Preguntas del laboratorio de fpga

1. Los elementos lógicos (LEs) en la FPGA Cyclone III están compuestos por una LUT de 4 entradas que implementa funciones lógicas de hasta cuatro variables, un flip-flop para almacenamiento de estados, un selector de modo que permite operar en modo lógico o aritmético, y una cadena de acarreo (carry chain) que facilita operaciones rápidas como sumas. También cuentan con señales de control como *clock*, *enable* y *set/reset* para gestionar el comportamiento del LE. Los LEs se agrupan en Bloques Lógicos (LABs, Logical Array Blocks), que tienen una estructura de 10 LEs conectados entre sí y con recursos compartidos, lo que optimiza la interconexión y distribución de señales dentro de la FPGA.
2. El Nios® II es un procesador softcore de 32 bits diseñado por Intel (anteriormente Altera) específicamente para FPGAs de la serie Cyclone, Stratix y Arria. Este procesador se implementa dentro de la FPGA como un bloque de lógica programable y es completamente personalizable, lo que permite adaptarlo a una variedad de aplicaciones embebidas según los requisitos del usuario. El Nios II es flexible en cuanto a tamaño, potencia y rendimiento, ya que puede configurarse en varias versiones: desde una versión de bajo costo y consumo hasta una versión de alto rendimiento que soporta operaciones en punto flotante y multitarea.
3. La diferencia entre *IP cores* y bloques embebidos en una FPGA es que los *IP cores* son bloques de diseño lógico predefinidos que implementan funciones específicas (como procesadores o interfaces de comunicación) y se configuran en la lógica programable de la FPGA, aunque pueden aprovechar los bloques embebidos para optimizar su eficiencia. Los bloques embebidos, como los multiplicadores, bloques de memoria y DSPs, son componentes de hardware específicos ya integrados físicamente en la FPGA. Mientras que los *IP cores* son configurables y ocupan recursos lógicos, los bloques embebidos funcionan como unidades especializadas que no consumen lógica programable y brindan alto rendimiento en operaciones específicas.
4. El dispositivo FPGA Cyclone III utiliza celdas de programación basadas en tecnología SRAM (Static Random Access Memory), lo cual le permite ser completamente reconfigurable. Esta tecnología permite programar y reprogramar el dispositivo varias veces sin desgaste físico, lo que es ideal para aplicaciones que requieren actualizaciones o ajustes frecuentes del diseño. Sin embargo, debido a que las celdas SRAM son volátiles, el contenido de la programación se pierde al apagar el dispositivo, por lo que es necesario cargar la configuración en cada inicio mediante una memoria externa, como una memoria flash o EEPROM, que almacene el bitstream de configuración.
5. library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity ff\_jk is

Port ( clk : in STD\_LOGIC; j : in STD\_LOGIC; k : in STD\_LOGIC; q : out STD\_LOGIC );

end ff\_jk;

architecture Behavioral of ff\_jk is

signal q\_i : STD\_LOGIC := '0';

begin

process (clk)

begin

if rising\_edge(clk) then

if j = '0' and k = '1' then

q\_i <= '0';

elsif j = '1' and k = '0' then

q\_i <= '1';

elsif j = '1' and k = '1' then

q\_i <= not q\_i;

end if;

end if;

end process;

q <= q\_i; end Behavioral;

6) LIBRARY ieee;

USE ieee.std\_logic\_1164.ALL;

ENTITY sumador\_completo IS

PORT

( a : IN STD\_LOGIC;

b : IN STD\_LOGIC;

carry\_in : IN STD\_LOGIC;

suma : OUT STD\_LOGIC;

carry\_out : OUT STD\_LOGIC );

END sumador\_completo;

ARCHITECTURE behavioral OF sumador\_completo IS

BEGIN

PROCESS (a, b, carry\_in)

BEGIN

suma <= a XOR b XOR carry\_in;

carry\_out <= (a AND b) OR (carry\_in and(a xor b));

END PROCESS;

END behavioral;

7)library IEEE;

use IEEE.STD\_LOGIC\_1164.ALL;

entity Full\_Adder\_TB is

end Full\_Adder\_TB;

architecture Behavioral of Full\_Adder\_TB is

signal A, B, Cin : STD\_LOGIC;

signal Sum, Cout : STD\_LOGIC;

component Full\_Adder

Port ( A : in STD\_LOGIC;

B : in STD\_LOGIC;

Cin : in STD\_LOGIC;

Sum : out STD\_LOGIC;

Cout : out STD\_LOGIC);

end component;

begin

UUT: Full\_Adder Port map (

A => A, B => B, Cin => Cin, Sum => Sum, Cout => Cout );

Process

begin

A <= '0';

B <= '0';

Cin <= '0';

wait for 10 ns;

A <= '0';

B <= '0';

Cin <= '1';

wait for 10 ns;

A <= '0';

B <= '1';

Cin <= '0';

wait for 10 ns;

A <= '0';

B <= '1';

Cin <= '1';

wait for 10 ns;

A <= '1';

B <= '0';

Cin <= '0';

wait for 10 ns;

A <= '1';

B <= '0';

Cin <= '1';

wait for 10 ns;

A <= '1';

B <= '1';

Cin <= '0';

wait for 10 ns;

A <= '1';

B <= '1';

Cin <= '1';

wait for 10 ns;

wait;

end process;

end Behavioral;