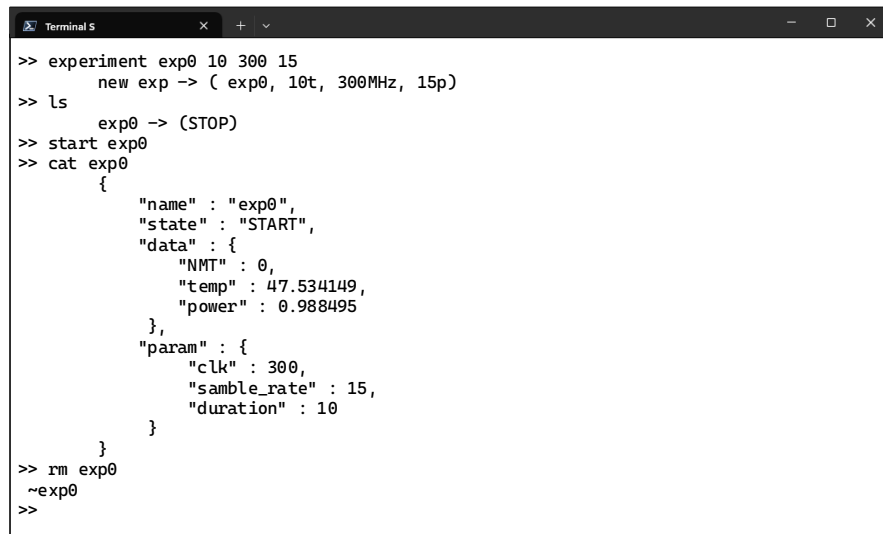
Este enfoque provoca que no sea imprescindible desarrollar una aplicación compleja en el *host* (el dispositivo que controla la herramienta) y permite que todo el procesamiento y la gestión de datos se realice de manera local en el propio *chip*. Por tanto, el usuario puede realizar desarrollar todos los experimentos y exportarlos a un directorio local utilizando herramientas simples para lanzar un terminal serie como *minicom*, *putty* o *terminal-s* (la aplicación mostrada en la figura anterior).

Para implementar el intérprete de comandos fue necesario realizar una adaptación (*port*) de la librería extendida *FreeRTOS\_PLUS\_CLI* para esta arquitectura de *hardware*, lo cual es un aporte a la comunidad ya que esta implementación aún no se encontraba disponible. El desarrollo de este componente del firmware trajo consigo delegar un hilo de ejecución (una tarea de prioridad baja) explícitamente para la gestión de la *CLI* por parte del *Zynq*. A continuación se muestra una lista de los comandos disponibles para el usuario:

- **version** : este comando devuelve la versión del firmware que se está utilizando. Este comando es especialmente útil a la hora de implementar una aplicación de alto nivel que se comunica con la herramienta desarrollada por esta vía.
- **ls** : este comando lista todos los experimentos realizados y pendientes en el orden en que fueron añadidos. Los experimentos pendientes se visualizan comuna señal de (STOP) y los ejecutados con una señal de (START).
- **experiment <name> <d> <f> <sr>** : este comando se utiliza para instanciar un nuevo experimento con los parámetros de configuración requeridos, donde el campo <name> se utiliza para asignar un identificador de menos de 20 caracteres al experimento, el campo <d> define la duración del experimento en nanosegundos , <f> se utiliza para definir la frecuencia de trabajo del *circuito test* (en esta primera versión del firmware es fija a 300MHz). Por último, el campo <sr> se utiliza para definir el ciclo útil de la señal en un valor porcentual.
- **start <name>** : este comando da inicio al experimento identificado como <name>. En caso de recibir como parámetro el valor "--all" este comando ejecuta secuencialmente todos los experimentos en el orden en que fueron añadidos con el comando <experiment>.
- **cat <name>** : este comando devuelve toda la información de un experimento, tanto los parámetros de configuración, como resultados y el estado. La información se devuelve en un formato estándar tipo *json* para facilitar su lectura y posterior análisis por otras herramientas.
- **rm <name>** : este comando se utiliza para borrar un experimento de la lista. En caso de recibir el como parámetro valor de "--all" se eliminan todos los experimentos de la lista.

