



Práctica 3

4x4-bit multiplicadores



Objetivo

- Implementar 2 multiplicadores 4x4-bit diferentes:
 - Usando el operador '*' de la librería `numeric_std`
 - Usando sumadores de 8 bits. (En el laboratorio 1 usamos sumadores de 4 bits. Hay que modificarlos)
- Estudiar los report de Vivado para encontrar:
 - Los elementos combinacionales utilizados
 - El retardo máximo de los caminos combinacionales


$$\begin{array}{cccccccc}
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & & 1 & 0 & 1 & 1 \\
 \hline
 & & & & & 1 & 1 & 0 & 1 \\
 & & & & 1 & 1 & 0 & 1 & \\
 & & 0 & 0 & 0 & 0 & & & \\
 & 1 & 1 & 0 & 1 & & & & \\
 \hline
 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1
 \end{array}$$



Sumador 4 bits

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

entity sumador is
  Port (
    A : IN    std_logic_vector(3 downto 0);
    B : IN    std_logic_vector(3 downto 0);
    C : OUT   std_logic_vector(3 downto 0)
  );
end sumador;

architecture Behavioral of sumador is

begin

  C <= A + B;

end Behavioral;
```

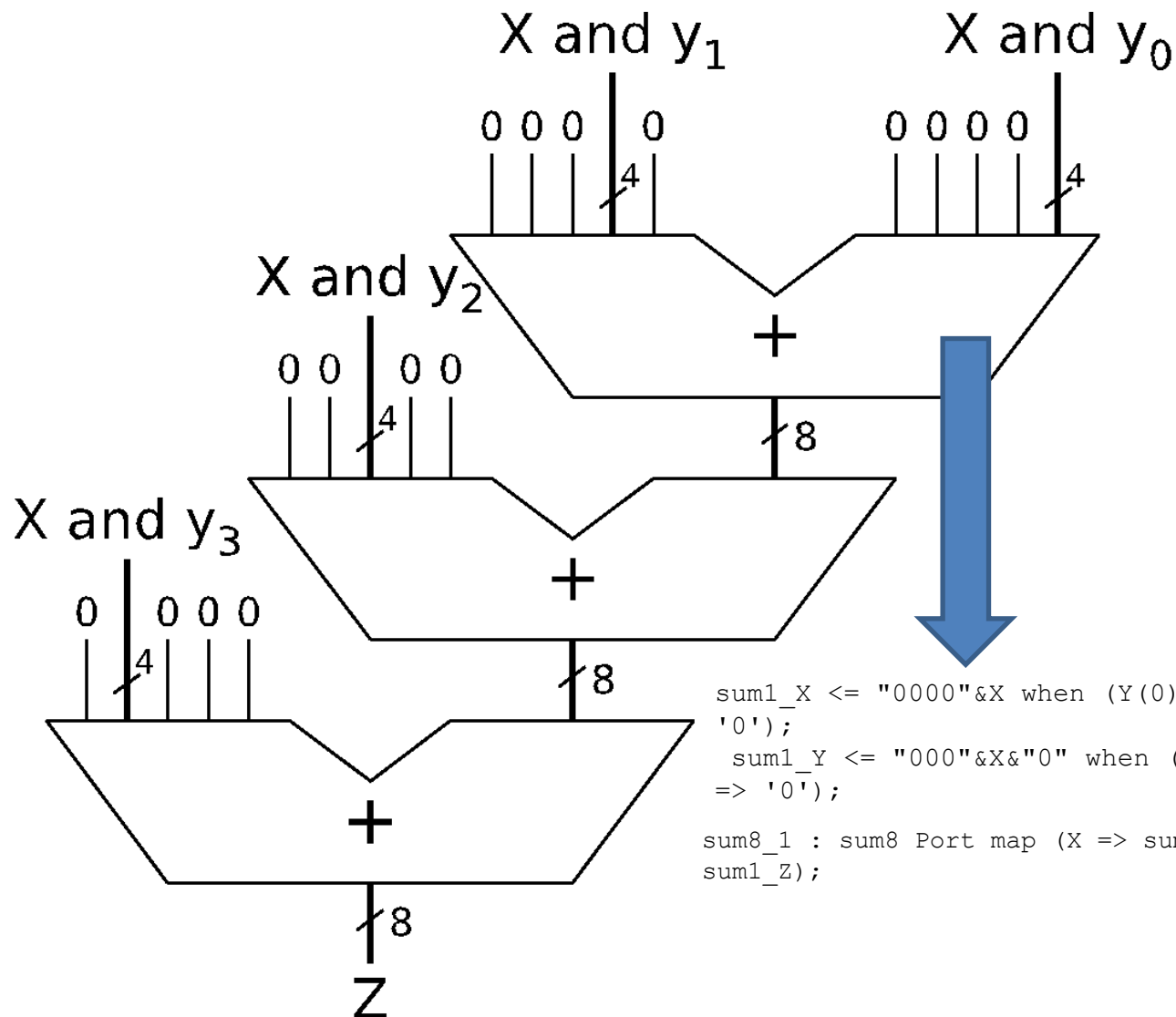
Hay que diseñar un
sumador de 8 bits



Multiplicador

