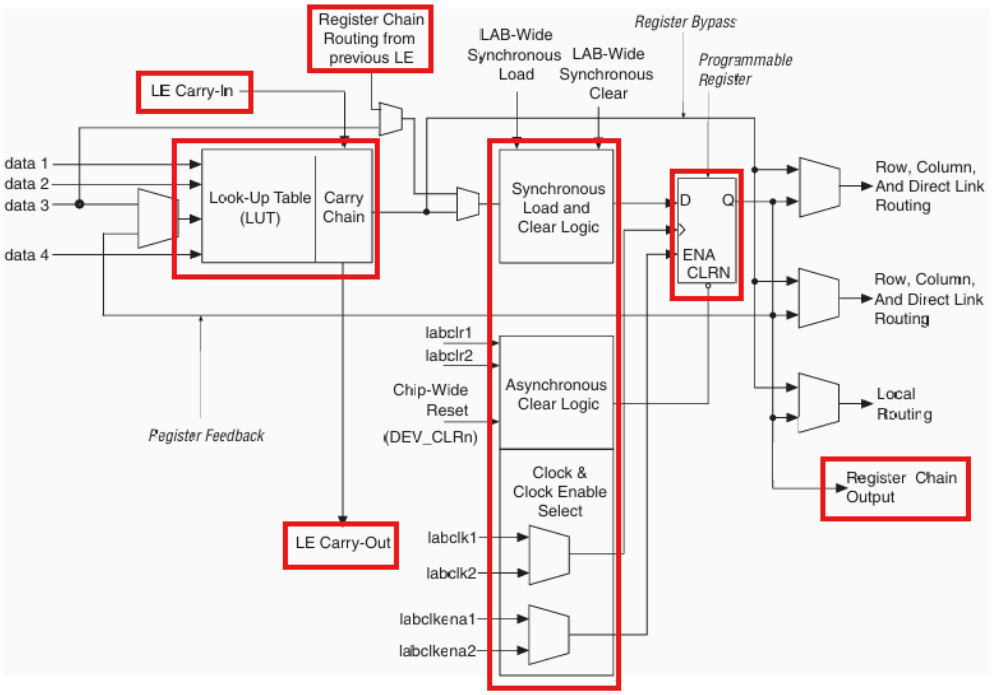
1) Los elementos lógicos de la FPGA Cyclone 3 están compuestos de una Look-Up Table de 4 entradas, un registro programable, los bloques lógicos necesarios para el control del registro y las entradas y salidas que permiten la interconexión entre los distintos LE.

