Introducción al CDC (clock domain crossing)

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/12/14/introduccion-al-cdc-clock-domain-crossing/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

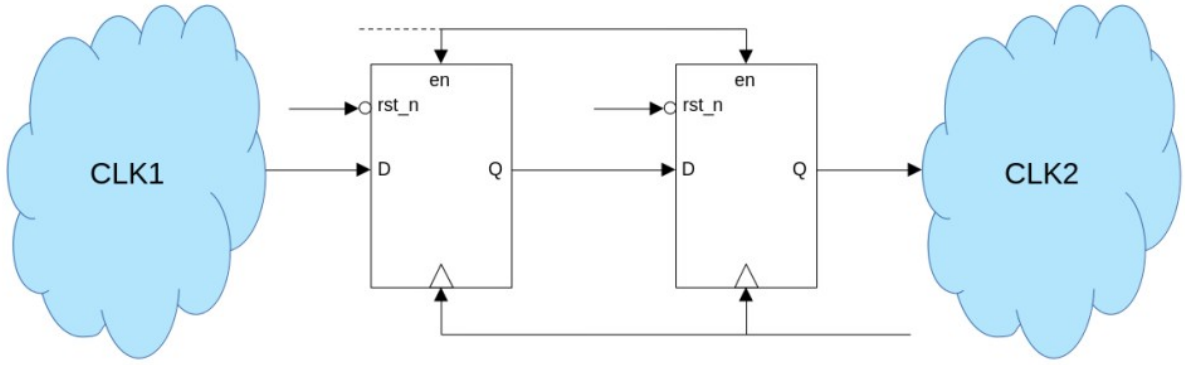
Uno de los elementos que surgen al manejar FPGAs son los dominios de tiempo diferentes al manejar señales e información. Esto provoca que manejar datos entre dominios se vuelve complejo debido a que si se tienen que comunicar hay que **sincronizar** la información con el sistema de llegada.

Para la transferencia de información hay varios métodos dependiendo de lo que se quiera transferir.

Señales

Para sincronizar señales solo hace falta hacer uso de dos biestables D que utilicen el sistema de reloj de la salida.

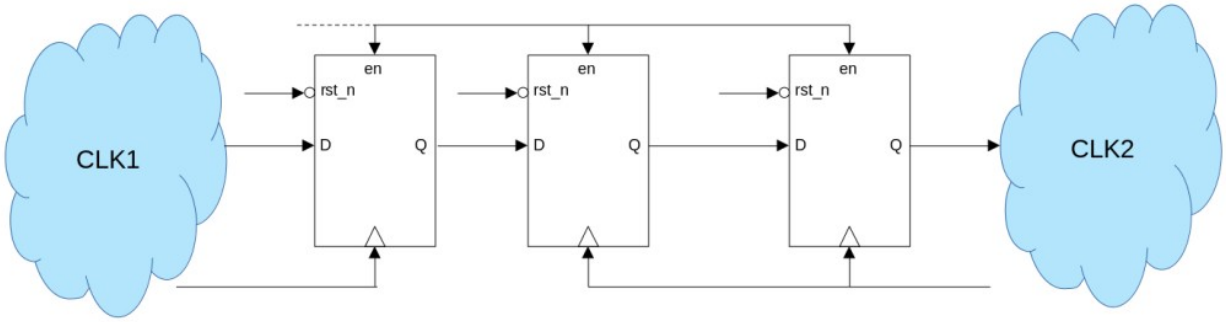
Son necesario dos biestables debido a que si solo pones un biestable, corres el riesgo de que aparezca una *metaestabilidad* en la señal de entrada al segundo sistema.



Ejemplo: Este ejemplo hace que el sintetizador sea capaz de generar dos biestables D para esta estructura.

```
process(clk2, rst_n, en)
begin
    if rst_n = '0' then
        signal_sync1 <= '0';
        signal_sync2 <= '0';
    elsif rising_edge(clk2) then
        if en = '1' then
            signal_sync1 <= signal_in;
            signal_sync2 <= signal_sync1;
        elsif en = '0' then
            signal_sync1 <= '0';
            signal_sync2 <= '0';
        end if;
    end if;
end process;
```

También es posible encontrarse con un sistema que tiene además de los dos biestables, un tercer biestable D que garantiza que la señal que sale del sistema de salida está sincronizada con el reloj de salida.