```
entity mult8b is
  port(
    X : in  std_logic_vector(3 downto 0);
    Y : in  std_logic_vector(3 downto 0);
    Z : out std_logic_vector(7 downto 0)
  );
end mult8b;
```

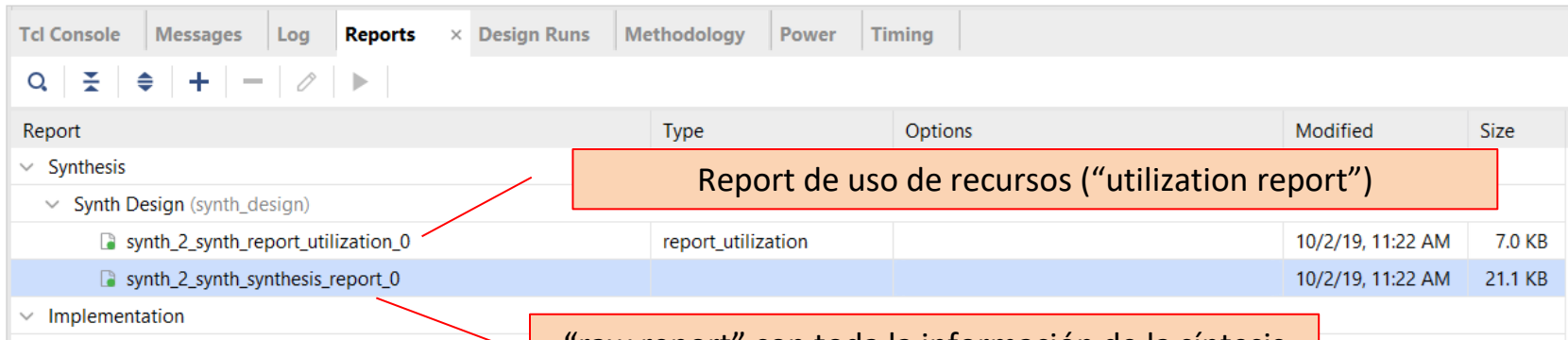
Multiplicador usando 8-bit adders



