# Systèmes embarqués automobile et aéronautique

**Master d'Université Spécialisé (MUS) en**
**Ingénierie des Systèmes Embarqués (ISE)**

# Chapitre 6: Les SoC (System on Chip) pour les applications embarquées avancées

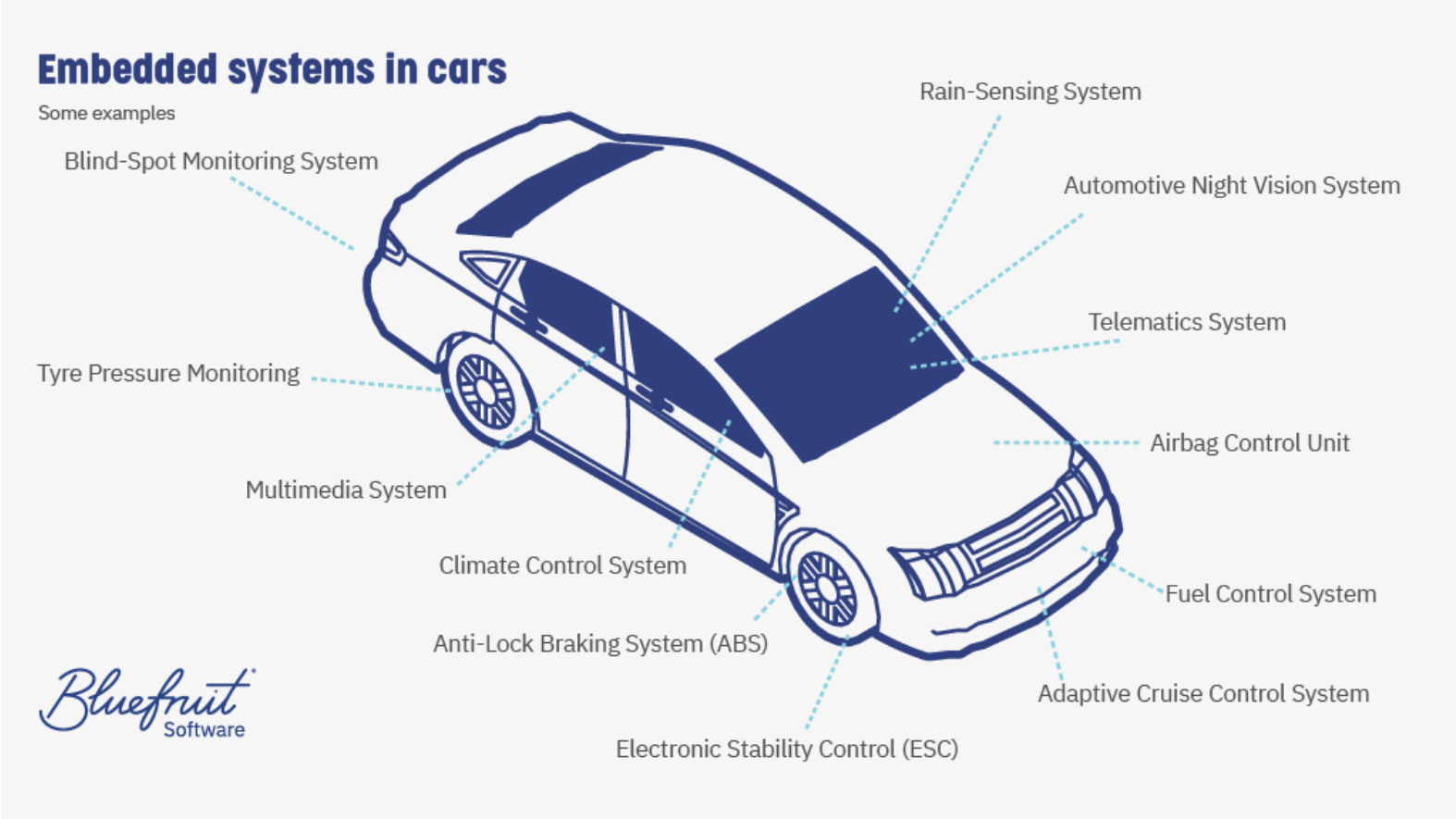
## Application: Implémentation d'un SoC Dual Core

**Année universitaire 2024/2025**

**Dr. Ing. LASSIOUI Abdellah**

# Introduction:

Les véhicules modernes intègrent plusieurs dizaines, voire centaines de capteurs pour surveiller et contrôler divers systèmes tels que le moteur, la sécurité, l'aide à la conduite et le confort. Ces capteurs collectent en temps réel des données sur la vitesse, la température, la pression, les obstacles environnants et bien plus encore. Leur intégration croissante reflète l'évolution vers des voitures plus intelligentes et autonomes

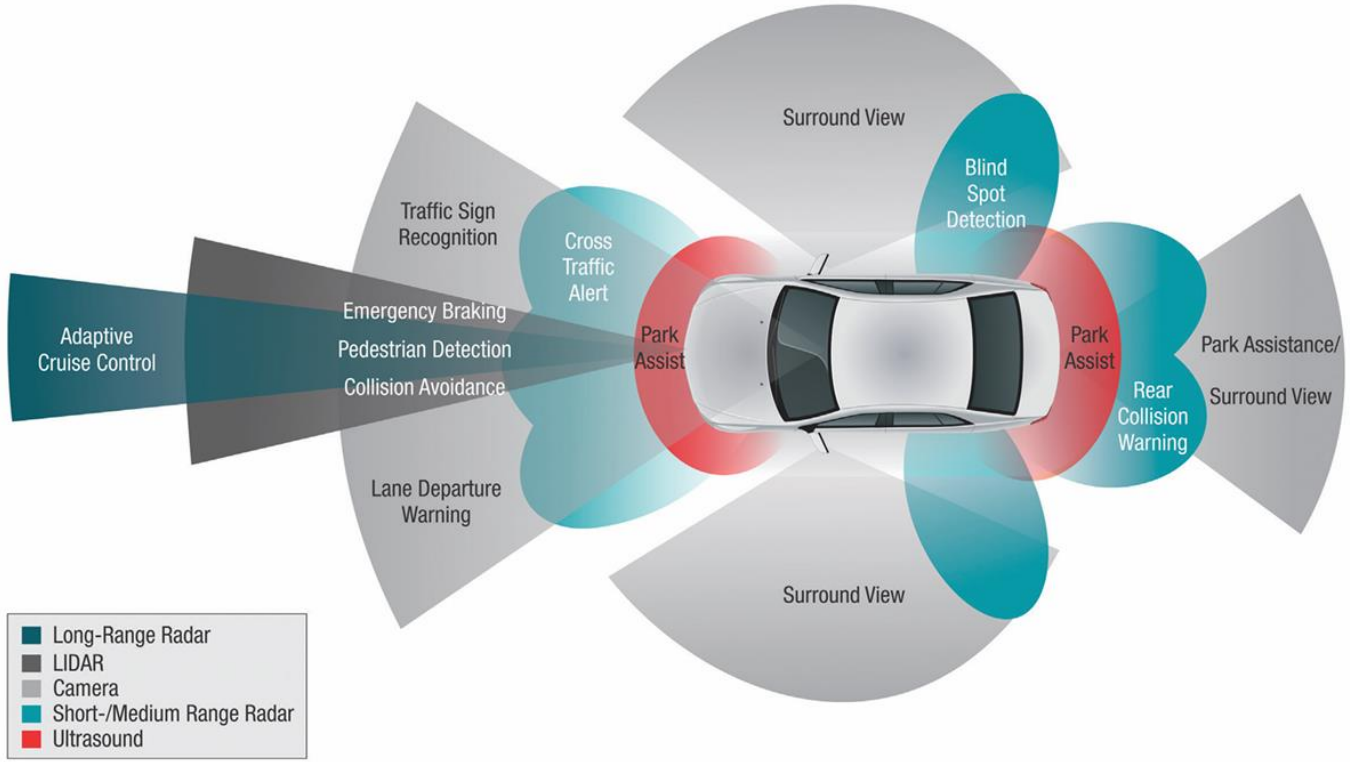



*https://bluefruit.co.uk/processes/what-embedded-systems-in-car/*

# Complexité des véhicules modernes:

## Exemple de la conduite autonome

La conduite autonome repose sur une forte intégration de contrôleurs embarqués, traitant en temps réel des données issues de capteurs, caméras et lidars. Ces contrôleurs exécutent des algorithmes avancés pour la perception, la planification de trajectoire et la prise de décision. Cette centralisation et l'optimisation du traitement garantissent une réponse rapide et fiable, essentielle pour la sécurité et l'efficacité des véhicules autonomes.




