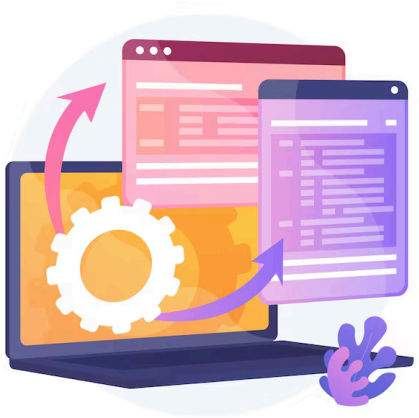
Práctica 5



Diseño Automático de Sistemas Fiables

UNIVERSIDAD NEBRIJA

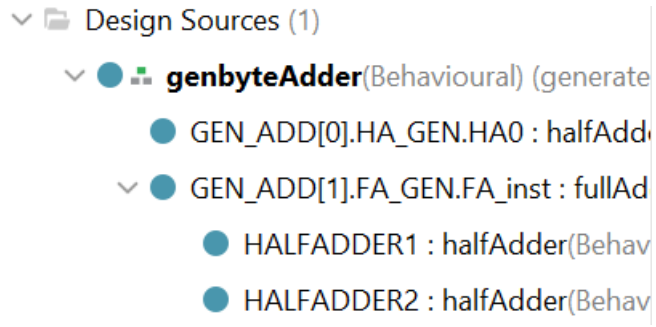
GUILLERMO LÓPEZ LÓPEZ

NICOLAS HURTADO CARDONA

Objetivo

Diseñar un sumador de 8 bits en VHDL usando componentes halfAdder y fullAdder, generando el circuito completo con generate y verificar su funcionamiento con simulaciones

Implementación del diseño



Half Adder

Suma un bit sin acarreo de entrada: S = A xor B, C = A and B

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

entity halfAdder is
    port (  A       : in    std_logic;
            B       : in    std_logic;
            S       : out   std_logic;
            C_out   : out   std_logic
    );
end halfAdder;

architecture Behavioural of halfAdder is
begin
    S <= A xor B;
    C_out <= A and B;
end Behavioural;
```

Full Adder

Usa dos half Adder y una OR para incluir el acarreo de entrada.

```
                S       : out   std_logic;
                C_out   : out   std_logic
            );
    end component;

begin
    HALFADDER1 : halfAdder port map(
            A       =>  A,
            B       =>  B,
            S       =>  S_halfAdder1,
            C_out   =>  C_halfAdder1
    );

    HALFADDER2 : halfAdder port map(
            A       =>  C_in,
            B       =>  S_halfAdder1,
            S       =>  S,
            C_out   =>  C_halfAdder2
    );

    C_out <= C_halfAdder1 or C_halfAdder2;

end Behavioural;
```

GenbyteAdder

Sumador de 8 bits implementado de forma modular

Primero instancia un half Adder para el bit 0 y para el resto (bits 1 al 7) un fullAdder

La salida **C_out** es el último acarreo generado (C(7)), indicando si ha habido un desbordamiento

```
begin
    GEN_ADD: for I in 0 to 7 generate

    HA_GEN: if I = 0 generate
        HA0: halfAdder
        port map (
            A     => A(0),
            B     => B(0),
            S     => S(0),
            C_out => C(0)
        );
    end generate;

    FA_GEN: if I > 0 generate
        FA_inst: fullAdder
        port map (
            A     => A(I),
            B     => B(I),
            C_in  => C(I-1),
            S     => S(I),
            C_out => C(I)
        );
    end generate;

end generate GEN_ADD;
```

Simulación

El test bench adjuntado tb_genbyteadder valida el funcionamiento del sumador genbyteAdder de una manera escalable.

Se ha tomado como referencia el tb_byteAdder en el uso de check(...), que recibe A y B, junto con el valor esperado de la suma y el acarreo.

Además, se utiliza una función auxiliar to_int para convertir los vectores std_logic_vector a enteros.