- **clear** : este comando se utiliza para despejar el contenido de la consola en tiempo real (posee un atajo de teclado equivalente utilizando la combinación de las teclas *Ctrl+L*).
- **help**: muestra información de todos los comandos y los modos de uso de cada uno de ellos.

Otra de las tareas (o hilos de ejecución) que se realizan en este firmware es una constante monitorización de los valores de temperatura y alimentación del chip. Esto permite adjuntar a la información de las condiciones de estrés externas sobre las que se realiza cada experimento. Para esto además de crear las estructuras de datos inherentes a estos campos también es necesario realizar configuraciones específicas sobre el periférico *XADC* para que funcione en modo secuenciador y con cálculo de promedio automático. La figura 13 muestra un ejemplo de uso de la herramienta por *CLI* donde se pueden apreciar los resultados del experimento (0 eventos de metaestabilidad) y los valores de temperatura y alimentación de la placa.



```

>> experiment exp0 10 300 15
      new exp -> ( exp0, 10t, 300MHz, 15p)
>> ls
      exp0 -> (STOP)
>> start exp0
>> cat exp0
      {
        "name" : "exp0",
        "state" : "START",
        "data" : {
          "NMT" : 0,
          "temp" : 47.534149,
          "power" : 0.988495
        },
        "param" : {
          "clk" : 300,
          "sample_rate" : 15,
          "duration" : 10
        }
      }
>> rm exp0
      ~exp0
>>
  
```

Figura 13: ejemplo de uso de la herramienta.

## 5.- Conclusiones

La metaestabilidad es un comportamiento que puede afectar el funcionamiento de los circuitos digitales, especialmente los circuitos secuenciales sincrónicos. Se diseñó un Circuito test para detectar este comportamiento y un Circuito para pruebas que permite realizar experimentos utilizando el Circuito test. Además, se desarrolló la primera versión de una herramienta para el estudio de los eventos de metaestabilidad en dispositivos *SoC-FPGA* usando un *firmware* propio basado en el sistema operativo *FreeRTOS*® que permite modularizar las tareas y es fácilmente portable a otras plataformas de hardware. En conclusión, el estudio de la metaestabilidad es esencial para el diseño y la operación eficiente de los circuitos digitales, y las herramientas y métodos desarrollados en este trabajo proporcionan una base sólida para la investigación y el análisis de este fenómeno.

## REFERENCIAS.

- [1] P. Alfke, «Metastable Recovery in Virtex-II Pro FPGAs», 2005.
- [2] «AXI Interconnect v2.1 LogiCORE IP Product Guide • AXI Interconnect LogiCORE IP Product Guide (PG059) • Reader • AMD Adaptive Computing Documentation Portal». Accedido: 29 de febrero de 2024. [En línea]. Disponible en: <https://docs.xilinx.com/r/en-US/pg059-axi-interconnect/AXI-Interconnect-v2.1-LogiCORE-IP-Product-Guide>
- [3] «Clocking Wizard v5.3 Product Guide • Viewer • AMD Adaptive Computing Documentation Portal». Accedido: 29 de febrero de 2024. [En línea]. Disponible en: <https://docs.xilinx.com/v/u/5.3-English/pg065-clk-wiz>
- [4] Xilinx, «XADC Wizard v3.3 LogiCORE IP Product Guide Vivado Design Suite». Xilinx, 5 de octubre de 2016.