*Source :*
*Advanced Driver Assistance (ADAS) Solutions Guide (Texas Instrument)*

# Limitations des contrôleurs classiques et migration vers les SoC

Les microcontrôleurs traditionnels, limités en puissance de calcul et en traitement parallèle, peinent à gérer ces tâches en temps réel.

Pour répondre à ces exigences croissantes, les SoC (System-on-Chip) offrent des architectures multi-cœurs et des accélérateurs dédiés, optimisant ainsi les performances des systèmes embarqués.

# Définition d’un SoC

Un **SoC** (**System on Chip**) est un circuit intégré qui regroupe **plusieurs composants** d'un système informatique complet sur une seule puce.

Ces composants peuvent inclure :

- **Un ou plusieurs processeur** (CPU)
- **Mémoire** (RAM, ROM)
- **Unités de traitement spécialisées** (GPU, DSP pour le traitement du signal, etc.)
- **Interfaces de communication** (UART, SPI, I2C, Ethernet, USB, etc.)
- **Blocs analogiques** (convertisseurs ADC/DAC, régulateurs de tension, etc.)
- **Contrôleurs de périphériques** (écran, capteurs, stockage, etc.)

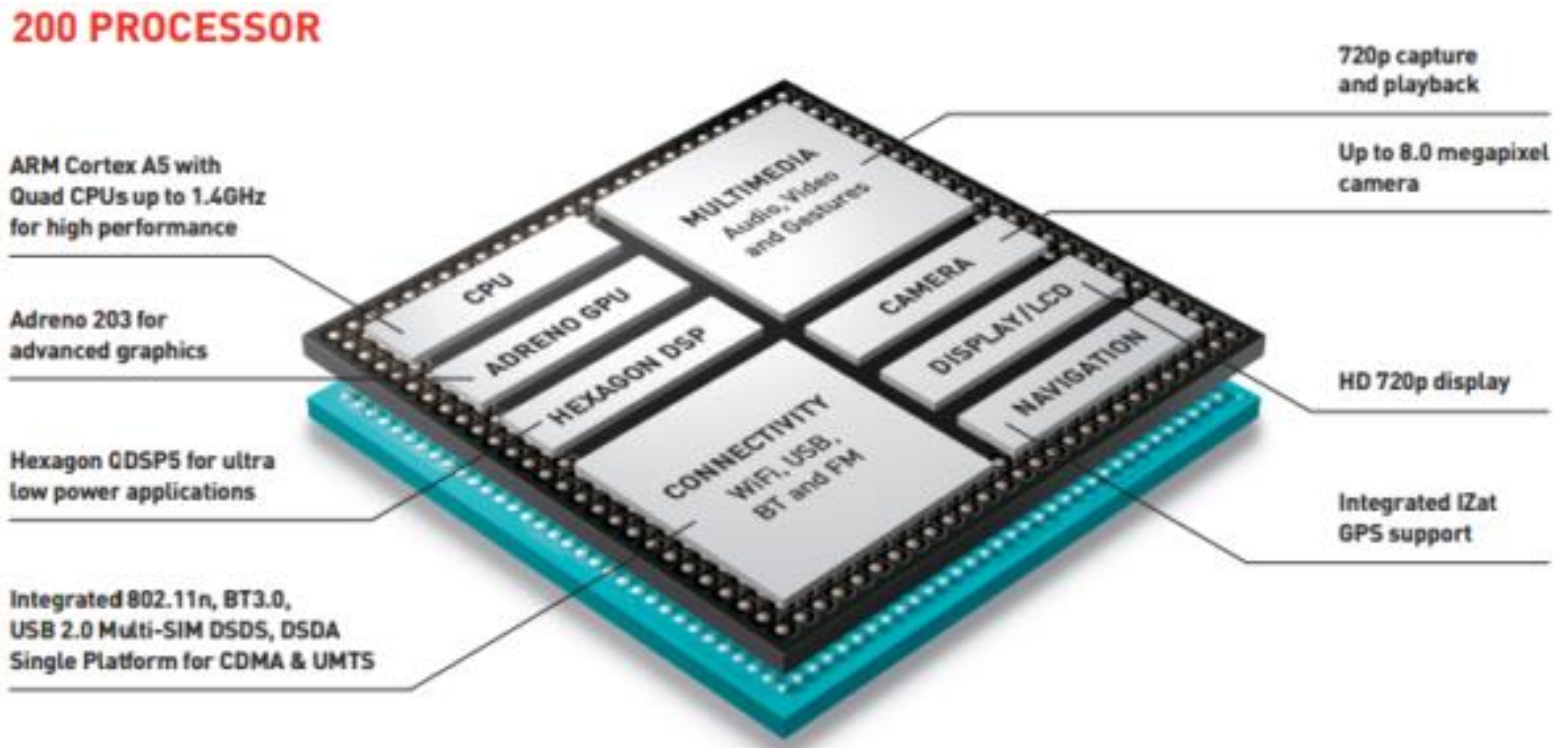
# Caractéristiques d’un SoC

- **Intégration élevée** : Tout ou presque est intégré sur une seule puce, ce qui réduit la taille globale du système et les coûts de production.
- **Faible consommation d'énergie** : Optimisé pour des applications embarquées, le SoC est souvent conçu pour consommer très peu d’énergie.
- **Haute performance** : Les composants intégrés communiquent directement sur la même puce, réduisant les temps de latence.

# Exemple d’un SoC: Qualcomm Snapdragon

Un SoC comme le **Snapdragon** de Qualcomm dans les smartphones intègre :

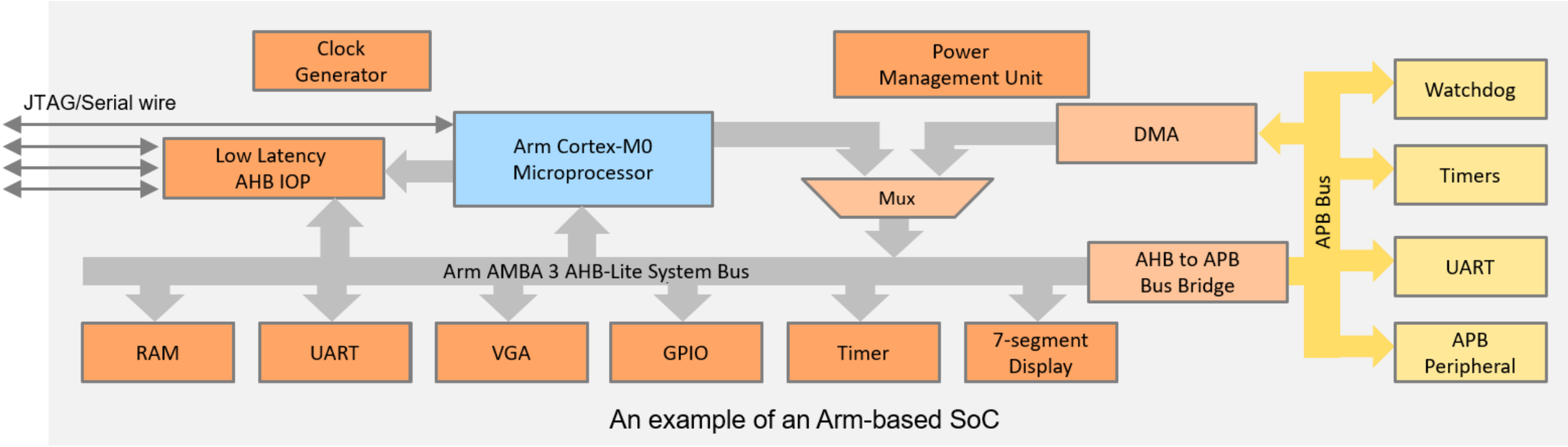
- ❑ Un **processeur principal** (CPU) pour les calculs.
- ❑ Un **GPU** pour le rendu graphique.
- ❑ Un **DSP**
- ❑ Un contrôleur de connectivité (WiFi, BT, USB,…)
- ❑ Des **contrôleurs** pour les périphériques comme caméras, écrans, etc.




