Cómo compartir un proyecto firmware sin compartir el código fuente. Parte 1

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/05/24/como-compartir-un-proyecto-firmware-sin-compartir-el-codigo-fuente/

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

GitHub: https://github.com/DRubioG

Fecha última modificación: 22/02/2025

Para compartir un proyecto firmware sin compartir el código fuente tienes dos opciones:

- **Cifrar el código**, para cifrar el código fuente en Vivado tienes el comando «encrypt» que cifrar el código con una clave seleccionada por el usuario. Pero tiene un inconveniente y que para poder usar ese comando en Vivado se requiere de una licencia especial que Xilinx sólo entrega a empresas, si eres usuario no puedes habilitar este comando. Para más información consulta el foro de Xilinx y los vídeos que hay. Entonces, aparece una segunda opción.
- Esta segunda opción se basa en **exportar/importar la netlist** que genera Vivado al sintetizar. La lógica queda visible, pero el código queda ofuscado entre LUTs, además, de que se le puede decir al sintetizador que cambie la forma de sintetizar para dificultar se posible intento de descifrar. Esta segunda opción es la que voy a comentar a continuación.

Exportar una netlist

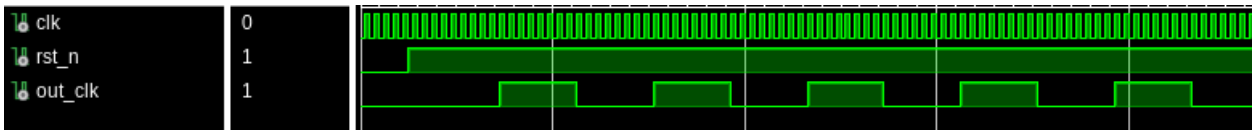
La mejor forma de explicar esto es con un ejemplo.

Imaginemos que tengo el funcionamiento de un prescaler de 16 al 50% que quiero compartir, pero no le quiero compartir el código a la persona a la que se lo paso.

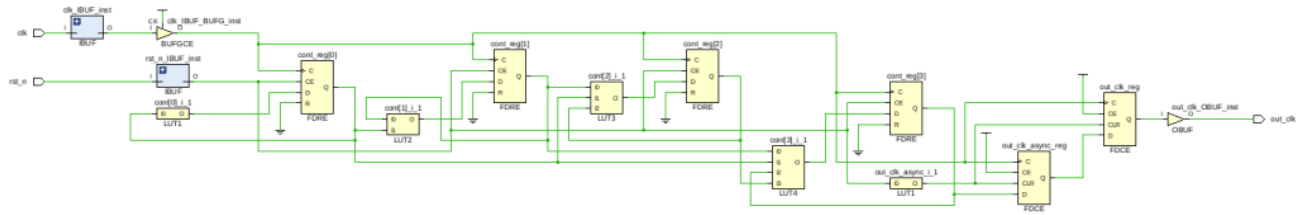
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity prescaler is
    Port (
        clk : in std_logic;
        rst_n : in std_logic;
        out_clk : out std_logic
    );
end prescaler;

architecture Behavioral of prescaler is

signal cont : integer range 0 to 15;
signal out_clk_async : std_logic;

begin

    process(clk, rst_n)
    begin
        if rst_n = '0' then
            out_clk_async <= '0';
        elsif rising_edge(clk) then
            if cont = 15 then
                cont <= 0;
            else
                cont <= cont +1;
            end if;

            if cont <= 7 then
                out_clk_async <= '0';
            else
                out_clk_async <= '1';
            end if;
        end if;
    end process;

    process(clk, rst_n)
    begin
        if rst_n = '0' then
            out_clk <= '0';
        elsif rising_edge(clk) then
            out_clk <= out_clk_async;
        end if;
    end process;