```
sum1_X <= "0000"&X when (Y(0) = '1') else (others => '0');
sum1_Y <= "000"&X&"0" when (Y(1) = '1') else (others => '0');
sum8_1 : sum8 Port map (X => sum1_X, Y => sum1_Y, Z => sum1_Z);
```

Synthesis reports

- Después de la síntesis, en la pestaña Reports (debajo) podemos ver dos reports de síntesis:



The screenshot shows the Reports tab with the following table:

Report	Type	Options	Modified	Size
▼ Synthesis				
▼ Synth Design (synth_design)				
synth_2_synth_report_utilization_0	report_utilization		10/2/19, 11:22 AM	7.0 KB
synth_2_synth_synthesis_report_0			10/2/19, 11:22 AM	21.1 KB
▼ Implementation				

Annotations in the image:

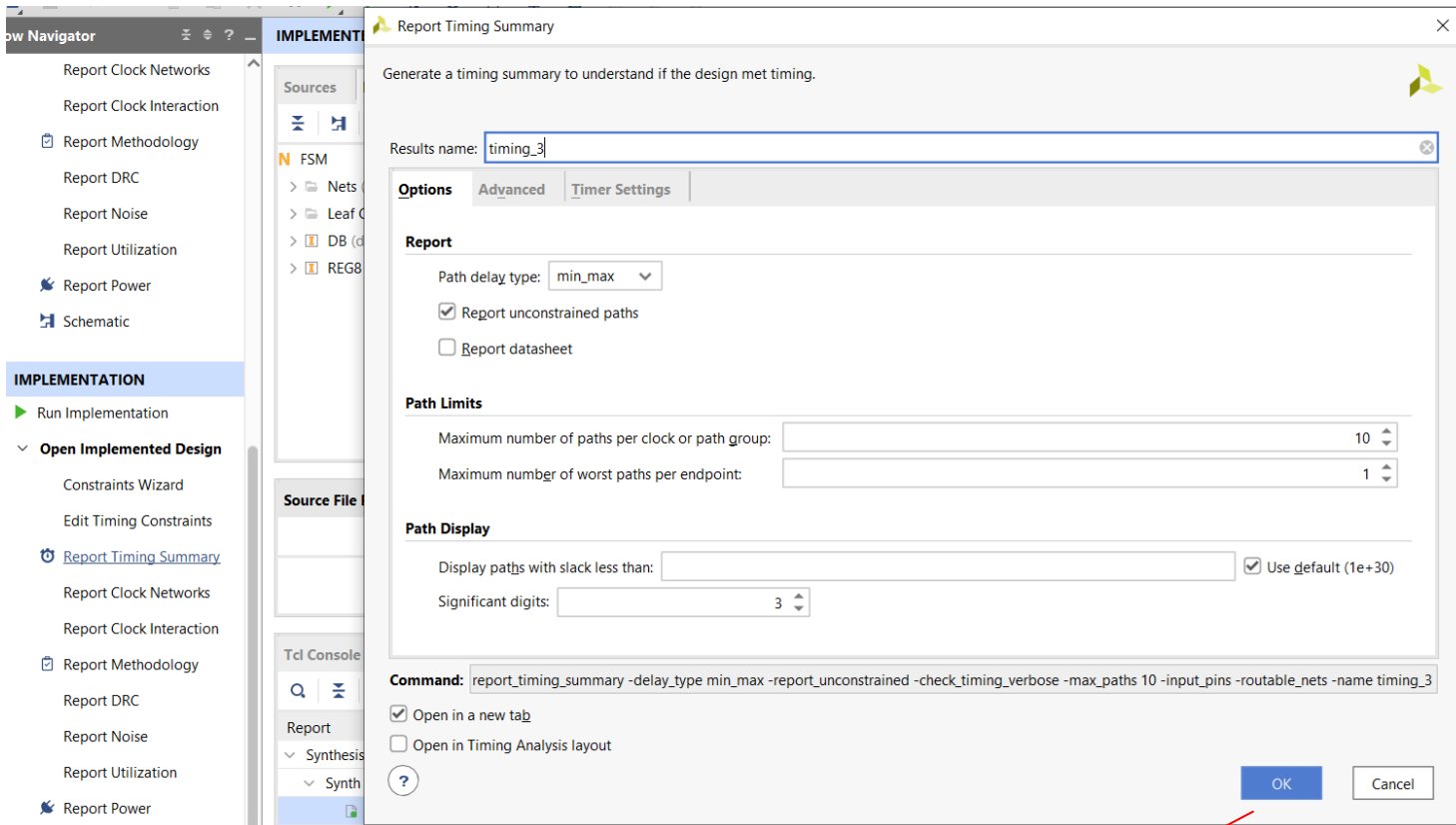
- A red box highlights the row for `synth_2_synth_report_utilization_0` with the text: "Report de uso de recursos ("utilization report")".
- A red box highlights the row for `synth_2_synth_synthesis_report_0` with the text: "raw report" con toda la información de la síntesis".

- Por ejemplo (utilization report, de la práctica 2):

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	46	0	20800	0.22
LUT as Logic	46	0	20800	0.22
LUT as Memory	0	0	9600	0.00