*Source: https://techawarness.com/what-is/what-is-qualcomms-snapdragon-soc/*

# Exemple d’un SoC: Le processeur ARM (Advanced RISC Machines)

Un SoC Arm de base se compose généralement d'un processeur Arm, tel que le **Cortex-M0**, d’un bus AMBA (Advanced Microcontroller Bus Architecture, d’un ou plusieurs périphériques (UART, VGA, GPIO, TIMERS…), d’un générateur d’horloge, d’une unité de gestion d’alimentation, AHB (Advanced High Performance Bus), APB (Advanced Peripheral Bus)…

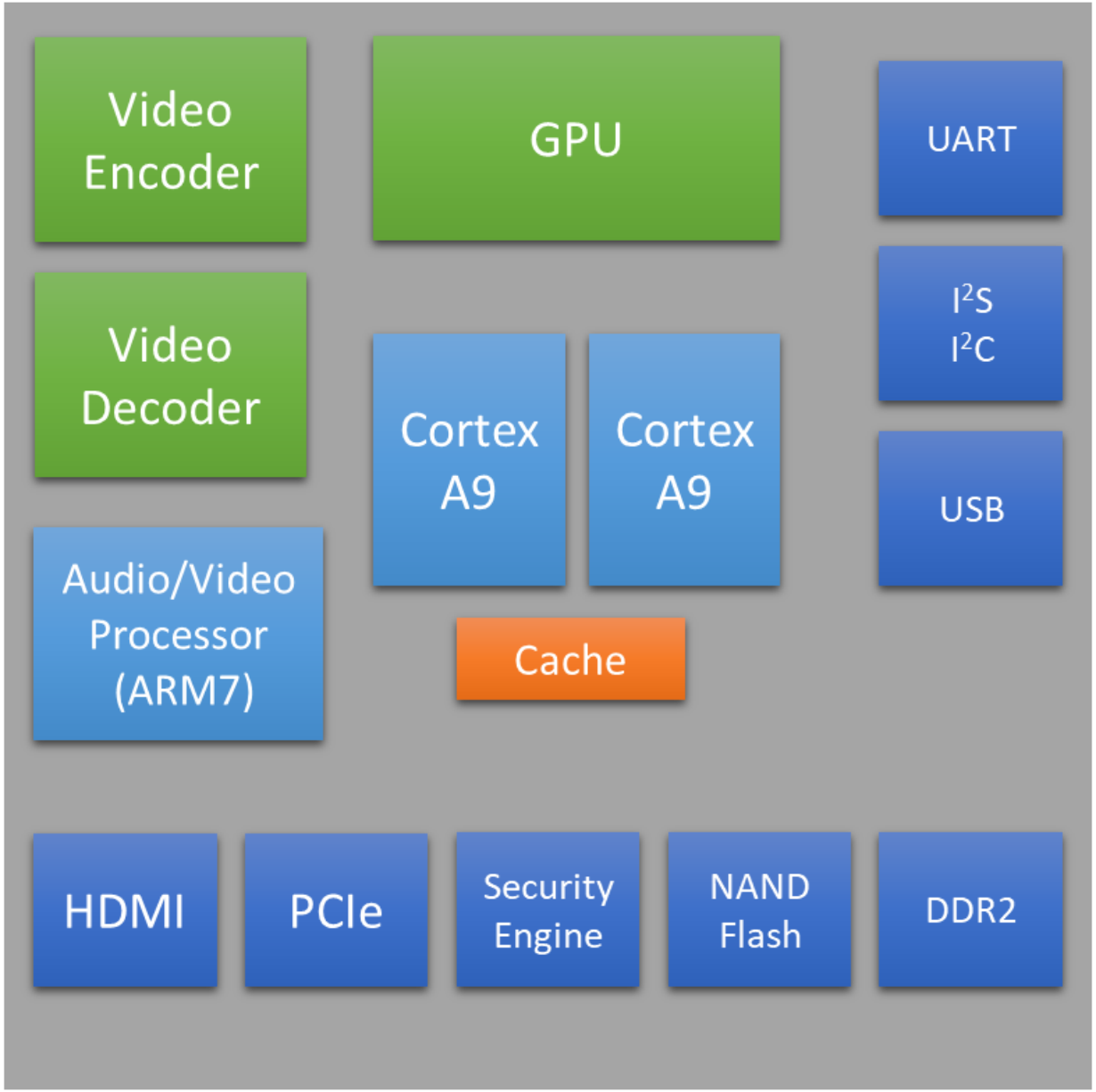



An example of an Arm-based SoC

# Exemple d’un SoC: La carte NVIDEA Tegra 2

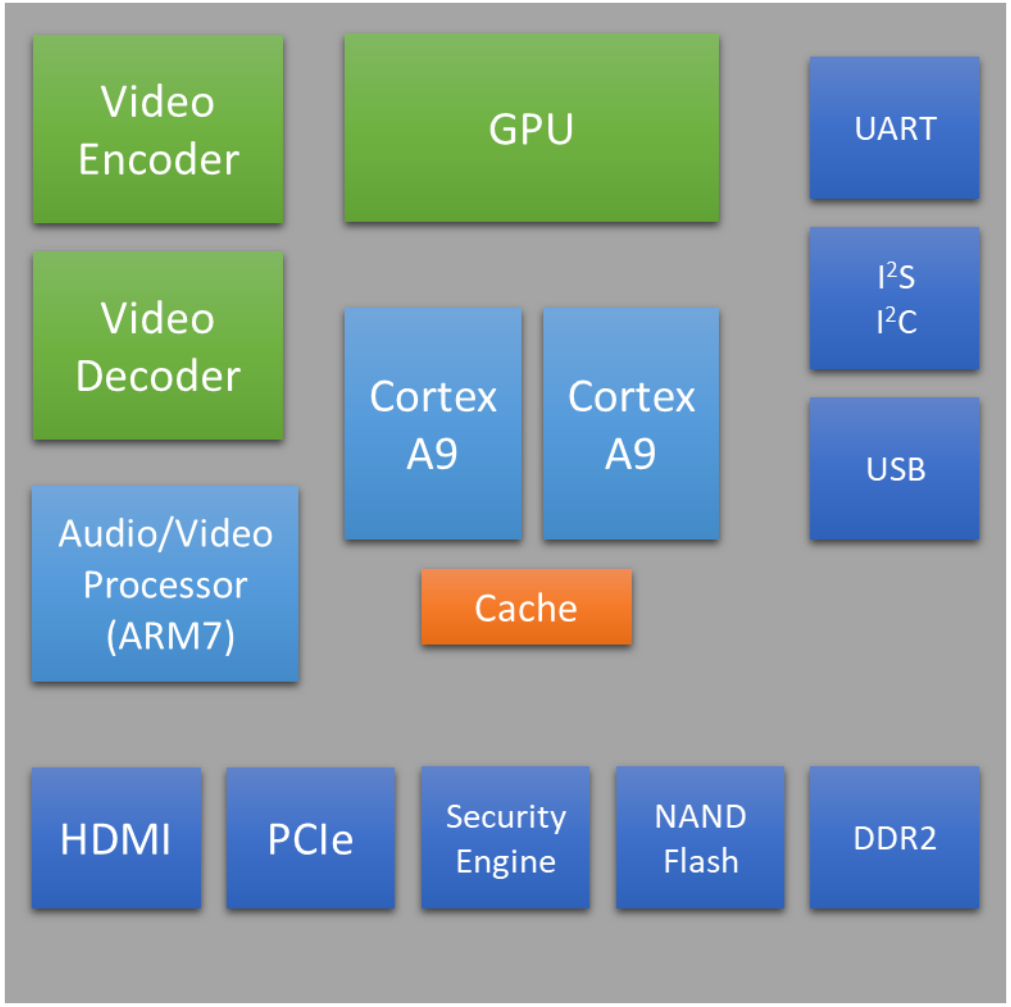
La fonction principale du Tegra 2 est de fournir un système intégré complet permettant :