end Behavioral;
```

https://soceame.wordpress.com/2024/05/24/como-compartir-un-proyecto-firmware-sin-compartir-el-codigo-fuente/

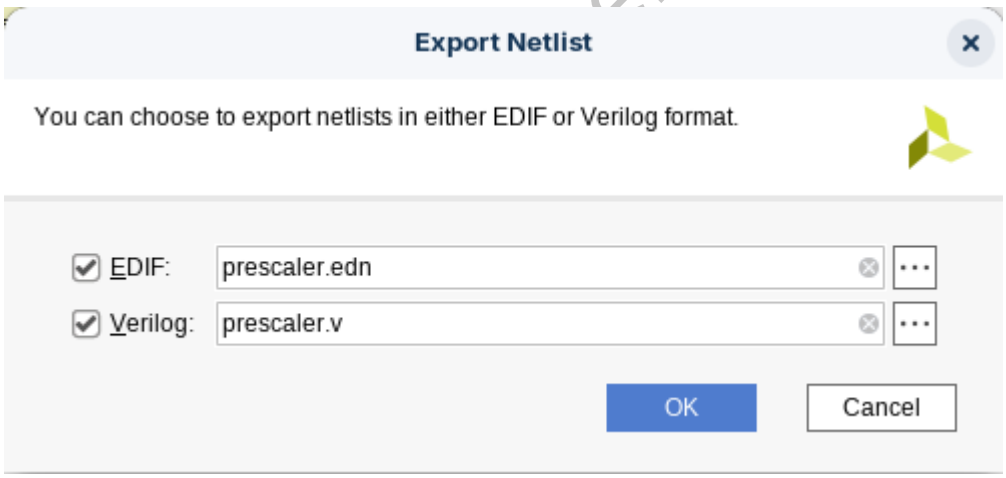
Este prescaler funciona de la siguiente manera.



Con este reloj ya generado, se sintetiza y se obtiene la siguiente síntesis.



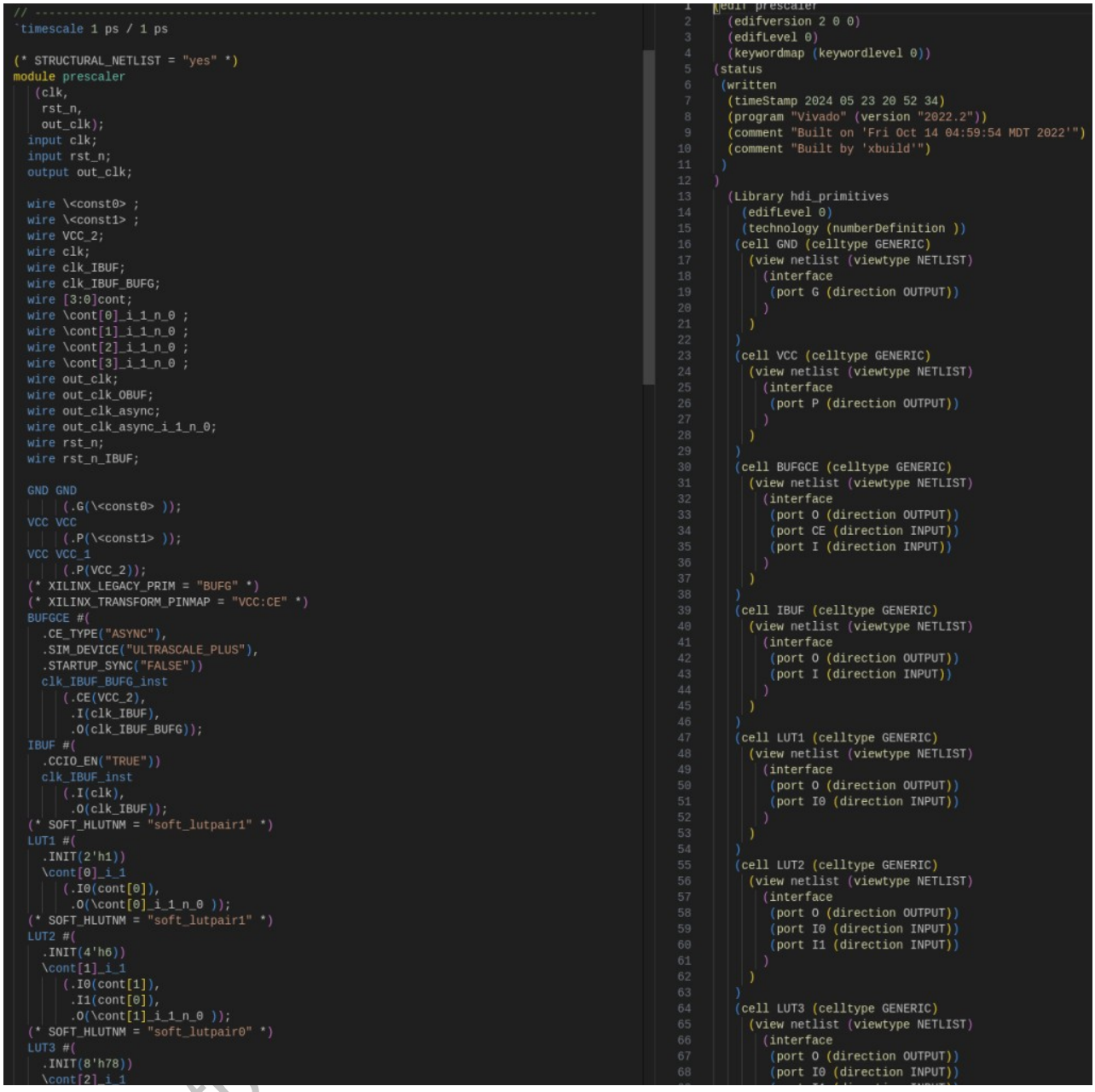
Con esta síntesis se decide exportarla, para ello primero tienes que estar en la opción «Open Synthesized Design» (vale con abrir el «Schematic» que genera la síntesis). Y una vez en esta opción, en la pestaña «File > Export» se habilita la opción «Export Netlist…». En esta opción se abre una pestaña en la que se puede generar o un fichero EDIF (.edn) o un fichero Verilog (.v) o ambos.



Estos ficheros son los que incluyen toda la síntesis generada.

Si se abren se puede ver que la lógica está ofuscada a la vista, aunque la opción de Verilog deja al descubierto los puestos de entrada y las señales utilizadas.

https://soceame.wordpress.com/2024/05/24/como-compartir-un-proyecto-firmware-sin-compartir-el-codigo-fuente/

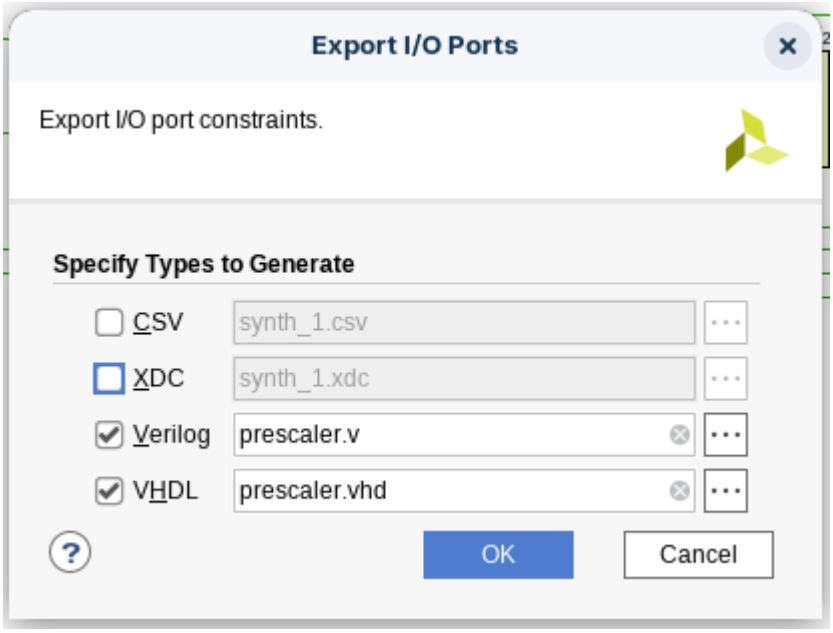