## ANEXO A (DESCRIPCIÓN VHDL DE MÓDULO IP DE USUARIO)

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY Circuito_para_pruebas IS
    PORT (
        clk_bus, clk_circuito_test, clk_generador_dato : IN STD_LOGIC;
        reset                                           : IN STD_LOGIC;
        iniciar                                         : IN STD_LOGIC;
        habilitar_interrupcion                         : STD_LOGIC;
        base_conteo                                     : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
        eventos_metaestables                          : OUT STD_LOGIC_VECTOR(15 DOWNTO 0);
        desbordamiento                                : OUT STD_LOGIC;
        interrupcion                                   : OUT STD_LOGIC;
        estado                                          : OUT STD_LOGIC);
END ENTITY;

ARCHITECTURE arch OF Circuito_para_pruebas IS
    -- signals
    SIGNAL reset_sinc_circuito_test                   : STD_LOGIC;
    SIGNAL reset_sinc_bus                             : STD_LOGIC;
    SIGNAL reset_sinc_generador_dato                  : STD_LOGIC;
    SIGNAL pulso_inicio                               : STD_LOGIC;
    SIGNAL en_contador_tiempo, fin_contador_tiempo    : STD_LOGIC;
    SIGNAL en_contador_tiempo_sinc                    : STD_LOGIC;
    SIGNAL clear_contador_eventos                     : STD_LOGIC;
    SIGNAL dato_asincronico                           : STD_LOGIC;
    SIGNAL rco_contador_eventos                       : STD_LOGIC;
    SIGNAL desbordamiento_signal                       : STD_LOGIC;
    -- componentes
    COMPONENT Sincroniza_RST IS
        PORT (
            ACLR : IN STD_LOGIC;
            Clock : IN STD_LOGIC;
            EN : IN STD_LOGIC;
            SCLR : OUT STD_LOGIC);
    END COMPONENT;

    COMPONENT Sincronizador IS
        PORT (
            Reset : IN STD_LOGIC;
            CLK : IN STD_LOGIC;
            X : IN STD_LOGIC;
            Salida : OUT STD_LOGIC);
    END COMPONENT;

    COMPONENT DFP IS
        PORT (
            Reset : IN STD_LOGIC;
            CLK : IN STD_LOGIC;
            X : IN STD_LOGIC;
            Pulso : OUT STD_LOGIC);
    END COMPONENT;

    COMPONENT Contador_NBits IS
        GENERIC (
            N : INTEGER := 4;
            M : INTEGER := 10);
        PORT (
            CLK : IN STD_LOGIC;
```

```

    ACLR : IN  STD_LOGIC
    SCLR : IN  STD_LOGIC := '0';
    Enable : IN  STD_LOGIC := '0';
    Load : IN  STD_LOGIC := '0';
    Dato : IN  STD_LOGIC_VECTOR (N - 1 DOWNT0 0) := (OTHERS => '0');
    RCO : OUT STD_LOGIC;
    Q : OUT STD_LOGIC_VECTOR (N - 1 DOWNT0 0));
END COMPONENT;

COMPONENT Contador_32bits IS

    PORT (
        CLK : IN  STD_LOGIC;
        ACLR : IN  STD_LOGIC;
        SCLR : IN  STD_LOGIC := '0'; -- asincronico y activo en '1'
        Enable : IN  STD_LOGIC := '0'; -- sincrónico y activo en '1'
        Load : IN  STD_LOGIC := '0'; -- activo en '1'
        Dato : IN  STD_LOGIC_VECTOR (31 DOWNT0 0) := (OTHERS => '0');
        RCO : OUT STD_LOGIC;
        Q : OUT STD_LOGIC_VECTOR (31 DOWNT0 0));
END COMPONENT;

COMPONENT Circuito_test IS
    PORT (
        dato, clk, clr, clr_s, en : IN  STD_LOGIC; -- clr y clr_s activos en '1'
        rco : OUT STD_LOGIC;
        eventos : OUT STD_LOGIC_VECTOR(15 DOWNT0 0));
END COMPONENT;

-- end component

BEGIN

    -- Dominio de reloj clk_bus

    Sincronizador_reset_reloj_bus : Sincroniza_RST
    PORT MAP(
        ACLR => reset,
        Clock => clk_bus,
        EN => '1',
        SCLR => reset_sinc_bus);

    Detector_frente_positivo : DFP
    PORT MAP
    (
        Reset => reset_sinc_bus,
        CLK => clk_bus,
        X => iniciar,
        Pulso => pulso_inicio);

