Algoritmo GCD

El máximo común divisor (MCD ó GCD en inglés) de dos o más números enteros, es el mayor número entero que los divide sin dejar residuo alguno. Por ejemplo:

$$MCD(15,70)$$
$$Divisores\ de\ 15 \Rightarrow 1,3,5,15$$
$$Divisores\ de\ 70 \Rightarrow 1,2,5,7,10,14,35,70$$

Del ejemplo, se deduce fácilmente que el mayor número que divide a ambos enteros es el 5. Un método más eficiente es el algoritmo de *Euclides*, que utiliza el algoritmo de la división junto con el hecho de que el MCD de dos números también divide su diferencia. *Por ejemplo, para calcular* ***gcd(15,70)****, dividimos 70 entre 15, para obtener un cociente de 4 y un residuo de 10 (70 – 15 x 4). Luego dividimos 15 entre 10, para obtener un cociente de 1 y un residuo de 5 (15 – 10 x 1). Luego dividimos 10 entre 5, para obtener un cociente de 2 y un residuo de 0 (10 – 5 x 2), lo cuál significa que* ***5 es el gcd de 15 y 70****.*

Formalmente el algoritmo se describe como:

$$gcd(a,0) = a$$
$$gcd(a,b) = gcd(b, a \bmod b)$$
$$donde:$$
$$a \bmod b = a - b\left\lfloor \frac{a}{b} \right\rfloor$$

Si los argumentos de ambos términos son mayores que cero, el algoritmo se puede escribir en términos más elementales como:

$$gcd(a,a) = a$$
$$gcd(a,b) = gcd(a-b,b), si\ a > b$$
$$gcd(a,b) = gcd(a,b-a)\ si\ b > a$$

De acuerdo con lo anterior, el algoritmo puede ser escrito en C como:

```
input : int x, y;
output: int gcd;
while (x /= y) {
        if (x < y)
                y = y - x;
        else
                x = x - y;
}
gcd = x;
```

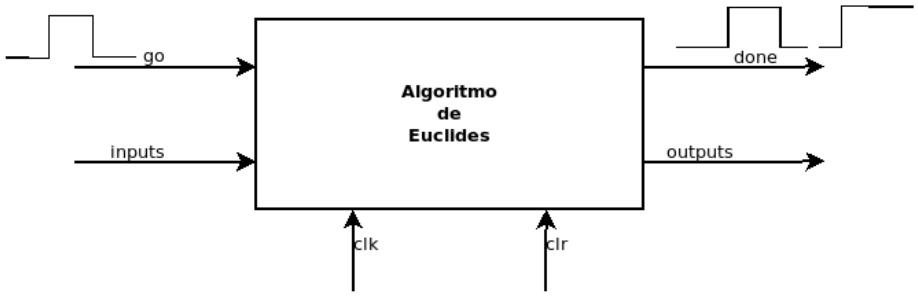
El problema de este algoritmo es que en VHDL un ciclo while puede ser escrito y simulado, pero no sintetizado. Una primera aproximación del algoritmo, en VHDL se puede escribir como:

```
Algoritmo_Euclides: process (clk, clr)
begin
        if clr = '1' then
                --limpiar registros
        elsif clk'event and clk = '1' then
                if x = y then
                        gcd  <= x;
                elsif x < y then
                        y <= y - x;
                else
                        x <= x - y;
                end if;
        end if;
end process Algoritmo_Euclides:
```

Aquí estamos usando la sentencia "**if** x = y" en lugar de while. En general, podemos implementar un ciclo while con sentencias if, pero necesitamos algún tipo de goto al final del ciclo para regresar al inicio de éste.

Recordemos que cuando usamos la sentencia "clk'event **and** clk = '1'" todas las señales o variables que siguen, se registran (por ejemplo: "gcd <= x;", "y <= y – x;", "x <= x – y;") en cada pulso de subida del reloj. Esto es, la sentencia "**if** x = y" se probará continuamente. Entonces, ¿como podemos detener la prueba de la sentencia para terminar el ciclo?

Podemos usar una señal de "***go***" para comenzar un algoritmo y otra "***done***" para indicar que se ha completado (mostrado en la siguiente figura).



Dependiendo de la aplicación, la señal "***done***" puede ser un pulso (que puede ser usado como señal de entrada "***go***" para otro algoritmo) o sólo el cambio de nivel.

El siguiente código muestra cómo puede ser generado el pulso "***done***" con un algoritmo. Es importante notar que la variable "***calc***" está en "***1***" durante la ejecución del algoritmo.

```
algoritmo_genera_pulso: process (clk, clr)
variable calc, donev : std_logic;
begin
        if clr = '1' then
                --limpiar registros
                donev := '0';
                calc := '0';
        elsif clk'event and clk = '1' then
                donev := '0';
                if go = '1" then
```

```
                --iniciliza señales y variables
                calc := '1';
        elsif calc = '1' then
                if --condición para "done"-- then
                        --asignar salidas
                        done := '1';
                        calc := '0';
                else
                        --sentencias del algoritmo
                end if;
    end if;
    done <= donev;
end process algoritmo_genera_pulso:
```

Finalmente, en el siguiente código se muestra la combinación de ambos algoritmos; el que genera el pulso y el que determina el máximo común divisor (GCD).