```
// ------------------------------------------------------------------------------
`timescale 1 ps / 1 ps

(* STRUCTURAL_NETLIST = "yes" *)
module prescaler
   (clk,
    rst_n,
    out_clk);
  input clk;
  input rst_n;
  output out_clk;

  wire \<const0> ;
  wire \<const1> ;
  wire VCC_2;
  wire clk;
  wire clk_IBUF;
  wire clk_IBUF_BUFG;
  wire [3:0]cont;
  wire \cont[0]_i_1_n_0 ;
  wire \cont[1]_i_1_n_0 ;
  wire \cont[2]_i_1_n_0 ;
  wire \cont[3]_i_1_n_0 ;
  wire out_clk;
  wire out_clk_OBUF;
  wire out_clk_async;
  wire out_clk_async_i_1_n_0;
  wire rst_n;
  wire rst_n_IBUF;

  GND GND
       (.G(\<const0> ));
  VCC VCC
       (.P(\<const1> ));
  VCC VCC_1
       (.P(VCC_2));
  (* XILINX_LEGACY_PRIM = "BUFG" *)
  (* XILINX_TRANSFORM_PINMAP = "VCC:CE" *)
  BUFGCE #(
    .CE_TYPE("ASYNC"),
    .SIM_DEVICE("ULTRASCALE_PLUS"),
    .STARTUP_SYNC("FALSE"))
    clk_IBUF_BUFG_inst
       (.CE(VCC_2),
        .I(clk_IBUF),
        .O(clk_IBUF_BUFG));
  IBUF #(
    .CCIO_EN("TRUE"))
    clk_IBUF_inst
       (.I(clk),
        .O(clk_IBUF));
  (* SOFT_HLUTNM = "soft_lutpair1" *)
  LUT1 #(
    .INIT(2'h1))
    \cont[0]_i_1
       (.I0(cont[0]),
        .O(\cont[0]_i_1_n_0 ));
  (* SOFT_HLUTNM = "soft_lutpair1" *)
  LUT2 #(
    .INIT(4'h6))
    \cont[1]_i_1
       (.I0(cont[1]),
        .I1(cont[0]),
        .O(\cont[1]_i_1_n_0 ));
  (* SOFT_HLUTNM = "soft_lutpair0" *)
  LUT3 #(
    .INIT(8'h78))
    \cont[2]_i_1
```

```
(edif prescaler
 (edifversion 2 0 0)
 (edifLevel 0)
 (keywordmap (keywordlevel 0))
(status
 (written
  (timeStamp 2024 05 23 20 52 34)
  (program "Vivado" (version "2022.2"))
  (comment "Built on 'Fri Oct 14 04:59:54 MDT 2022'")
  (comment "Built by 'xbuild'")
 )
)
 (Library hdi_primitives
  (edifLevel 0)
  (technology (numberDefinition ))
 (cell GND (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port G (direction OUTPUT))
   )
  )
 )
 (cell VCC (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port P (direction OUTPUT))
   )
  )
 )
 (cell BUFGCE (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port O (direction OUTPUT))
    (port CE (direction INPUT))
    (port I (direction INPUT))
   )
  )
 )
 (cell IBUF (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port O (direction OUTPUT))
    (port I (direction INPUT))
   )
  )
 )
 (cell LUT1 (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port O (direction OUTPUT))
    (port I0 (direction INPUT))
   )
  )
 )
 (cell LUT2 (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port O (direction OUTPUT))
    (port I0 (direction INPUT))
    (port I1 (direction INPUT))
   )
  )
 )
 (cell LUT3 (celltype GENERIC)
  (view netlist (viewtype NETLIST)
   (interface
    (port O (direction OUTPUT))
    (port I0 (direction INPUT))
```

Nota: si estás trabajando en un proyecto en VHDL, no te preocupes por exportar un fichero Verilog, ya que Vivado permite hacer de un fichero Verilog un componente para un modulo escrito en VHDL.

[VHDL] Bien entonces ahora se tienen dos ficheros, pero si solo se quisiera que hubiese un fichero ofuscado con el código (el .edn) y un fichero con SOLO los puertos de salida, Vivado permite exportar ese fichero. Para ello al igual que se ha exportado la netlist, también se pueden exportar los puertos en la opción «Export I/O Ports…». En la pestaña que se abre se puede elegir si quieres los puertos en VHDL *(la opción de Verilog aunque aparece, no funciona para esta exportación)*.



El código VHDL generado con los puertos es el siguiente:

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity prescaler is
  Port (
    clk : in STD_LOGIC;
    out_clk : out STD_LOGIC;
    rst_n : in STD_LOGIC
  );

end prescaler;

architecture Behavioral of prescaler is
begin

end Behavioral;
```

Una vez tenemos claro la exportación de una netlist pasamos a importar esta netlist.