    -- latch SR para generar la señal en_contador_tiempo

    PROCESS (pulso_inicio, reset_sinc_bus, fin_contador_tiempo)
    BEGIN
        IF (pulso_inicio = '1') THEN
            en_contador_tiempo <= '1';
        ELSIF ((reset_sinc_bus OR fin_contador_tiempo) = '1') THEN
            en_contador_tiempo <= '0';
        END IF;
    END PROCESS;

    -- contador de tiempo

```

```

Contador_tiempo : Contador_32bits
PORT MAP(
    CLK    => clk_bus,
    ACLR   => reset_sinc_bus,
    SCLR   => '0',
    Enable => en_contador_tiempo,
    Load  => pulso_inicio,
    Dato   => base_conteo,
    -- para puebas
    --Q    => prueba,
    RCO    => fin_contador_tiempo);

-- Dominio de reloj clk_circuito_test

-- Sincronizador de reset, dominio de reloj clk_circuito_test

Sincronizador_reset_circuito_test : Sincroniza_RST
PORT MAP(
    ACLR   => reset,
    Clock  => clk_circuito_test,
    EN     => '1',
    SCLR   => reset_sinc_circuito_test);

-- Sincronizador para la señal en_contador_tiempo, esta señal cruza del dominio de reloj
-- clk_bus al dominio clk_circuito_test

Sincronizador_serial_en_contador_tiempo : Sincronizador
PORT MAP(
    Reset  => reset_sinc_circuito_test,
    CLK    => clk_circuito_test,
    X      => en_contador_tiempo,
    Salida => en_contador_tiempo_sinc);

-- DFP para generar la señal clear del contador de eventos del circuito test

Detector_frente_positivo_circuito_test : DFP
PORT MAP
(
    Reset => reset_sinc_circuito_test,
    CLK   => clk_circuito_test,
    X     => en_contador_tiempo_sinc,
    Pulso => clear_contador_eventos);

-- circuito test

Circuito_test_metaestabilidad : Circuito_test
PORT MAP(
    dato    => dato_asincronico,
    clk     => clk_circuito_test,
    clr     => reset_sinc_circuito_test,
    clr_s   => clear_contador_eventos,
    en      => en_contador_tiempo_sinc,
    rco     => rco_contador_eventos,
    eventos => eventos_metaestables);

-- Lógica para generar la señal desbordamiento (Latch SR)
PROCESS (reset_sinc_circuito_test, clear_contador_eventos, rco_contador_eventos)
BEGIN
    IF (rco_contador_eventos = '1') THEN
        desbordamiento_signal <= '1';
    ELSIF (clear_contador_eventos = '1' OR reset_sinc_circuito_test = '1') THEN
        desbordamiento_signal <= '0';
    END IF;
END IF;

```

```

END PROCESS;

-- Dominio de reloj clk_generador_dato

-- Sincronizador para la señal de reset en el dominio de reloj clk_generador_dato

Sincronizador_reset_generador_dato : Sincroniza_RST
PORT MAP(
    ACLR => reset,
    Clock => clk_generador_dato,
    EN    => '1',
    SCLR  => reset_sinc_generador_dato);

-- Generador de dato asincrónico (FFT) genera la señal dato_asincronico

PROCESS (clk_generador_dato, reset_sinc_generador_dato)
BEGIN
    IF (reset_sinc_generador_dato = '1') THEN
        dato_asincronico <= '0';
    ELSIF (clk_generador_dato'event AND clk_generador_dato = '1') THEN
        dato_asincronico <= NOT(dato_asincronico);
    END IF;
END PROCESS;

-- Señal de estado, mientras esta señal es '1' el experimento esta corriendo, cuando pasa a '0'
indica que
-- finalizó el experimento y puede leerse en la salida del contador de eventos el número de fa-
llos
-- detectados

estado <= en_contador_tiempo_sinc;

interrupcion <= en_contador_tiempo_sinc AND (NOT desbordamiento_signal) AND habili-
tar_interrupcion;

desbordamiento <= desbordamiento_signal;

END arch;

--Circuito test
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY Circuito_test IS