- ❑ **Calcul efficace** (double cœur ARM Cortex-A9).
- ❑ **Traitement graphique avancé** (GPU ULP GeForce).
- ❑ **Support multimédia** (lecture/encodage vidéo HD 1080p).
- ❑ **Optimisation de la consommation d'énergie** pour une utilisation dans les systèmes embarqués.



# Exemple d’un SoC: La carte NVIDEA Tegra 2

| | |
|---|---|
| Designer | NVIDIA |
| Year | 2010 |
| Processor | Arm Cortex-A9 (dual-core) |
| Frequency | Up to 1.2 GHz |
| Memory | 1 GB 667 MHz LP-DDR2 |
| Graphics | ULP GeForce |
| Process | 40 nm |
| Package | 12 × 12 mm (package on package) |



# Le NIOS II Embedded Processor

Le Nios II est un processeur softcore développé par Intel (anciennement Altera), conçu pour être intégré dans des systèmes sur puce (SoC) basés sur FPGA. Il offre une grande flexibilité en permettant aux concepteurs d'adapter l'architecture selon les besoins spécifiques de l'application.

# Le NIOS II Embedded Processor

**Caractéristiques principales du Nios II dans les SoC**

**1.Architecture configurable** : Contrairement aux microcontrôleurs classiques, le Nios II peut être personnalisé en termes de mémoire, d'unités de calcul et de périphériques.

**2.Intégration avec FPGA** : Fonctionne sur les FPGA Intel (ex-Altera), permettant une adaptation rapide aux exigences des systèmes embarqués.

**3.Optimisation des performances** : Disponible en plusieurs variantes (économique, standard, performant) selon le besoin en puissance de calcul.

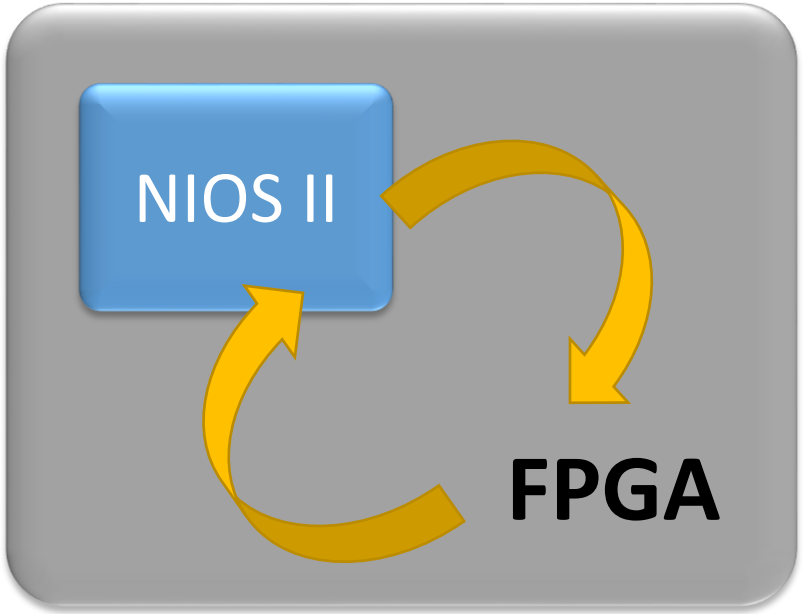
**4.Applications** : Utilisé dans l'automobile, les télécommunications, le contrôle industriel et les systèmes embarqués avancés.

# Le NIOS II Embedded Processor

- **Famille: Un processeur de 32Bits**
- **Performances : 300 MIPS**
- **Pourcentage des CLB occupé : 25% (Cyclone III FPGA)**
- **Contrairement au processeurs fixes (ARM), NIOS est configurable (possibilité d'ajouter des périphériques en fonction du besoin)**

# Le NIOS II Embedded Processor

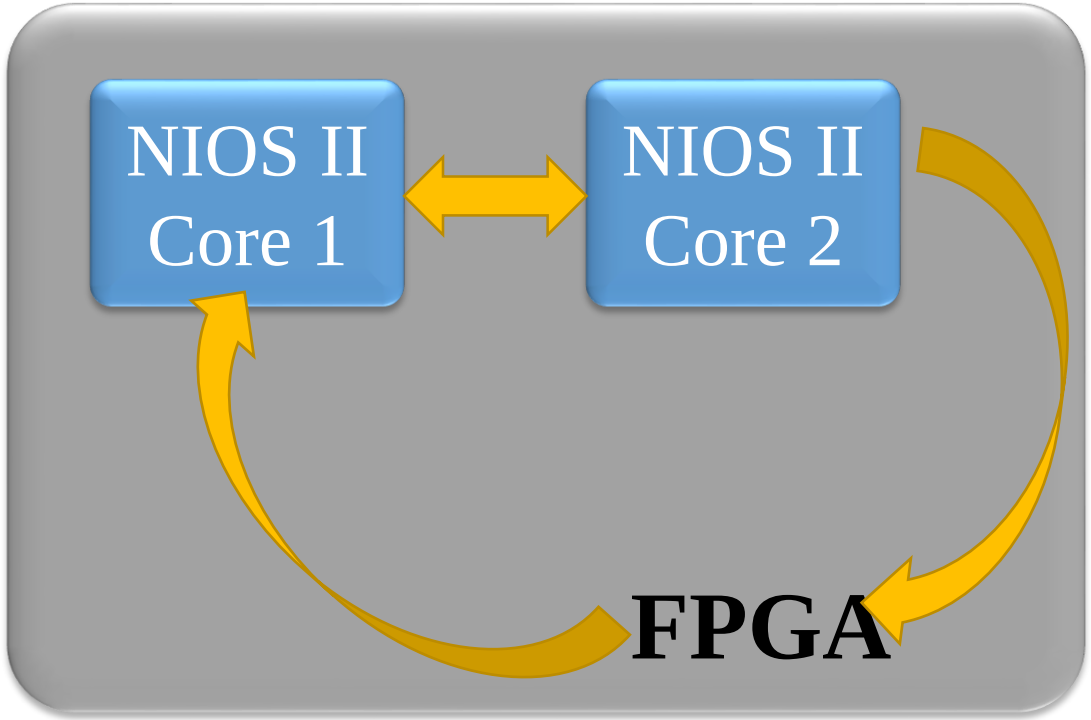
## ❑ Coexistence avec les circuits FPGA:




- ❑ **Combinaison d'un traitement logiciel (via NIOS II) et d'un traitement matériel (via les CLBs)**
- ❑ Le processeur NIOS II peut travailler en parallèle avec des blocs matériels personnalisés pour des **tâches critiques en temps réel** ou des **traitements de signaux rapides**.