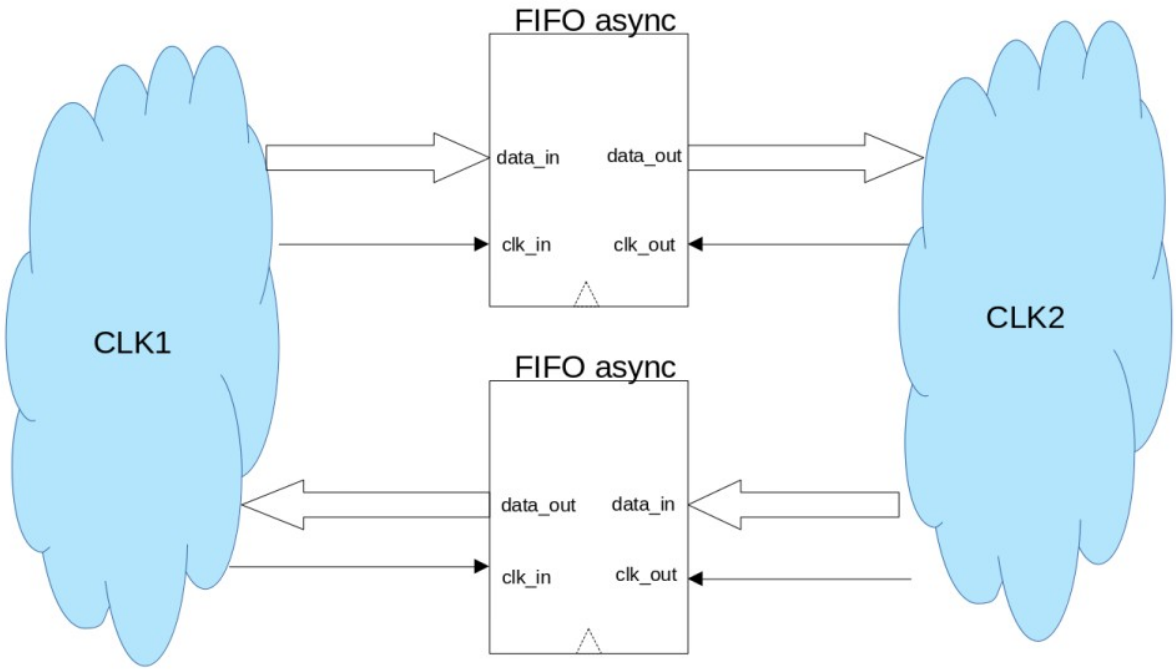
¿Qué es una metaestabilidad?

Para entender que es es necesario conocer que los sistemas digitales no son sistemas inmediatos en los que poner la señal a ‘0’ o a ‘1’ se hace inmediatamente, si no que hay un tiempo de transición entre estados, estos son llamados **tiempo de setup** *(tiempo que tarda la señal en alcanzar el valor lógico)* y **tiempo de hold***(tiempo que la señal se tiene que mantener en ese estado lógico)*. Bien, pues durante estos tiempos no se puede muestrear la señal, porque si lo haces corres el riesgo de quedarte con un valor de tensión entre el ‘0’ lógico y el ‘1’ lógico.
Entonces, si tenemos dos señales de reloj asíncronas y tenemos un biestable D que actúa como memoria, dejaría a la salida el valor lógico de la tensión no válida, entonces, al poner un segundo biestable, esta metaestabilidad desaparece, porque la sensibilidad de este segundo biestable D a la entrada lo convierte en un ‘0’ lógico, y no la deja pasar.

Datos

La transferencia de datos simples entre sistemas de tiempos se puede hacer como si fuesen señales, siguiendo el ejemplo anterior con dos biestables D.

Pero demás, también se puede utilizar un FIFO asíncrono por cada dirección, esto permite al sistema emisor o al receptor elegir cuando quieren leer los datos o escribir los datos.
Esto es importante porque cuando el sistema que genera los datos es más rápido que el sistema que los lee, porque los datos hay que dejarlos almacenados mientras sigue generando nuevos datos.
Y cuando el sistema que los lee es más rápido que el sistema que los genera, se corre el riesgo de que el sistema de lectura lea más de una vez un mismo dato, entonces, para evitar errores de lectura, se pone el FIFO.