    PORT (
        dato, clk, clr, clr_s, en : IN  STD_LOGIC; -- clr y clr_s activos en '1'
        rco                        : OUT STD_LOGIC;
        eventos                   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
    );
END Circuito_test;

ARCHITECTURE arch OF Circuito_test IS

    -- Component

    COMPONENT Contador_NBits IS
        GENERIC (
            N : INTEGER := 4; -- número de bits
            M : INTEGER := 10); -- base de conteo
        PORT (
            CLK      : IN  STD_LOGIC;
            ACLR     : IN  STD_LOGIC;

```

```

        SCLR   : IN  STD_LOGIC                                := '0'; -- sincrónico y activo en '1'
        Enable : IN  STD_LOGIC                                := '0'; -- activo en '1'
        Load   : IN  STD_LOGIC                                := '0'; -- sincrónico y activo en '1'
        Dato    : IN  STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0');
        RCO     : OUT STD_LOGIC;
        Q       : OUT STD_LOGIC_VECTOR (N - 1 DOWNTO 0));
END COMPONENT;

```

```
-- señales
```

```
SIGNAL q_dff1, q_dff2, q_dff3, q_dff4 : STD_LOGIC;
```

```
SIGNAL en_contador_de_eventos : STD_LOGIC;
```

```
BEGIN
```

```
-- DFF 1
```

```
PROCESS (clk, clr)
```

```
BEGIN
```

```
    IF clr = '1' THEN
```

```
        q_dff1 <= '0';
```

```
    ELSIF clk'event AND clk = '1' THEN
```

```
        q_dff1 <= dato;
```

```
    END IF;
```

```
END PROCESS;
```

```
-- DFF 2
```

```
PROCESS (clk, clr)
```

```
BEGIN
```

```
    IF clr = '1' THEN
```

```
        q_dff2 <= '0';
```

```
    ELSIF clk'event AND clk = '1' THEN
```

```
        q_dff2 <= q_dff1;
```

```
    END IF;
```

```
END PROCESS;
```

```
-- DFF 3
```

```
PROCESS (clk, clr)
```

```
BEGIN
```

```
    IF clr = '1' THEN
```

```
        q_dff3 <= '0';
```

```
    ELSIF clk'event AND clk = '0' THEN
```

```
        q_dff3 <= q_dff1;
```

```
    END IF;
```

```
END PROCESS;
```

```
-- DFF 4
```

```
PROCESS (clk, clr)
```

```
BEGIN
```

```
    IF clr = '1' THEN
```

```
        q_dff4 <= '0';
```

```
    ELSIF clk'event AND clk = '0' THEN
```

```
        q_dff4 <= q_dff2 XOR q_dff3;
```

```
    END IF;
```

```
END PROCESS;
```

```
en_contador_de_eventos <= q_dff4 AND en;
```

```
Contador_de_eventos : Contador_NBits
```

```
GENERIC MAP(
```

```
    N => 16,
```

```
    M => 65536)
```

```
PORT MAP(
```

```

        CLK    => clk,
        ACLR   => clr,
        SCLR   => clr_s,
        Enable => en_contador_de_eventos,
        Load   => '0',
        Dato    => (OTHERS => '0'),
        RCO    => rco,
        Q       => eventos);

END arch;

-- Contador_32_bits
-- Contador de 32 bits
-- Entrada de habilitación sincrónica
-- Reset asincrónico
-- Load   sincrónico
--
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE IEEE.Std_Logic_arith.ALL; -- para usar la funcion: conv_std_logic_vector
USE IEEE.Std_Logic_unsigned.ALL; -- para poder incrementar (+)
USE ieee.numeric_std.ALL;

ENTITY Contador_32bits IS

    PORT (
        CLK    : IN  STD_LOGIC;
        ACLR   : IN  STD_LOGIC;
        SCLR   : IN  STD_LOGIC          := '0'; -- asincronico y activo en '1'
        Enable : IN  STD_LOGIC          := '0'; -- sincrónico y activo en '1'
        Load   : IN  STD_LOGIC          := '0'; -- activo en '1'
        Dato    : IN  STD_LOGIC_VECTOR (31 DOWNTO 0) := (OTHERS => '0');
        RCO    : OUT STD_LOGIC;
        Q       : OUT STD_LOGIC_VECTOR (31 DOWNTO 0));