```
function to_int(v : std_logic_vector) return integer is
    begin
        return to_integer(unsigned(v));
    end function;
```

En cada prueba, se comparan la salida del sumador (S) y el bit de acarreo (C_out) con los valores esperados. Si hay errores, se muestra por pantalla el fallo, mostrando los valores de entrada, el resultado obtenido y el esperado. Pero no detiene la ejecucion del código (se cambia los severity error por severity notes)

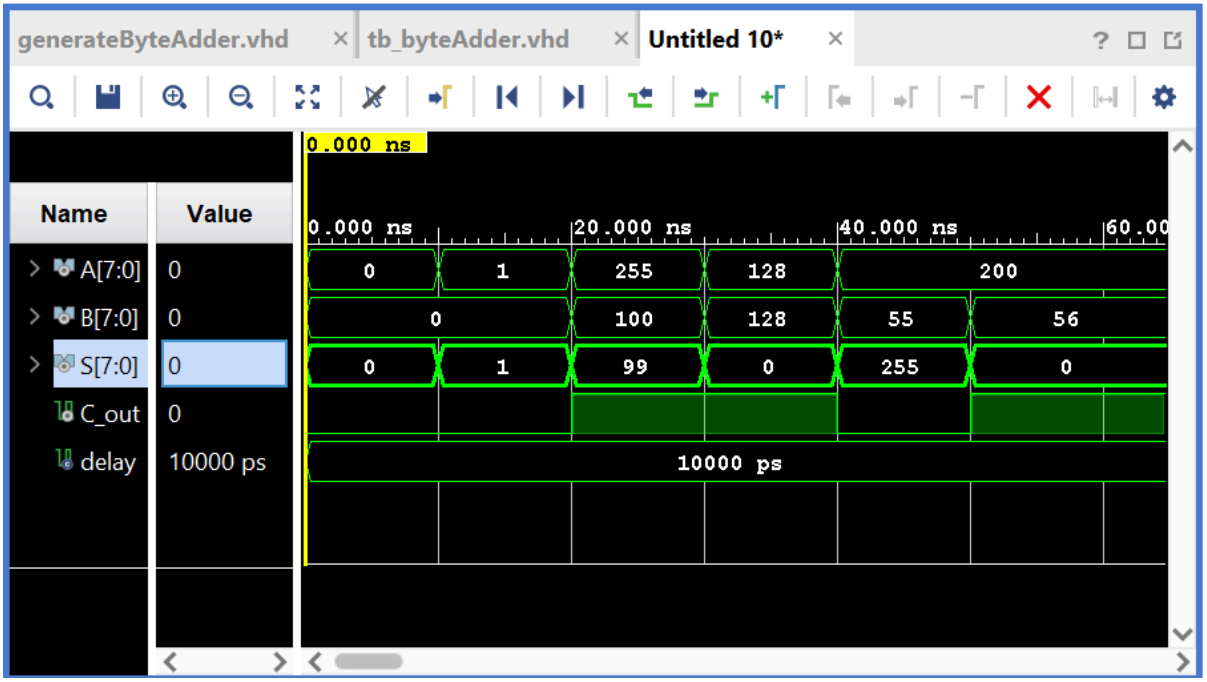
```
            wait for delay;

            assert to_int(S) = expected_sum
                report "Error en suma: A=" & integer'image(a_val) &
                       " B=" & integer'image(b_val) &
                       " -> Esperado=" & integer'image(expected_sum) &
                       " Obtenido=" & integer'image(to_int(S))
        severity note;

            assert C_out = expected_c
                report "Error en acarreo: A=" & integer'image(a_val) &
                       " B=" & integer'image(b_val) &
                       " -> Esperado=" & std_logic'image(expected_c) &
                       " Obtenido=" & std_logic'image(C_out)
        severity note;
```

Si todo es correcto, también se imprime una línea confirmando la validez de la operación.

```
# run 1000ns
Note: Correcto: A=0 B=0 -> Suma=0 Carry='0'
Time: 10 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
Note: Correcto: A=1 B=0 -> Suma=1 Carry='0'
Time: 20 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
Note: Correcto: A=255 B=100 -> Suma=99 Carry='1'
Time: 30 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
Note: Correcto: A=128 B=128 -> Suma=0 Carry='1'
Time: 40 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
Note: Correcto: A=200 B=55 -> Suma=255 Carry='0'
Time: 50 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
Note: Correcto: A=200 B=56 -> Suma=0 Carry='1'
Time: 60 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
Note: Testbench finalizado correctamente.
Time: 60 ns  Iteration: 0  Process: /tb_genbyteAdder/stimulus  File: C:/Users/Guill/Desktop/Universite/2425/fiables/pr5/byteAdder/b
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_genbyteAdder_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
```



Suma sin desbordamiento (por ejemplo, 1 + 0)

Suma con desbordamiento (por ejemplo, 255 + 100 → 99 con acarreo)

Casos límite (como 128 + 128 = 0 con acarreo)

Transición entre límite inferior y superior (200 + 55 sin acarreo frente a 200 + 56 con acarreo)