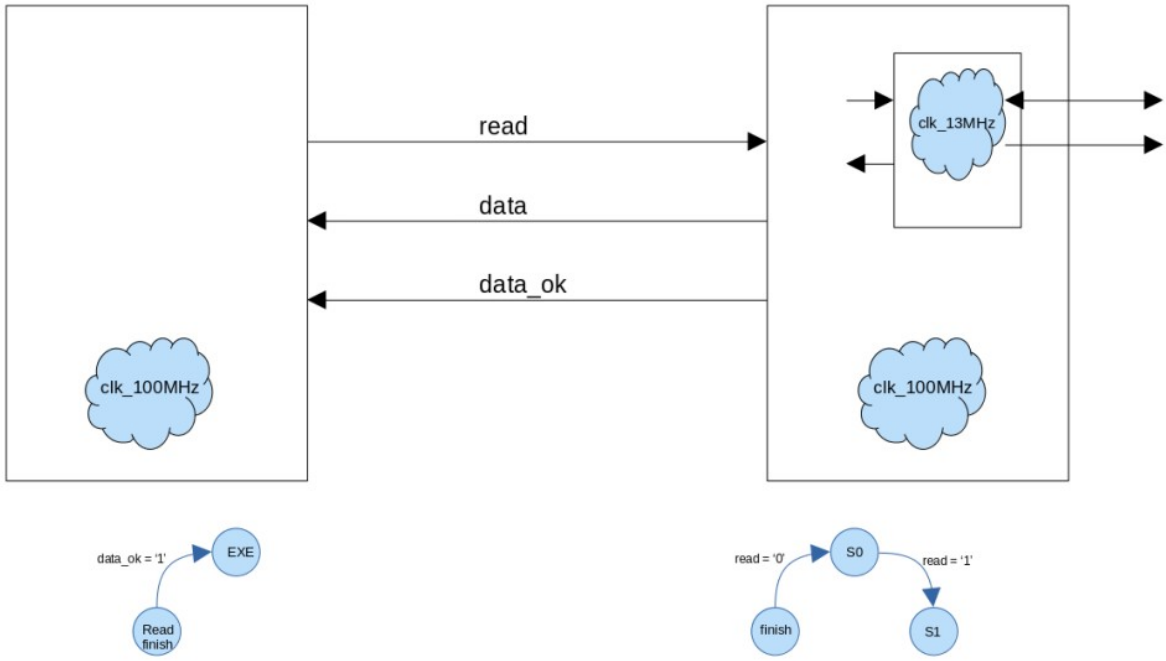


Es importante tener en cuenta que el FIFO asíncrono puede estar en otro dominio de tiempo distinto al emisor/receptor, por lo que se vuelve necesario resincronizar los datos antes de almacenarlos o sacarlos del FIFO, siempre dependiendo del sistema de tiempos que maneje el receptor de los datos.

Máquinas de estados

Para la sincronización entre máquinas de estados se recomienda un esquema como el siguiente. Este esquema se basa en que cuando el sistema rápido (de 100MHz) quiere leer de un sistema lento, le manda la orden (*read* <= ‘1’), el módulo al que se lo manda trabaja en la misma frecuencia que el rápido, solo que internamente tiene el sistema con el dominio de reloj lento. Entonces, este módulo es el que se encarga de que cualquier entrada o salida del sistema con otro dominio de reloj se haga siempre con la frecuencia de reloj principal *(el reloj principal es siempre el más rápido)*.

Una vez el sistema lento ha echo la lectura, se la pasa al módulo que resincroniza la salida y este se lo envía al módulo principal. Lo que ocurre ahora es que la máquina del sistema principal sale del estado de lectura, y lo que hace es poner la línea de lectura (*read*) a ‘0’ provocando que así el sistema de lectura pueda volver al estado inicial, con garantías de que el funcionamiento de todo el sistema sigue siendo determinista.



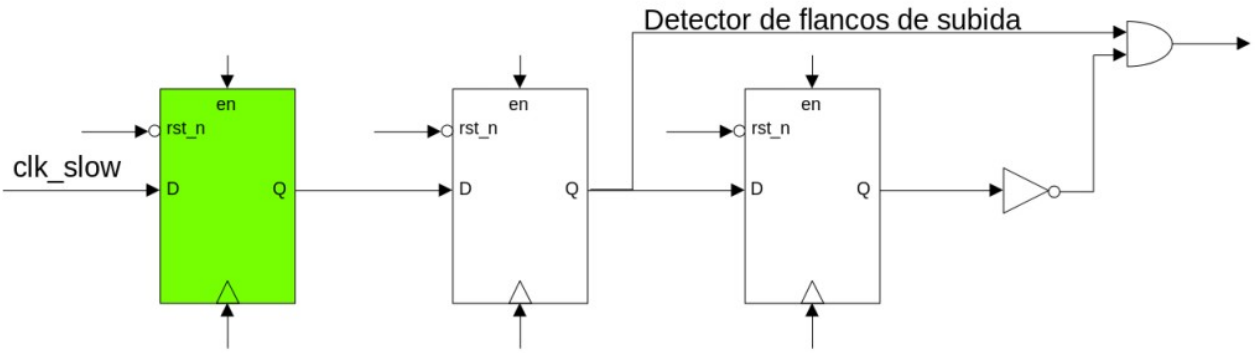
Para todo este sistema se recomienda utilizar máquinas de *estado de tipo DOW.*