Importar una netlist

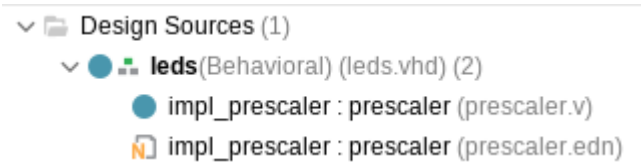
Continuando con el ejemplo anterior, ahora soy yo el que recibe el código ofuscado (.edn o .v) de un prescaler de 16 al 50%, y quiero incluirlo en mi código que hace parpadear unos leds.

Lo primero una vez he creado el proyecto es importar un fichero que tenga la netlist. Para ello se importa igual que si se tratara de un fichero .vhd o .v al proyecto.

Una vez importado uno de los ficheros, el que queramos, necesitamos aprovecharnos de que hemos exportado o conocemos los puertos del bloque ofuscado. Y con ello tratamos este bloque como si se tratase de un fichero más. Entonces, en nuestra aplicación de los leds declaramos un component con el prescaler y lo implementamos en nuestro código.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity leds is
    Port (
        clk : in std_logic;
        rst_n : in std_logic;
        leds : out std_logic_vector(3 downto 0)
    );
end leds;

architecture Behavioral of leds is

component prescaler is
    Port (
        clk : in std_logic;
        rst_n : in std_logic;
        out_clk : out std_logic
    );
end component;

signal prescaler_clk : std_logic;

begin

impl_prescaler : prescaler
    Port map (
        clk => clk,
        rst_n => rst_n,
        out_clk => prescaler_clk
    );

    process(clk, rst_n)
    begin
        if rst_n = '0' then
            leds <= (others=>'0');
        elsif rising_edge(clk) then
            if prescaler_clk = '0' then
                leds <= (others=>'0');
            elsif prescaler_clk = '1' then
                leds <= (others=>'1');
            end if;
        end if;
    end process;