# Le NIOS II Embedded Processor

- **Plusieurs Nios II** peuvent être ajoutés dans un même FPGA, en fonction de la taille et des ressources disponibles sur le circuit FPGA (l'espace logique, la mémoire, les entrées/sorties, etc.).
- **Parallélisme et multiprocessing**: Utiliser plusieurs processeurs Nios permet de créer des systèmes parallèles, où chaque processeur peut gérer des tâches distinctes



# Le NIOS II Embedded Processor

- Il existe trois variantes du processeur NIOS II
- NIOS II/f Core (Pipeline 6, Hrdware Multiplier 1 cycle, Logic Elements 1800)
- NIOS II/s Core (Pipeline 5, Hrdware Multiplier 3 cycle, Logic Elements 1200)
- NIOS II/e Core (Pipeline None, Hrdware Multiplier xx, Logic Elements 600)

# Le NIOS II Embedded Processor

## ❑ Notion du pipeline

Le pipeline décrit l'enchainement d'exécution d'instructions dans un processeur :

**1.IF (Instruction Fetch)** : L'instruction est récupérée de la mémoire.

**2.ID (Instruction Decode)** : L'instruction est décodée pour déterminer quelles opérations doivent être effectuées et quels registres ou mémoires doivent être accédés.

**3.EX (Execute)** : L'opération est exécutée (par exemple, une addition ou un accès mémoire).

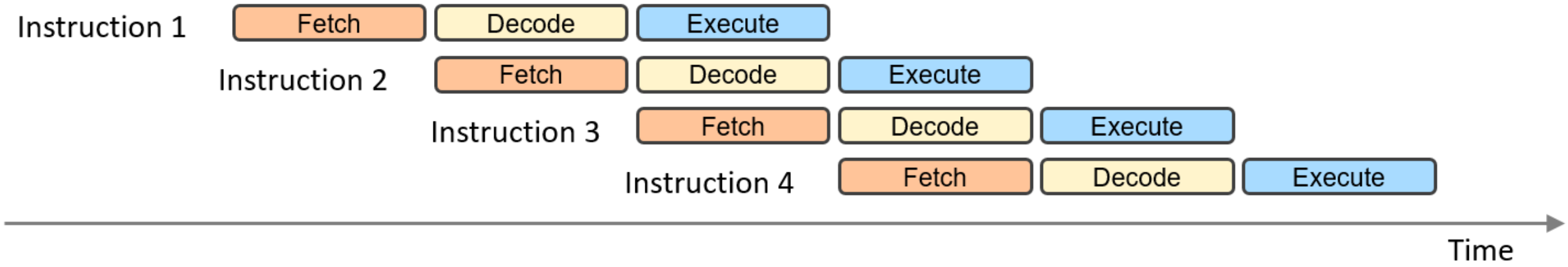
**4.MEM (Memory Access)** : Si nécessaire, l'accès à la mémoire est effectué (lecture ou écriture).

**5.WB (Write Back)** : Le résultat est écrit dans les registres ou dans la mémoire.

# Le NIOS II Embedded Processor

## ❑ Notion du pipeline

Ce diagramme illustre les principes de fonctionnement d'un pipeline à trois étapes. L'essence de cette architecture est qu'elle permet de traiter jusqu'à trois instructions simultanément. Par exemple, lorsque la première instruction est en cours d'exécution, le processeur peut décoder la deuxième instruction et récupérer la troisième instruction en même temps, ce qui contribue à améliorer le débit du pipeline du processeur, ce qui se traduit par de meilleures performances.



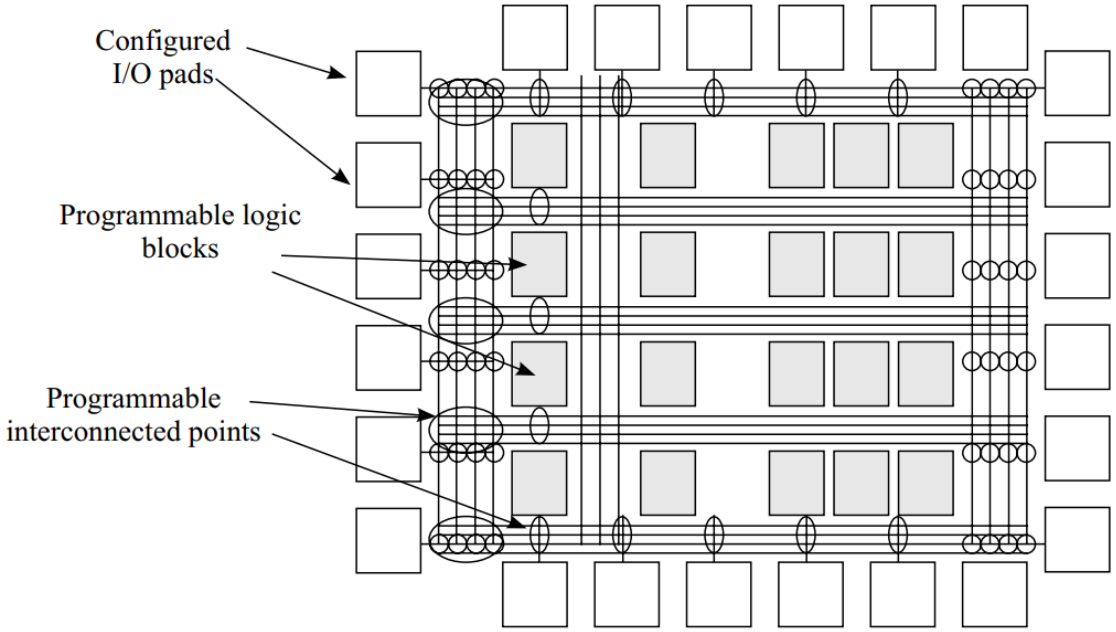
# Le NIOS II Embedded Processor

❑ **Notion du pipeline: exemple d'une pipeline à 6 cycles**

| Cycle 1 | Cycle 2 | Cycle 3 | Cycle 4 | Cycle 5 | Cycle 6 |
|---|---|---|---|---|---|
| IF (I1) | IF (I2) | IF (I3) | IF (I4) | IF (I5) | IF (I6) |
| | ID (I1) | ID (I2) | ID (I3) | ID (I4) | ID (I5) |
| | | EX (I1) | EX (I2) | EX (I3) | EX (I4) |
| | | | MEM (I1) | MEM (I2) | MEM (I3) |
| | | | | WB (I1) | WB (I2) |
| | | | | | RET (I1) |

# Le NIOS II Embedded Processor

- Sur un circuit FPGA on peut implémenter plusieurs Microprocesseur NIOS II
- NIOS II/f Core: Logic Elements 1800

