1_ En la FPGA Cyclone III, los LEs (logic elements) son los bloques lógicos básicos. Cada LE consta de una LUT (Look-Up Table) de 4 entradas, un Flip-Flop D,una lógica de acerreo y una pequeña cantidad de lógica adicional para facilitar funciones comunes como multiplexores.

Los LEs están agrupados en bloques de 8, estos bloques se denominan LABs (lógica array blocks).Cada uno de estos bloques comparten recursos como el ruteo y las conexiones hacia otro de estos bloques.

Nios II es un procesador de 32 bits diseñado para fpga (por ejemplo Cyclone III) por Intel. Es reconfigurable y se usa para tareas específicas dando la posibilidad de personalizar tanto el hardware como el software.

3_ Los IP Cores (Núcleos de Propiedad Intelectual) son módulos lógicos prediseñados por otra persona y que cualquiera puede agregar y personalizar en su diseño. No son bloques físicos que se encuentren en la FPGA sino que se implementan al momento de programar.

Los bloques embebidos son bloques físicos que están integrados dentro de la FPGA y no son personalizables. Sirven para realizar tareas específicas y no tener que usar bloques lógicos.

4_ La FPGA Cyclone III utiliza las celdas de programación SRAM (Memoria estática de acceso aleatorio). Estás memorias son reprogramables , tiene buen rendimiento, pero son un poco más lentas y consume más potencia en comparación a otro tipos de celdas

5_
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

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

architecture JK of FFJK is
    signal Q_int : STD_LOGIC := '0';
begin
    process (CLK)
    begin
        if rising_edge(CLK) then
            if (J = '0' and K = '0') then
                Q_int <= Q_int; -- mantiene el estado
            elsif (J = '0' and K = '1') then
```

```
            Q_int <= '0'; -- reset
        elsif (J = '1' and K = '0') then
            Q_int <= '1'; -- set
        elsif (J = '1' and K = '1') then
            Q_int <= not Q_int; -- toggle
        end if;
      end if;
   end process;

   Q <= Q_int;
   Qn <= not Q_int;
end JK;
```

6_

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

entity FA is
   Port ( A : in STD_LOGIC;
          B : in STD_LOGIC;
          Cin : in STD_LOGIC;
          Sum : out STD_LOGIC;
          Cout : out STD_LOGIC);
end FA;

architecture suma of FA is
begin
   Sum <= A XOR B XOR Cin;
   Cout <= (A AND B) OR (A AND Cin) OR (B AND Cin);
end suma;
```

7_

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

entity tbFA is
end tbFA;

architecture suma of tbFA is
   signal A, B, Cin : STD_LOGIC := '0';
   signal Sum, Cout : STD_LOGIC;

   component FA
      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: FA Port map ( A => A, B => B, Cin => Cin, Sum => Sum, Cout => Cout );

    process
    begin
        -- 0 + 0 + 0
        A <= '0'; B <= '0'; Cin <= '0'; wait for 10 ns;
        -- 0 + 0 + 1
        A <= '0'; B <= '0'; Cin <= '1'; wait for 10 ns;
        -- 0 + 1 + 0
        A <= '0'; B <= '1'; Cin <= '0'; wait for 10 ns;
        -- 1 + 1 + 1
        A <= '1'; B <= '1'; Cin <= '1'; wait for 10 ns;
        wait;
    end process;

end suma;
```