end Behavioral;
```

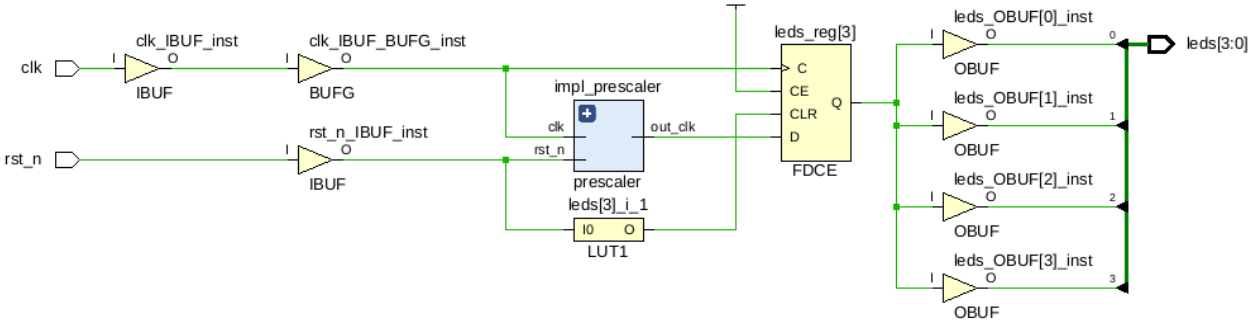
Esto provoca que el «Design Sources» establezca el orden de nuestros ficheros, fijando cuál es el top, y cuál es el bottom.



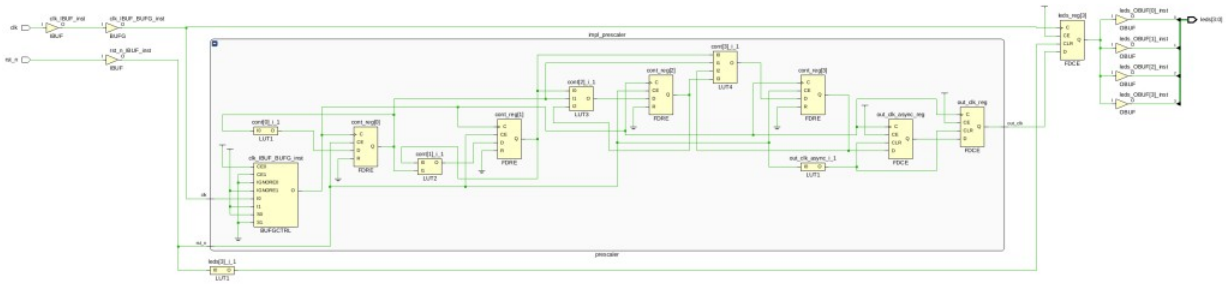
(Esto es un ejemplo demostrativo de que Vivado puede poner un fichero Verilog como bottom de un fichero VHDL. Solo se necesitaría uno de los dos [recomiendo el .edn con los puertos exportados en VHDL como en el paso anterior])

https://soceame.wordpress.com/2024/05/24/como-compartir-un-proyecto-firmware-sin-compartir-el-codigo-fuente/

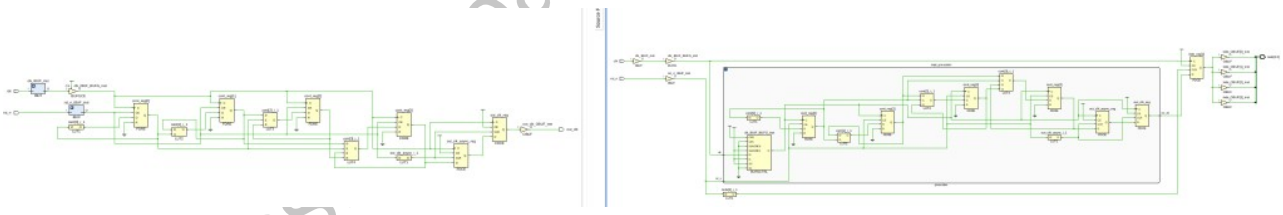
Y si ahora sintetizamos el código que hemos generado, nos encontramos que Vivado crea un bloque con el código sintetizado previamente con el prescaler.



Si lo expandimos nos queda de la siguiente forma



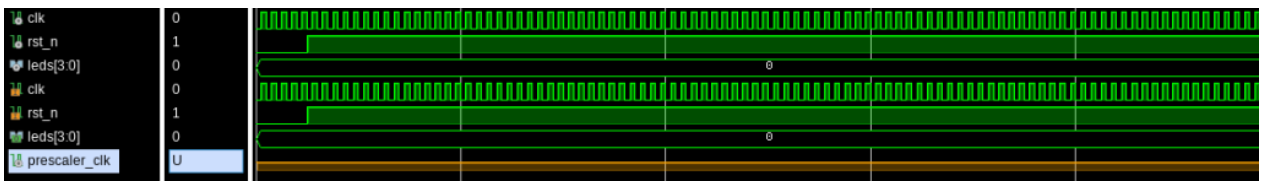
Y si lo comparamos con la síntesis del bloque prescaler (prescaler: izda, completo: dcha) podemos ver que son iguales, con la misma lógica de síntesis.



Y con esto doy por terminada la explicación de como importar una netlist. Y ahora nos surge una duda, una vez se ha importado una netlist, …

¿Se puede simular?

La respuesta es un rotundo **NO**. No se puede simular normalmente, no se puede hacer una simulación post-síntesis, ni tampoco una post-implementación, solo generar el bitstream.



https://soceame.wordpress.com/2024/05/24/como-compartir-un-proyecto-firmware-sin-compartir-el-codigo-fuente/

Parte 2:

https://soceame.wordpress.com/2024/05/26/como-compartir-un-proyecto-firmware-sin-compartir-el-codigo-fuente-parte-2/