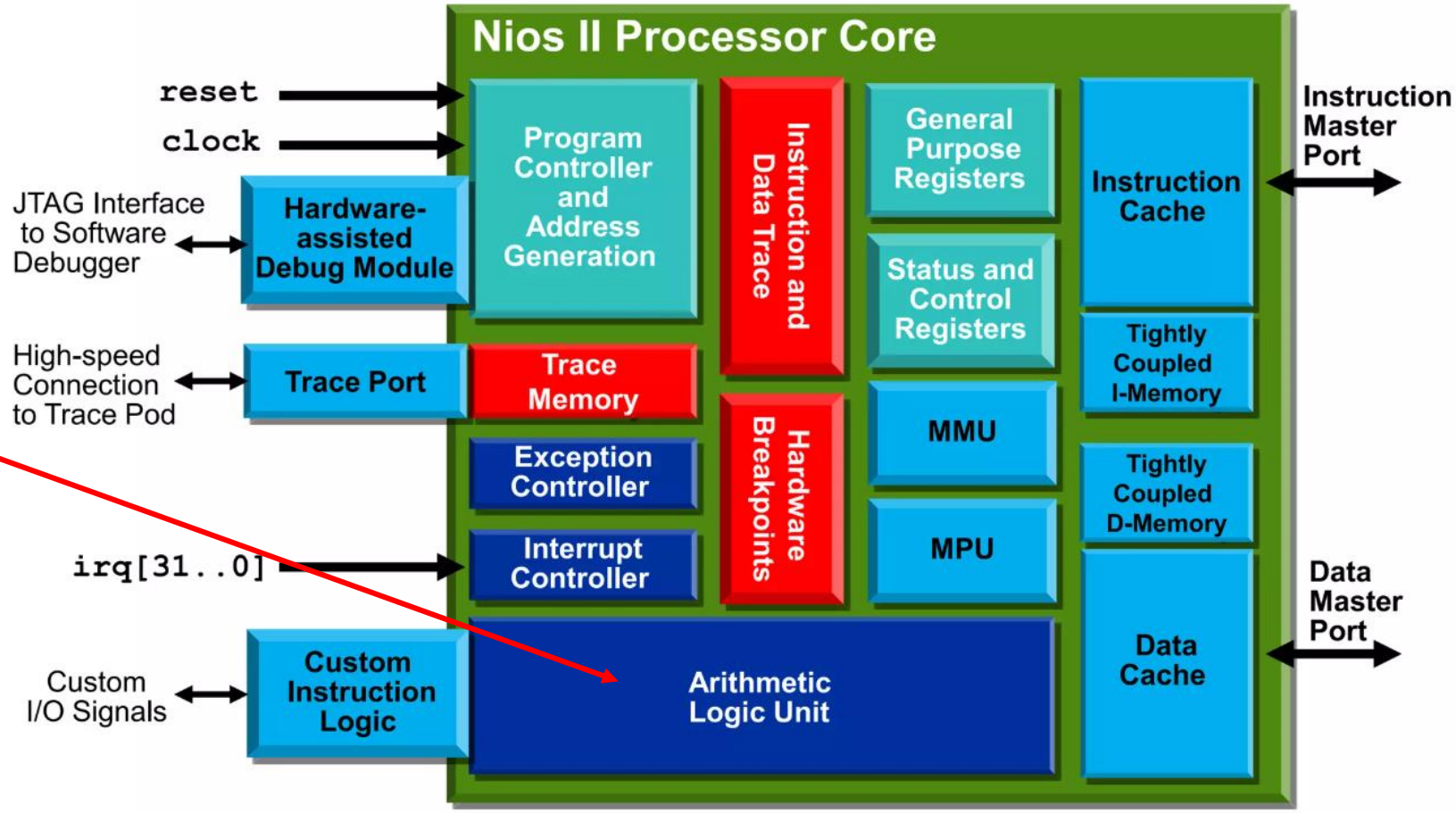



**Exemple: Cyclone V FPGA device familty (source Intel)**

| Density | Ordering Part Number (OPN) |
|---|---|
| 25K LE | 5CSEBA2U19I7LN |
| | 5CSEBA2U23I7LN |
| | 5CSXFC2C6U23I7LN |
| 40K LE | 5CSEBA4U19I7LN |
| | 5CSEBA4U23I7LN |
| | 5CSXFC4C6U23I7LN |
| 85K LE | 5CSEBA5U19I7LN |
| | 5CSEBA5U23I7LN |
| | 5CSXC5C6U23I7LN |
| 110K LE | 5CSEBA6U19I7LN |
| | 5CSEBA6U23I7LN |
| | 5CSXFC6C6U23I7LN |

# Le NIOS II Embedded Processor architecture

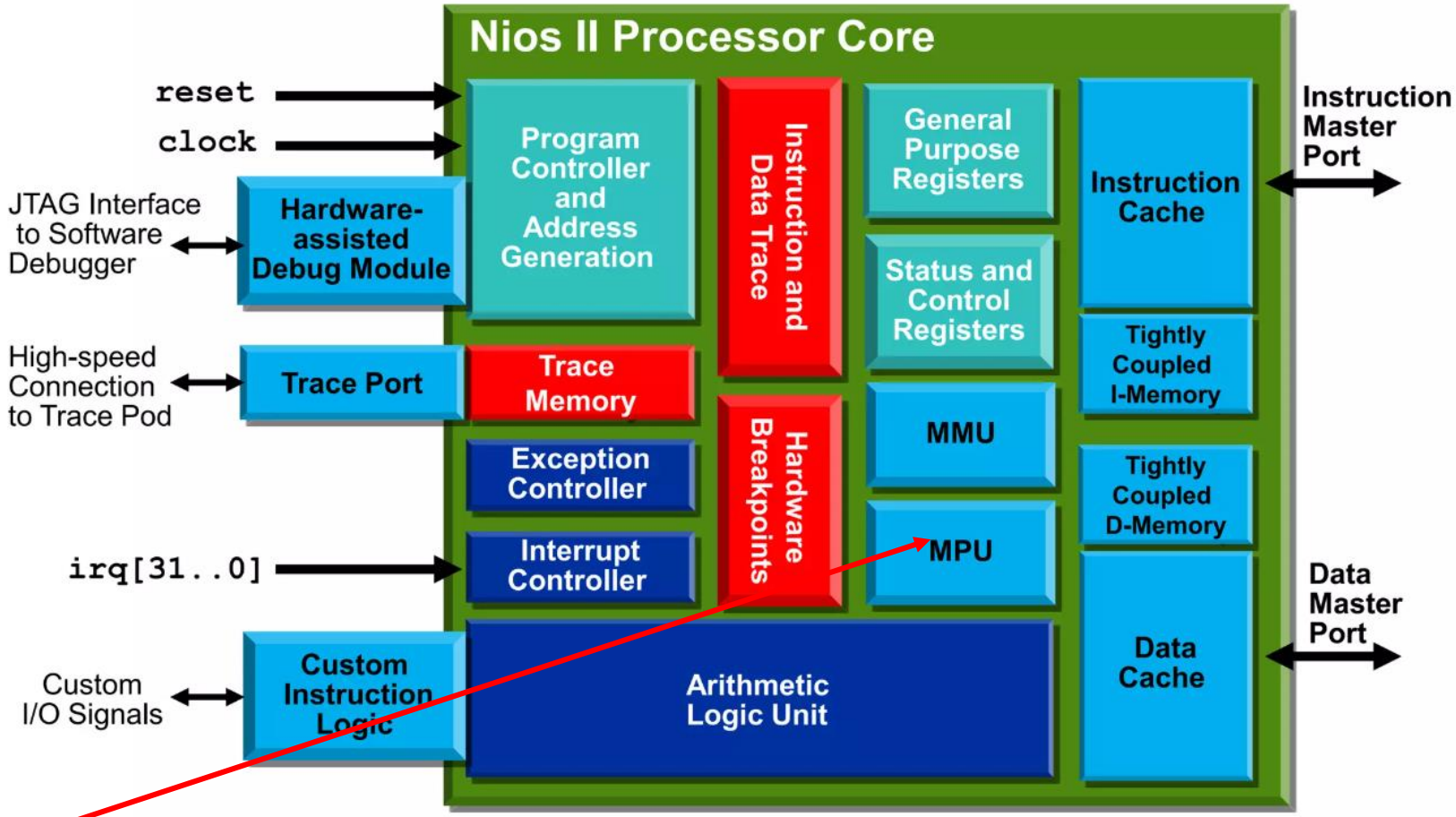
- **ALU: une unité performant les opérations arithmétiques et logiques**