https://soceame.wordpress.com/2024/11/21/maquinas-de-estados-tipo-dow/

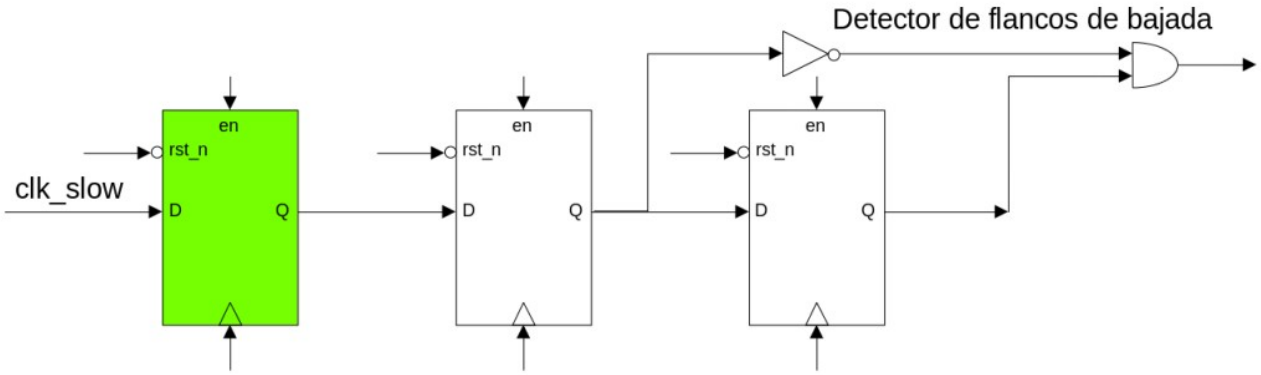
Aclaraciones

Siempre se puede escuchar la cantinela de «*utiliza el reloj lento como señales de enable del sistema rápido*«, y esto es cierto hasta determinado punto, porque corres el riesgo de realizar una múltiple lectura de un mismo enable. Para evitarlo tienes que implementar un detector de flancos.

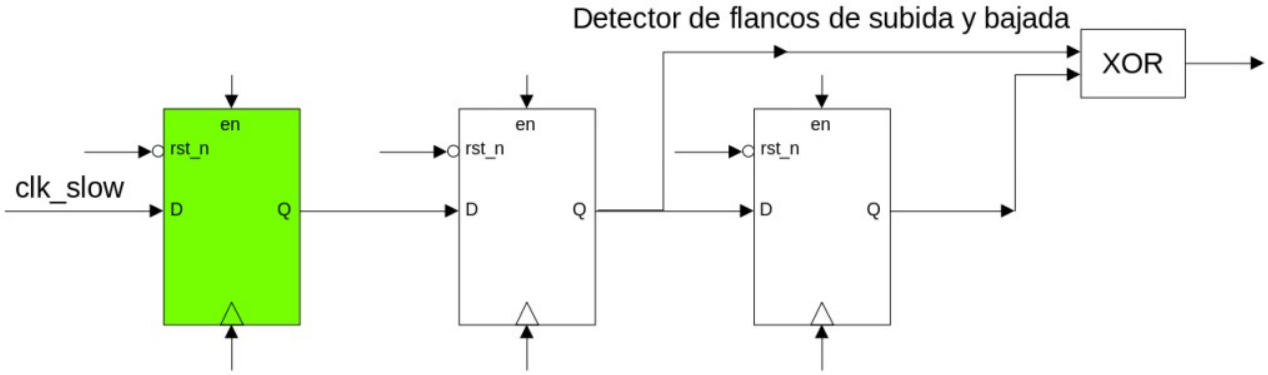
- **Detector de flancos de subida:** el detector de flancos se hace comparando la salida negada con la entrada. Si la entrada pasa de 0 a 1, la salida de la AND se vuelve 1.
  Aquí también aparece un biestable D (en verde) debido a que se tiene que sincronizar el reloj lento. Si las señales utilizadas están en el mismo dominio de reloj, el primer biestable se puede eliminar.