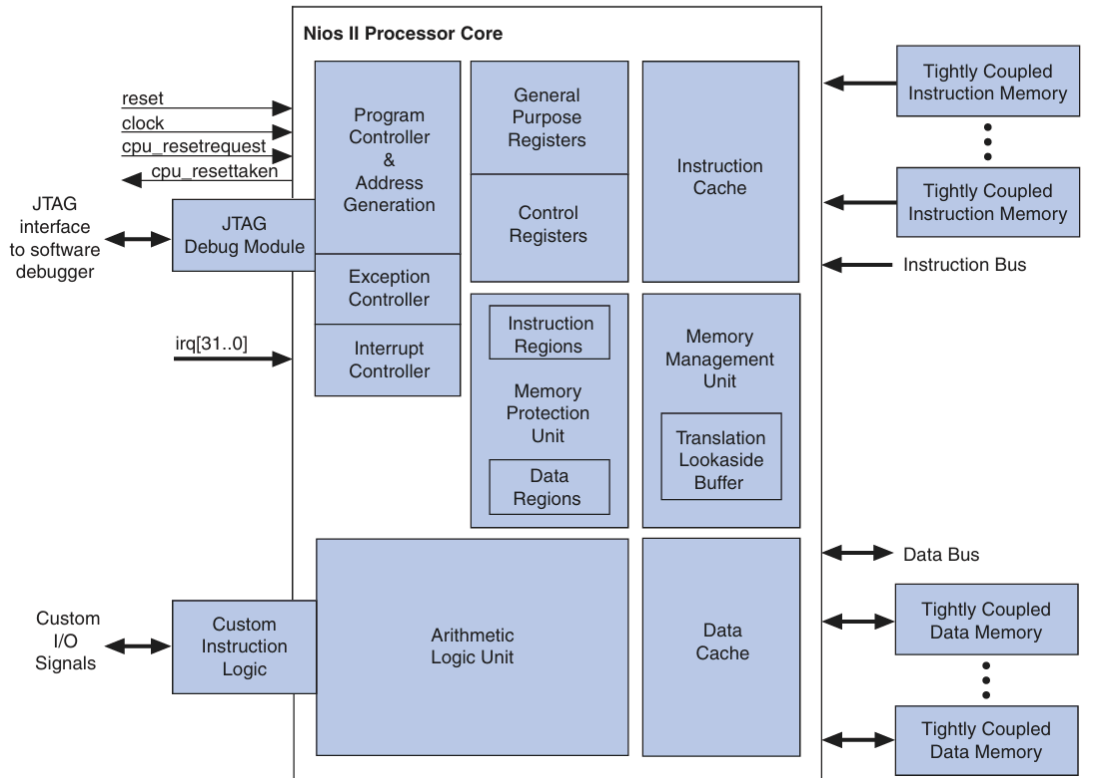


2) El Nios II es un procesador embebido programable de 32 bits que puede implementarse en las FPGAs Cyclone III y Cyclone IV. No está colocado físicamente en el chip, sino que se implementa mediante los elementos lógicos y bloques de memoria de la fpga.

Posee buses de datos e instrucciones, interfaces para instrucciones personalizadas y unidades de gestión y protección de memoria. Incluye una ALU que puede configurarse para realizar diversas operaciones:

Aritméticas: suma, resta, multiplicación y división de hasta 32×32 bits, con operandos con o sin signo.
Relacionales: comparaciones de igual, distinto, mayor, menor o igual para datos con operandos con y sin signo
Lógicas: operaciones AND, OR, XOR, NOR, entre otras.
De desplazamiento

. Nios II Processor Core Block Diagram



3) Los IP cores son bloques programados en VHDL o Verilog, están prediseñados para cumplir funciones específicas y pueden ser implementados en cualquier proyecto de FPGA para simplificar su diseño.
Los bloques embebidos cumplen funciones parecidas a los IP cores pero están implementados físicamente en la FPGA, por lo que estos son más eficientes y rápidos.

4) Las Cyclone 3 tienen celdas de programación de tipo SRAM para guardar los datos de configuración. Debido a que la memoria SRAM es volátil requieren memorias externas a la FPGA para recibir la información una vez que el chip recibe energía.

5) Flip-Flop JK

```
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;
        not_Q : out std_logic
         );
end ffjk;

architecture fnc of ffjk is
   signal estado : std_logic := '0';
   begin
      ffjk : process (clk) is
      variable V : std_logic_vector(1 downto 0);
         begin
            if rising_edge(clk) then
            V := J & K;
               case V is
                  when "00" =>
                     estado <= estado;
                  when "01" =>
                     estado <= '0';
                  when "10" =>
                     estado <= '1';
                  when "11" =>
                     estado <= not(estado);
               end case;
            end if;
         Q <= estado;
         not_Q <= not(estado);
      end process;
end fnc;
```

6) Full Adder 1 bit

```
library ieee;
use ieee.std_logic_1164.all;

entity FullAdder is
   port(A: in std_logic;
        B: in std_logic;
        CIN: in std_logic;
        S: out std_logic;
        COUT: out std_logic);
end FullAdder;

architecture fnc of FullAdder is
   signal xor1: std_logic;
   signal and1: std_logic;
   signal and2: std_logic;
   signal and3: std_logic;

begin
   xor1 <=B xor CIN;
   and1 <=B and CIN;
   and2 <=B and A;
   and3 <=A and CIN;
   S <=A xor xor1;
   COUT <=and1 or and2 or and3;
end fnc;
```

7) Testbench Full Adder 1 bit

```
library ieee;
use ieee.std_logic_1164.all;

entity FullAdder_tb is
end FullAdder_tb;

architecture fnc of FullAdder_tb is

    component FullAdder
       port(A : in std_logic;
            B : in std_logic;
            CIN : in std_logic;
            S : out std_logic;
            COUT : out std_logic
            );
    end component;

    signal A : std_logic := '0';
    signal B : std_logic := '0';
    signal CIN : std_logic := '0';
    signal S : std_logic;
    signal COUT : std_logic;

begin
    uut: FullAdder port map(
       A => A,
       B => B,
       CIN => CIN,
       S => S,
       COUT => COUT
       );

stim_proc: process
begin
    A <= '0'; B <= '0'; CIN <= '0'; wait for 10ns;
    A <= '1'; B <= '0'; CIN <= '0'; wait for 10ns;
    A <= '0'; B <= '1'; CIN <= '0'; wait for 10ns;
    A <= '1'; B <= '1'; CIN <= '0'; wait for 10ns;
    A <= '0'; B <= '0'; CIN <= '1'; wait for 10ns;
    A <= '1'; B <= '0'; CIN <= '1'; wait for 10ns;
    A <= '0'; B <= '1'; CIN <= '1'; wait for 10ns;
    A <= '1'; B <= '1'; CIN <= '1'; wait for 10ns;
    wait;
end process;
end;
```