1) Identifique qué elementos constituyen los LEs de la FPGA Cyclone III y qué estructura tienen las LABs

Un LE (Logic Element) es la unidad básica de lógica programable. Cada LE incluye:

- LUT (Look up Table) de 4 entradas, usada para implementar funciones lógicas
- Flip-Flop
- Multiplexores
- Carry chain: permite realizar operaciones aritméticas como suma y resta de manera eficiente.

Las LABs (Logic Array Blocks) son agrupaciones de LEs. En Cyclone III, cada LAB tiene 10 LEs) y comparten recursos comunes:

- Interconexiones locales: Cada LAB tiene su propia red de conexiones internas, facilitando el intercambio de señales entre los LEs.
- Una Carry Chain común**: Los LEs dentro de una LAB pueden compartir una cadena de acarreo, esencial para operaciones aritméticas rápidas.
- Control de señales de reloj, habilitación y clear: Las LABs pueden compartir señales de control de reloj y reset, lo que permite sincronización.
- Memoria compartida y multiplexores para seleccionar las entradas y salidas.

2) ¿De qué se trata el Nios® II?

El Nios® II es un procesador de propósito general suave (soft processor) que Altera (ahora Intel) diseñó para ser implementado dentro de las FPGAs, como la serie Cyclone III. Es un procesador completamente programable que se configura en la FPGA junto con otros circuitos de lógica, permitiendo un alto grado de personalización.

3) ¿Qué diferencia existe entre IP cores y los bloques embebidos (ej multiplicador embebido) disponibles en la FPGA?

- IP cores (Intellectual Property cores) son bloques de lógica preconfigurados que implementan funciones específicas. Estos bloques son generalmente modulares y reutilizables y pueden ser proporcionados por el fabricante de la FPGA o por terceros. Los IP cores pueden ser blandos (implementados en la FPGA) o duros (implementados en hardware dedicado).
- Bloques embebidos (como un multiplicador embebido) son componentes específicos que ya están disponibles dentro de la FPGA. Estos bloques son duros en el sentido de que están implementados de forma física dentro de la FPGA, optimizados para un rendimiento alto. A diferencia de los IP cores, los bloques embebidos no necesitan ser programados o configurados por el usuario, ya que son parte de la arquitectura física de la FPGA.

4) ¿Qué tipo de celda de programación posee el dispositivo FPGA Cyclone III?

El FPGA Cyclone III de Intel utiliza celdas de programación basadas en Celdas de Memoria SRAM (Static RAM). Esta arquitectura permite reconfigurar la FPGA en cualquier momento.

5) Descripción en VHDL de un Flip Flop JK

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

entity JK_FlipFlop is
    Port (
        J   : in  STD_LOGIC;
        K   : in  STD_LOGIC;
        CLK : in  STD_LOGIC;
        Q   : out STD_LOGIC
    );
end JK_FlipFlop;

architecture Behavioral of JK_FlipFlop is
begin
    process(CLK)
    begin
        if rising_edge(CLK) then
            if J = '0' and K = '0' then
                Q <= Q;             -- Mantener el estado actual
            elsif J = '0' and K = '1' then
                Q <= '0';           -- Reset
            elsif J = '1' and K = '0' then
                Q <= '1';           -- Set
            elsif J = '1' and K = '1' then
                Q <= not Q;         -- Toggle (cambio de estado)
            end if;
        end if;
    end process;

end Behavioral;
```

6) Descripción en VHDL de un sumador completo de un bit

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

entity sumador_completo is
    port (
        A, B, Carry_in : in std_logic;   -- Entradas
```

```
        Sum, Carry_out : out std_logic   -- Salidas
    );
end sumador_completo;

architecture behavioral of sumador_completo is
begin
    -- Lógica del sumador completo
    Sum <= A xor B xor Carry_in;
    Carry_out <= (A and B) or (Carry_in and (A xor B));
end behavioral;
```

7) Descripción en VHDL del test bench del sumador completo de un bit

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;


entity sumador_completo_tb is

end sumador_completo_tb;


architecture test of sumador_completo_tb is

    component sumador_completo

        port (

            A, B, Carry_in : in std_logic;

            Sum, Carry_out : out std_logic

        );

    end component;


    signal A, B, Carry_in : std_logic;  -- Entradas para el sumador

    signal Sum, Carry_out : std_logic;  -- Salidas del sumador


begin
```

```
-- Instancia del sumador completo
UUT: sumador_completo port map (
    A => A,
    B => B,
    Carry_in => Carry_in,
    Sum => Sum,
    Carry_out => Carry_out
);

-- Proceso de prueba
process
begin
    -- Prueba de todas las combinaciones de entradas
    A <= '0'; B <= '0'; Carry_in <= '0';
    wait for 10 ns;
    A <= '0'; B <= '0'; Carry_in <= '1';
    wait for 10 ns;
    A <= '0'; B <= '1'; Carry_in <= '0';
    wait for 10 ns;
    A <= '0'; B <= '1'; Carry_in <= '1';
    wait for 10 ns;
    A <= '1'; B <= '0'; Carry_in <= '0';
    wait for 10 ns;
    A <= '1'; B <= '0'; Carry_in <= '1';
```

```
        wait for 10 ns;
        A <= '1'; B <= '1'; Carry_in <= '0';
        wait for 10 ns;
        A <= '1'; B <= '1'; Carry_in <= '1';
        wait for 10 ns;

        -- Finaliza la simulación
        wait;
    end process;

end test;
```