- **Detector de flancos de bajada:** para la detección de flancos de bajada se hace al revés que para la detección de flancos de subida. *Aquí también aparece un biestable D (en verde) debido a que se tiene que sincronizar el reloj lento. Si las señales utilizadas están en el mismo dominio de reloj, el primer biestable se puede eliminar.*



- **Detector de flancos de subida y bajada**: para la detección de flancos de subida y de bajada a la vez se tiene que utilizar una puerta **XOR**. *Aquí también aparece un biestable D (en verde) debido a que se tiene que sincronizar el reloj lento. Si las señales utilizadas están en el mismo dominio de reloj, el primer biestable se puede eliminar.*



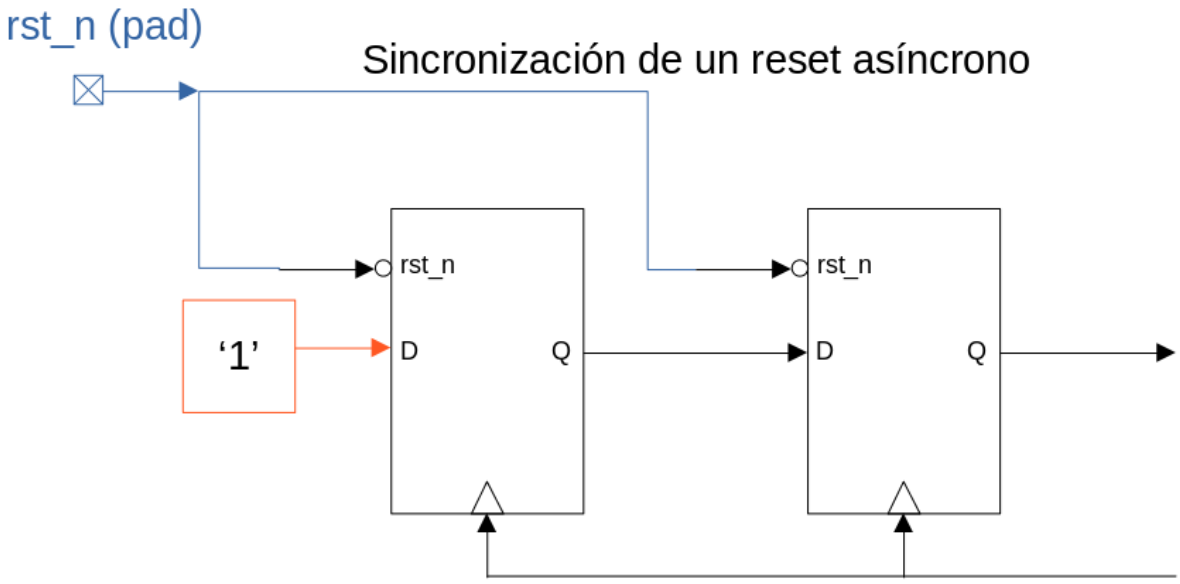
Recomendación

Todo el sistema de resincronismo es recomendable encapsularlo y meterlo dentro de los sistemas lentos, para así despreocuparse de que haya módulos de resincronismo perdidos por el firmware o señales que no se sepa de dónde vienen.

NOTA FINAL

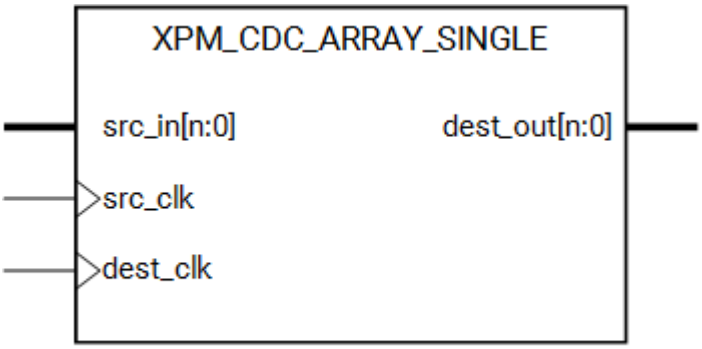
Todos los pines que vienen del mundo exterior de la FPGA se tienen que sincronizar con el reloj principal, porque se tienen que considerar como si vinieran de otro dominio de reloj, debido a que pueden generar metaestabilidades en el sistema.

El reset asíncrono de entrada también hay que resincronizarlo con el reloj principal, pero teniendo en cuenta que es una señal que se encarga de resetear toda la lógica del sistema hay que sincronizarlo de otra forma. La forma de sincronizarlo es utilizar ese reset como reset de dos biestables D, y la entrada de los biestables es un valor fijo a '1'. Si el reset asíncrono se activa, la salida de los biestables es '0'.



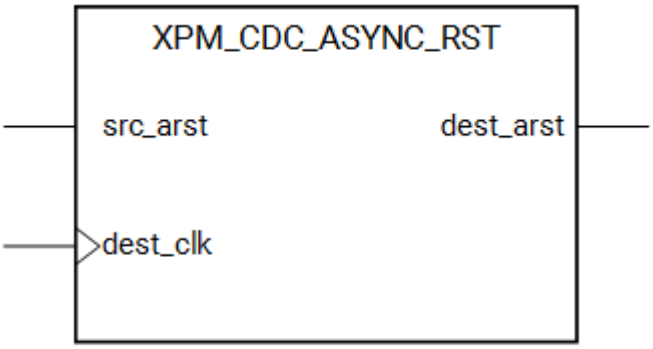
NOTA: Si trabajas en Xilinx, tienes dos opciones: o te sincronizas el sistema tú mediante FW, o puedes utilizar las macros que da Xilinx.