END Contador_32bits;

ARCHITECTURE A_Contador OF Contador_32bits IS
    SIGNAL EP, EF : STD_LOGIC_VECTOR(31 DOWNTO 0);
    CONSTANT MAX : STD_LOGIC_VECTOR(31 DOWNTO 0) := (OTHERS => '1');

BEGIN

    -----
    ---- Memoria de Estados
    PROCESS (CLK, ACLR)
    BEGIN
        IF ACLR = '1' THEN
            EP <= (OTHERS => '0');
        ELSIF Rising_Edge(CLK) THEN
            EP <= EF;
        END IF;
    END PROCESS;

    -----
    ---- Lógica del Próximo Estado (CLC)

    EF <= (OTHERS => '0') WHEN SCLR = '1' ELSE
        Dato WHEN Load = '1' ELSE
        EP + 1 WHEN Enable = '1' AND EP < MAX ELSE
        (OTHERS => '0') WHEN Enable = '1' AND EP >= MAX ELSE
        EP;

    -----

```



```

---- Lógica de la Salida    (CLC)

Q  <= EP;
RCO <= '1' WHEN (EP = MAX) AND (Enable = '1') ELSE
      '0';

END;

-- Contador_N_bits
-- Contador de N Bits
-- Entrada de habilitación sincrónica
-- Reset asincrónico
-- Load   sincrónico
--
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;
USE IEEE.Std_Logic_arith.ALL; -- para usar la funcion: conv_std_logic_vector
USE IEEE.Std_Logic_unsigned.ALL; -- para poder incrementar (+)

ENTITY Contador_NBits IS
  GENERIC (N : INTEGER := 4; -- número de bits
           M : INTEGER := 10); -- base de conteo
  PORT (
    CLK      : IN  STD_LOGIC;
    ACLR     : IN  STD_LOGIC;
    SCLR     : IN  STD_LOGIC          := '0'; -- asincronico y activo en '1'
    Enable   : IN  STD_LOGIC          := '0'; -- sincrónico y activo en '1'
    Load    : IN  STD_LOGIC          := '0'; -- activo en '1'
    Dato     : IN  STD_LOGIC_VECTOR (N - 1 DOWNTO 0) := (OTHERS => '0'); -- sincrónico y activo en '1'
    RCO      : OUT STD_LOGIC;
    Q        : OUT STD_LOGIC_VECTOR (N - 1 DOWNTO 0));
END Contador_NBits;

ARCHITECTURE A_Contador_NBits OF Contador_NBits IS
  SIGNAL EP, EF : STD_LOGIC_VECTOR(N - 1 DOWNTO 0);
  CONSTANT MAX : STD_LOGIC_VECTOR(N - 1 DOWNTO 0) := conv_std_logic_vector(M - 1, N);

BEGIN

  -----
  ---- Memoria de Estados
  PROCESS (CLK, ACLR)
  BEGIN
    IF ACLR = '1' THEN
      EP <= (OTHERS => '0');
    ELSIF Rising_Edge(CLK) THEN
      EP <= EF;
    END IF;
  END PROCESS;

  -----
  ---- Lógica del Próximo Estado    (CLC)

  EF <= (OTHERS => '0') WHEN SCLR = '1' ELSE
        Dato WHEN Load = '1' ELSE
        EP + 1 WHEN Enable = '1' AND EP < MAX ELSE
        (OTHERS => '0') WHEN Enable = '1' AND EP >= MAX ELSE
        EP;

  -----
  ---- Lógica de la Salida    (CLC)

  Q  <= EP;

```

```

        RCO <= '1' WHEN (EP = MAX) AND (Enable = '1') ELSE
            '0';

END;
-- DFP
-- Detector de Frente Positivo
-- MOORE
--
LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY DFP IS
    PORT (
        Reset : IN  STD_LOGIC; -- activo en '1'
        CLK   : IN  STD_LOGIC;
        X     : IN  STD_LOGIC;
        Pulso  : OUT STD_LOGIC);
END DFP;