Slice Registers	35	0	41600	0.08
Register as Flip Flop	35	0	41600	0.08
Register as Latch	0	0	41600	0.00
F7 Muxes	0	0	16300	0.00
F8 Muxes	0	0	8150	0.00

Timing reports

Debajo de “Open Implemented Design”, tanto en SYNTHESIS como en IMPLEMENTATION, se puede ver el “Report Timing Summary” (después de la implementación es más preciso)



Pulsar OK para generar el report



Timing reports

- Estos reports pueden visualizarse clicking en la pestaña “Timing” (debajo del Vivado GUI):

Worst-case setup margin

Worst-case hold margin

Setup	Hold	Pulse Width
Worst Negative Slack (WNS): 6,501 ns	Worst Hold Slack (WHS): 0,217 ns	Worst Pulse Width Slack (WPWS): 4,500 ns
Total Negative Slack (TNS): 0,000 ns	Total Hold Slack (THS): 0,000 ns	Total Pulse Width Negative Slack (TPWS): 0,000 ns
Number of Failing Endpoints: 0	Number of Failing Endpoints: 0	Number of Failing Endpoints: 0
Total Number of Endpoints: 54	Total Number of Endpoints: 54	Total Number of Endpoints: 36

All user specified timing constraints are met.

- “All user specified timing constraints are met” ? Cómo sabe la herramienta cuál es el periodo de reloj? Hay que definirlo en el fichero .xdc :