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

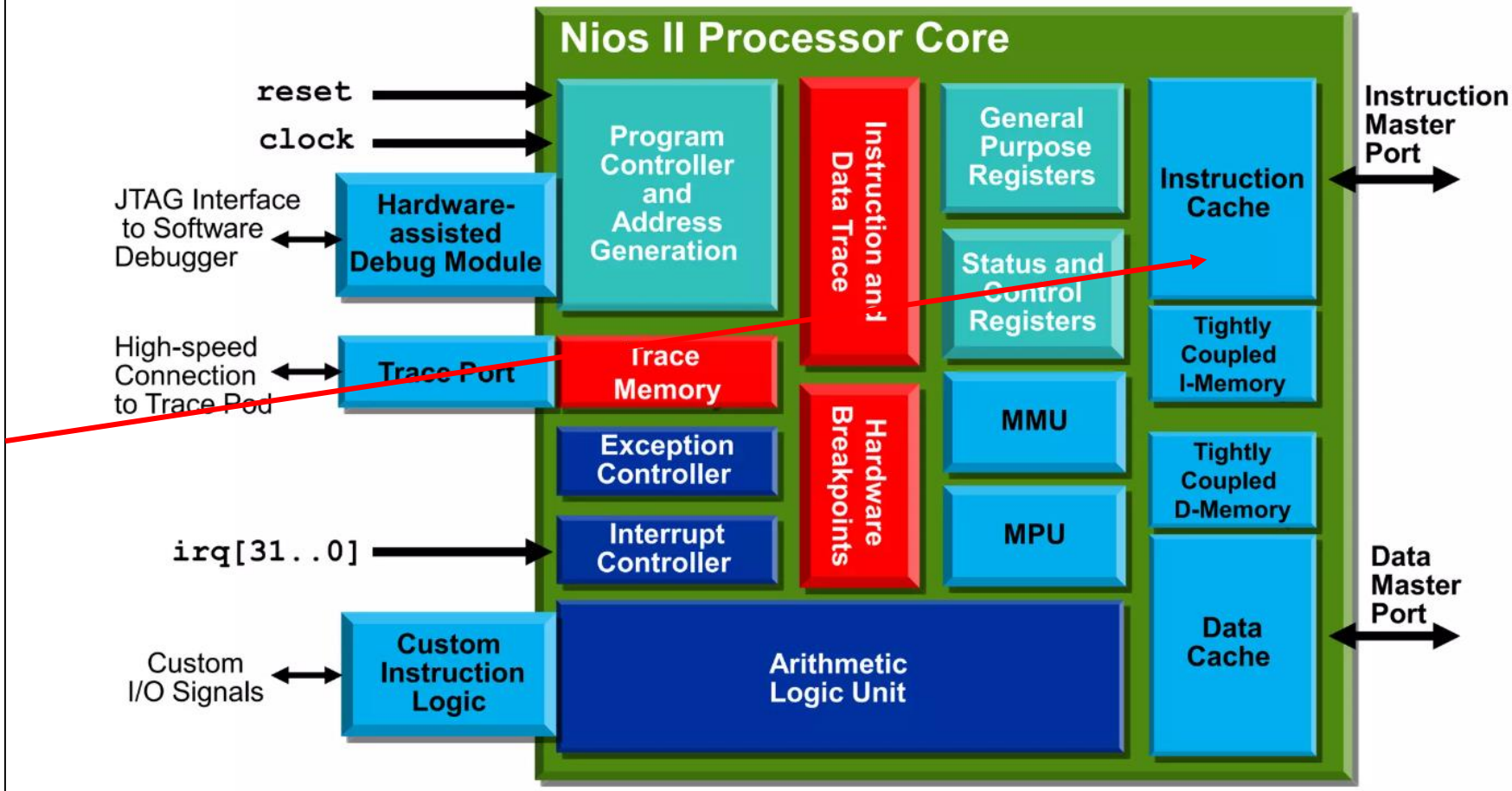
- **MPU, MMU (Memory Protection Unit et Management Unit): elles assurent la protection et le management de la mémoire. (Divise la mémoire en plusieurs régions. Pour chaque région on définit des droits d'accès spécifiques (lecture, écriture)).**




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

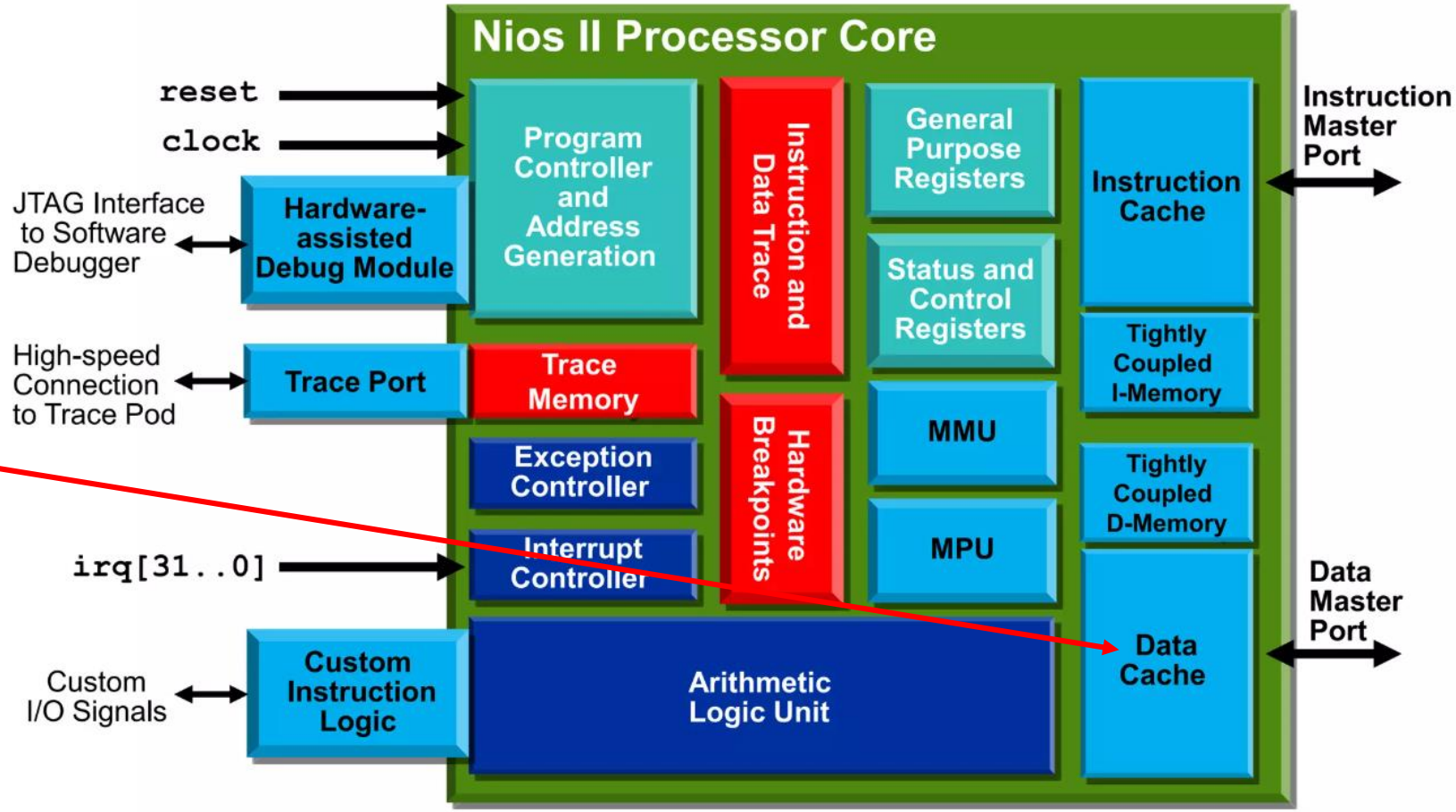
- **Instruction Cache: il s'agit d'une mémoire rapide située entre le processeur et la mémoire principale**
- **Elle stocke temporairement les instructions qui sont couramment utilisées pour accélérer l'exécution et optimiser le pipeline**