- **XPM_CDC_ARRAY_SINGLE**: esta macro sincroniza un dato de n bits.



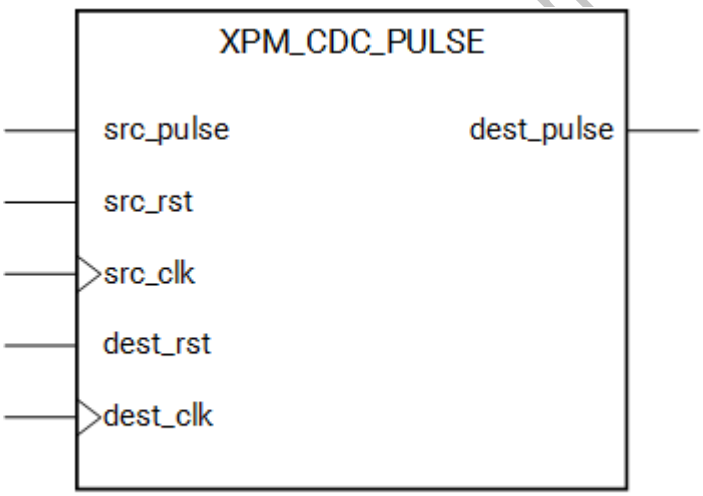
X15897-031116

- **XPM_CDC_ASYNC_RST**: esta macro sincroniza un reset asíncrono.



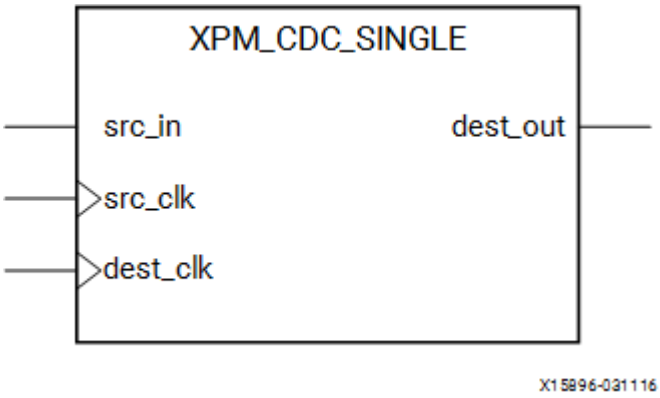
X15902-031116

- **XPM_CDC_PULSE**: esta macro sincroniza una señal de un sistema de reloj a otro.

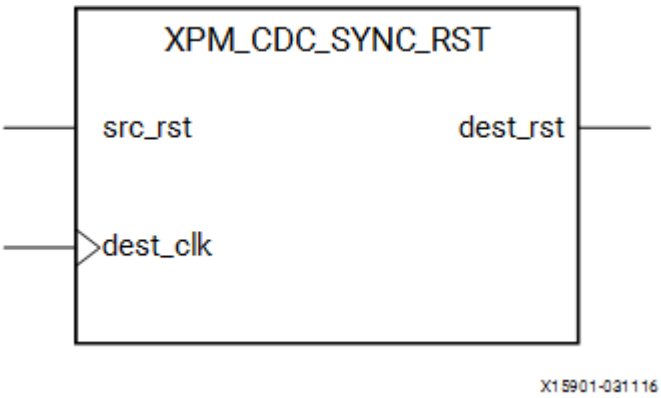


X15900-031116

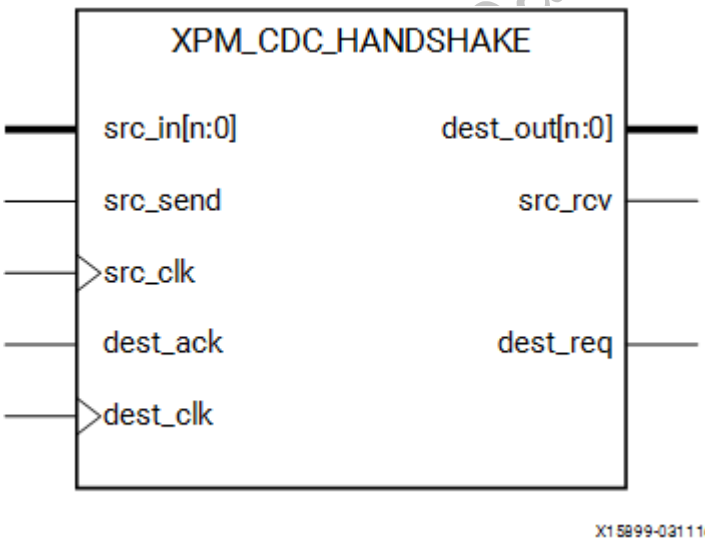
- **XPM_CDC_SINGLE**: esta macro sincroniza un dato de un bit.