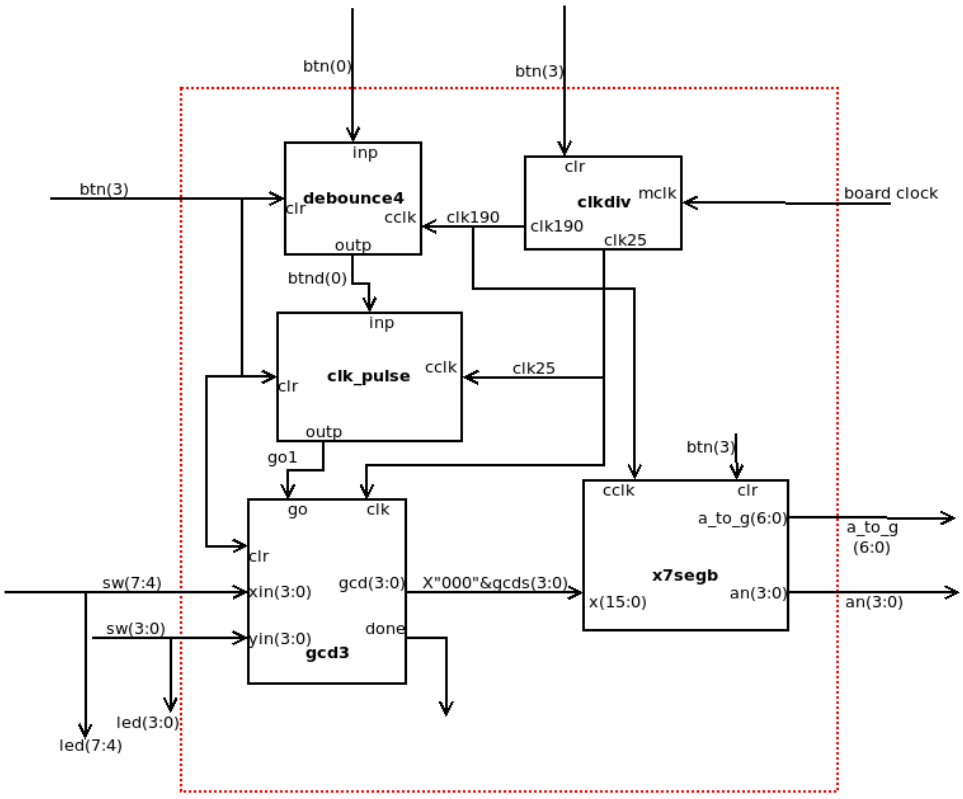
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.STD_LOGIC_unsigned.all;

entity gcd3 is
   Port ( clk : in  STD_LOGIC;
          clr : in  STD_LOGIC;
          go : in  STD_LOGIC;
          xin : in  STD_LOGIC_VECTOR (3 downto 0);
          yin : in  STD_LOGIC_VECTOR (3 downto 0);
          done : out  STD_LOGIC;
          gcd : out  STD_LOGIC_VECTOR (3 downto 0));
end gcd3;

architecture gcd3 of gcd3 is
    signal x, y: std_logic_vector (3 downto 0);
begin
    algoritmo_gcd: process (clk, clr)
    variable calc, donev : std_logic;
    begin
        if (clr = '1') then
            x <= (others => '0');
            y <= (others => '0');
            gcd <= (others => '0');
            donev := '0';
            calc := '0';
        elsif (rising_edge(clk)) then
            donev := '0';
            if (go = '1') then
                x <= xin;
                y <= yin;
                calc := '1';
            elsif (calc = '1') then
                if (x = y) then
                    gcd <= x;
                    donev := '1';
                    calc := '0';
                elsif (x < y) then
                    y <= y - x;
                else
                    x <= x - y;
                end if;
            end if;
        end if;
```

```
                    done <= donev;
                end process algoritmo_gcd;
end gcd3;
```

Para probar el algoritmo, podemos integrarlo al diseño “Top-level” mostrado en la siguiente figura.



- Notar que la señal ***go*** de entrada al bloque ***gcd3*** debe estar en alto por sólo un ciclo del reloj de ***25 Mhz***.
- Cuando la señal ***go*** es 1, entonces en el siguiente pulso de subida del reloj, los valores de ***x*** y ***y*** se inicializan a ***xin*** y ***yin*** y ***calc*** se pone en 1. Esto es, en el siguiente pulso de reloj, la señal ***go*** va a 0 cuando ***x*** es igual a ***y*** y el registro ***gcd*** toma el último valor de ***x***. En este punto la variable ***donev*** se pone en 1, pero se colocará en 0 en el siguiente pulso de reloj debido a que ***calc*** es ahora 0.