```
create_clock -add -name clk -period 10.00 -waveform {0 5} [get_ports clk]
```

- Esta línea caracteriza la señal de reloj, que viene del oscilador en el pin W5 (en la placa Basys3).
 - Define un periodo de reloj (10 ns), que es 0 la mitad del ciclo (5ns) y 1 el resto.
 - No es posible realizar análisis temporal sin esta línea, ni tampoco hacer place&route de acuerdo a restricciones temporales.



Timing reports - Paths

- Se pueden analizar los caminos (Paths) mediante un STA.
- Paths desde un Flip-Flop (u otro elemento de memoria) a otro Flip-Flop (u otro elemento de memoria) se llaman “**Intra-Clock Paths**”, y sus retardos pueden chequearse.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	7.782	3	4	2	U_REG_S1_2/q_reg[3]/C	U_REG_S2_2/q_reg[6]/D	2.084	1.594	0.490	10.0	clk
Path 2	8.010	2	3	2	U_REG_S1_2/q_reg[3]/C	U_REG_S2_2/q_reg[5]/D	1.872	1.391	0.481	10.0	clk
Path 3	8.070	2	3	2	U_REG_S1_2/q_reg[3]/C	U_REG_S2_2/q_reg[4]/D	1.812	1.331	0.481	10.0	clk
Path 4	8.219	2	3	3	U_REG_S1_2/q_reg[2]/C	U_REG_S2_2/q_reg[3]/D	1.663	1.175	0.488	10.0	clk
Path 5	8.610	1	2	3	U_REG_S1_2/q_reg[2]/C	U_REG_S2_2/q_reg[2]/D	1.239	0.751	0.488	10.0	clk
Path 6	8.795	0	1	1	U_REG_S1_1/q_reg[3]/C	U_REG_S2_1/q_reg[3]/D	0.790	0.456	0.334	10.0	clk

- Paths desde una entrada primaria a un Flip-Flop (u otro elemento de memoria) y de un Flip-Flop (u otro elemento de memoria) a una salida primaria, pueden encontrarse en “**Unconstrained Paths**” ? “**NONE to sys_clk_pin**” and “**sys_clk_pin to NONE**”, respectivamente.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination
Path 1	∞	2	2	5	Y[1]	reg_salida_s...ut_reg[2]/D	2.418	1.618	0.800	∞	input port clock	sys_clk_p
Path 2	∞	2	2	10	X[1]	reg_salida_s...ut_reg[4]/D	2.411	1.611	0.800	∞	input port clock	sys_clk_p
Path 3	∞	2	2	5	Y[1]	reg_salida_s...ut_reg[1]/D	2.390	1.590	0.800	∞	input port clock	sys_clk_p
Path 4	∞	2	2	5	Y[1]	reg_salida_s...ut_reg[3]/D	2.390	1.590	0.800	∞	input port clock	sys_clk_p
Path 5	∞	2	2	5	Y[1]	reg_salida_s...ut_reg[4]/D	2.390	1.590	0.800	∞	input port clock	sys_clk_p
Path 6	∞	2	2	5	Y[1]	reg_salida_s...ut_reg[5]/D	2.390	1.590	0.800	∞	input port clock	sys_clk_p



Timing reports - Paths

- Sin embargo, la parte básica de esta práctica es totalmente combinacional, y estos caminos (paths) no aparecen (todavía).
- Caminos de una entrada primaria a una salida primaria se nombran como **“Unconstrained Paths”** ? **“NONE to NONE”**.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	∞	5	4	13	Y[1]	Z[6]	8.457	5.944	2.512	∞	input port clock		
Path 2	∞	5	4	13	Y[1]	Z[5]	8.347	5.835	2.512	∞	input port clock		
Path 3	∞	5	4	13	Y[1]	Z[7]	8.259	5.747	2.512	∞	input port clock		
Path 4	∞	5	4	13	Y[1]	Z[4]	8.153	5.641	2.512	∞	input port clock		
Path 5	∞	4	3	13	Y[1]	Z[3]	7.810	5.737	2.073	∞	input port clock		
Path 6	∞	4	3	13	X[1]	Z[2]	7.470	5.870	1.599	∞	input port clock		



Calificación

- El estudiante debe acudir al laboratorio con la práctica estudiada y simulada desde casa. Puede utilizarse el test_bench proporcionado en el CV.
- El estudiante debe mostrar el multiplicador basado en sumadores funcionando y debe comprender la implementación y funcionalidad:
 - Si funciona en la FPGA 0.1 pts
 - Si solo funciona la simulación, 0.05 pts
 - Hay que enseñar los “*temporal and synthesis reports*”
- Parte avanzada 0.15 pts
- La práctica 3 no se recupera.



Testbench

```
-- Stimulus process
p_stim : process
    variable v_i : natural := 0;
    variable v_j : natural := 0;
begin
    i_loop : for v_i in 0 to 15 loop
        j_loop : for v_j in 0 to 15 loop
            X      <= std_logic_vector(to_unsigned(v_i, 4));
            Y      <= std_logic_vector(to_unsigned(v_j, 4));
            Z_xpct <= std_logic_vector(to_unsigned(v_i * v_j, 8));
            wait for 5 ns;
            assert Z = Z_xpct
                report "Error multiplying, "&integer'image(v_i) & " * " &
                    &integer'image(v_j) & " = "&integer'image(v_i*v_j) &
                    " not "&integer'image(to_integer(unsigned(Z)))
                severity error;
            wait for 5 ns;
        end loop j_loop;
    end loop i_loop;
    wait;
end process p_stim;
```