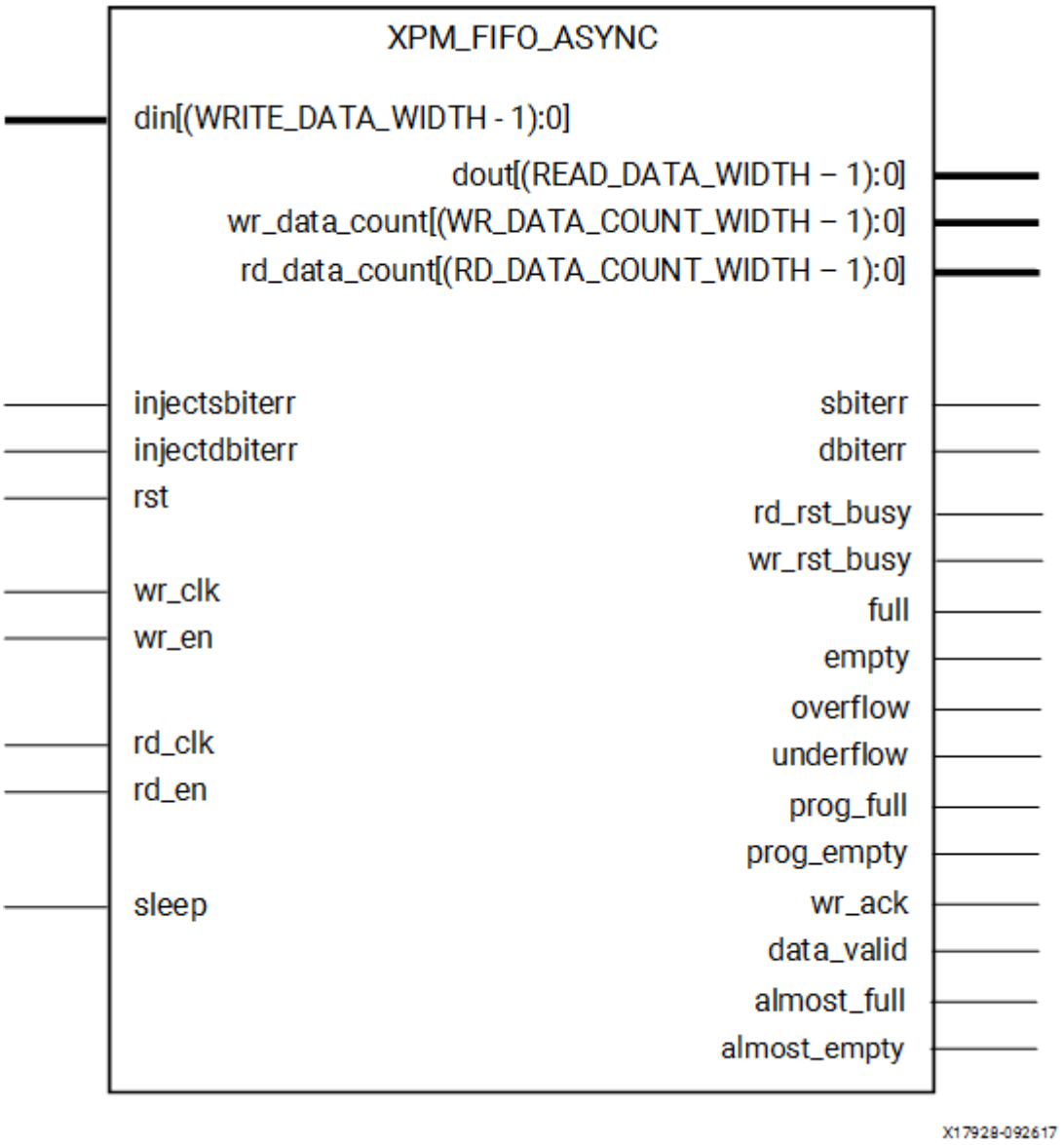
- **XPM_CDC_SYNC_RST**: esta macro sincroniza un reset síncrono que viene de otro dominio de tiempo.