ARCHITECTURE A_DFP OF DFP IS
    TYPE State IS (S0, S1, S2);
    SIGNAL EP, EF : State;
BEGIN

    -----
    -- Logica Combinacional del Proximo Estado
    PROCESS (EP, X)
    BEGIN
        CASE EP IS
            WHEN S0 => IF X = '1' THEN
                EF <= S0;
            ELSE
                EF <= S1;
            END IF;
            WHEN S1 => IF X = '0' THEN
                EF <= S1;
            ELSE
                EF <= S2;
            END IF;

            WHEN S2 => IF X = '0' THEN
                EF <= S1;
            ELSE
                EF <= S0;
            END IF;

            WHEN OTHERS => EF <= S0;
        END CASE;
    END PROCESS;

    PROCESS (CLK, Reset)
    BEGIN
        IF Reset = '1' THEN
            EP <= S0;
        ELSIF Rising_Edge(CLK) THEN
            EP <= EF;
        END IF;
    END PROCESS;

    -----
    -- Logica Combinacional para la SALIDA
    Pulso <= '1' WHEN EP = S2 ELSE

```

```

        '0';
END;

-- Sincroniza_RST
---- Circuito para sincronizar la señal de CLR asincrónica

LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY Sincroniza_RST IS
    PORT (
        ACLR : IN STD_LOGIC; -- reset asincronico activo en '1'
        Clock : IN STD_LOGIC;
        EN : IN STD_LOGIC; -- activo en '1'
        -----
        SCLR : OUT STD_LOGIC); -- salida activa en '1'
    -----
END Sincroniza_RST;

```

```

ARCHITECTURE A_Sincroniza_RST OF Sincroniza_RST IS
    --DECLARACION de SEÑALES
    -----
    SIGNAL RESET_1 : STD_LOGIC;
    SIGNAL RESET_2 : STD_LOGIC;
    -----
    CONSTANT K : INTEGER := 2;
    SIGNAL EF, EP : STD_LOGIC_VECTOR(K - 1 DOWNT0 0);
    -----

```

```

BEGIN
    -----
    -- Sincronización de la señal de RESET
    PROCESS (ACLR, Clock)
    BEGIN
        IF ACLR = '1' THEN
            EP <= (OTHERS => '1');
        ELSIF Rising_Edge(Clock) THEN
            EP <= EF;
        END IF;
    END PROCESS;
    -----
    ---- CLC del Próximo ESTADO

    EF <= EP(K - 2 DOWNT0 0) & '0' WHEN EN = '1' ELSE
        (OTHERS => '1');
    -----
    ---- CLC de la SALIDA
    SCLR <= EP(K - 1);

```

```

END;

-- Sincronizador
-- Fichero : Sincronizador.VHD
-- Sincronizador
-- Registro de desplazamiento de dos etapas

```

```

LIBRARY IEEE;
USE IEEE.Std_Logic_1164.ALL;

ENTITY Sincronizador IS
    PORT (
        Reset : IN STD_LOGIC;

```

```

        CLK      : IN  STD_LOGIC;
        X        : IN  STD_LOGIC;
        Salida   : OUT STD_LOGIC);
END Sincronizador;

ARCHITECTURE A_Sincronizador OF Sincronizador IS
    SIGNAL Temp : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
    PROCESS (Reset, CLK)
    BEGIN
        IF Reset = '1' THEN
            Temp <= (OTHERS => '0');
        ELSIF Rising_Edge(CLK) THEN
            Temp <= Temp(0) & X;
        END IF;
    END PROCESS;
    -- Logica Combinacional para la SALIDA
    Salida <= Temp(1);
END;
```

## ANEXO B (FIRMWARE)

Puede acceder a los ficheros fuentes de firmware y de hardware del proyecto en el siguiente repositorio público:

- [dezamora98/metastability-tool \(github.com\)](https://github.com/dezamora98/metastability-tool)