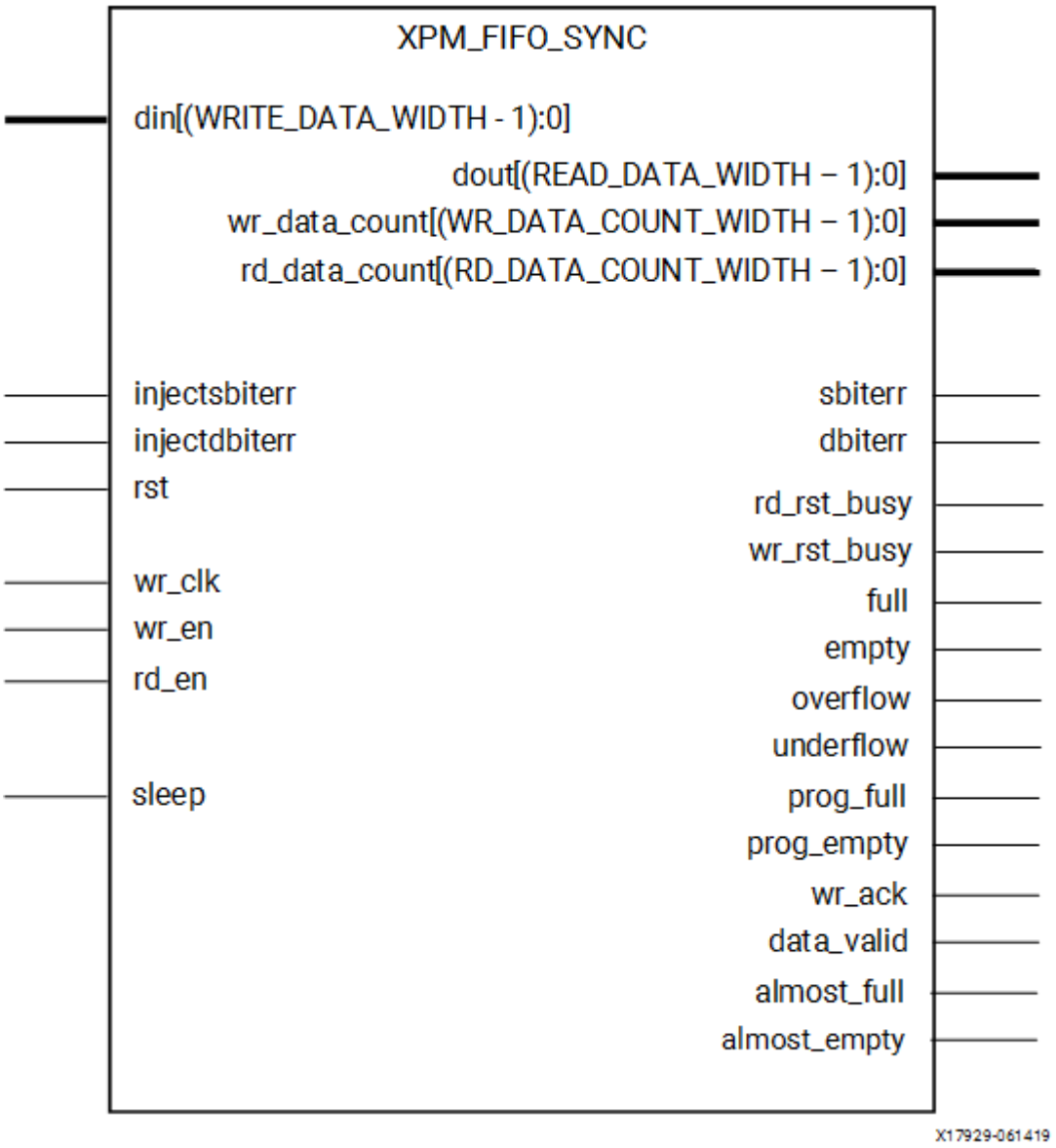
- **XPM_CDC_HANDSHAKE**: esta señal sincroniza un bus con un sistema de handshake



- **XPM_FIFO_ASYNC**: esta macro genera un FIFO asíncrono.



- **XPM_FIFO_SYNC**: esta macro genera un FIFO síncrono.


XPM_FIFO_SYNC
din[(WRITE_DATA_WIDTH - 1):0]
dout[(READ_DATA_WIDTH – 1):0]
wr_data_count[(WR_DATA_COUNT_WIDTH – 1):0]
rd_data_count[(RD_DATA_COUNT_WIDTH – 1):0]
injectsbiterr
injectdbiterr
rst
wr_clk
wr_en
rd_en
sleep
sbiterr
dbiterr
rd_rst_busy
wr_rst_busy
full
empty
overflow
underflow
prog_full
prog_empty
wr_ack
data_valid
almost_full
almost_empty
X17929-061419