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

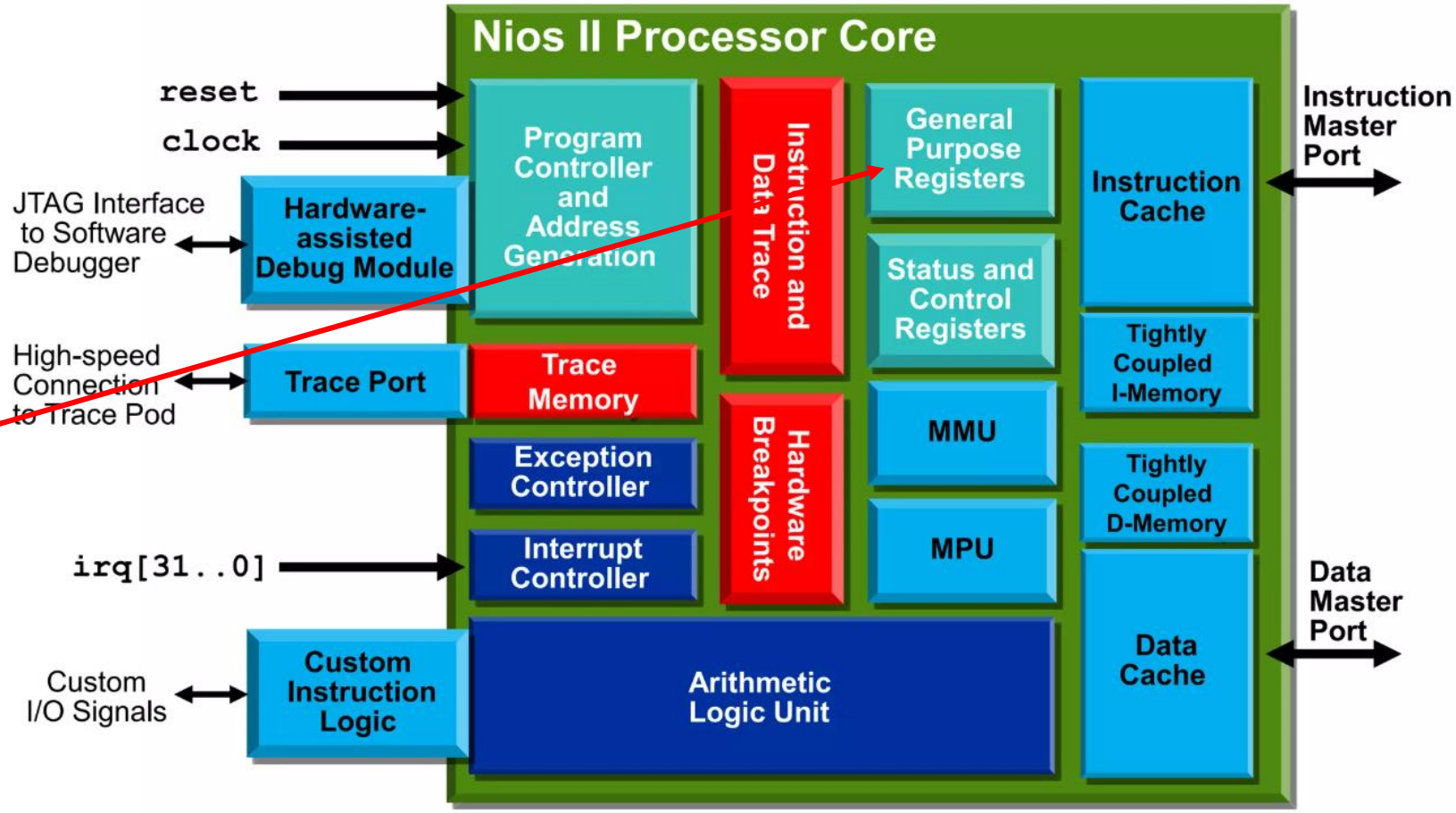
- **Data Cache**: il s'agit d'une mémoire rapide qui stocke temporairement les données lues ou écrites dans la mémoire principale
- Objectif: réduire le temps d'accès aux données




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

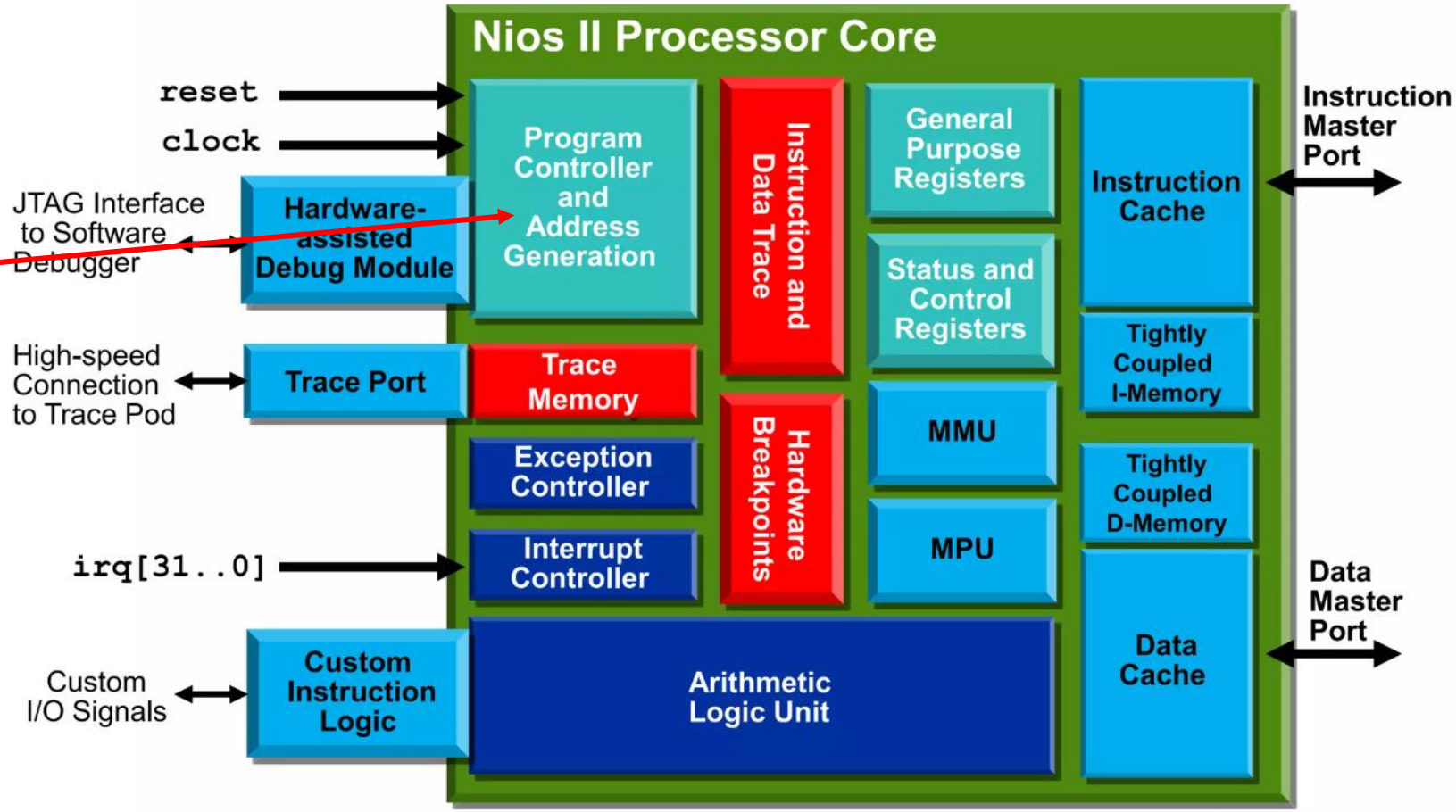
- **Status and control registres**: ces registres servent dans les opérations arithmétiques et logiques (**32 registres généraux** (R0 à R31) dans le NIOS 2)




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

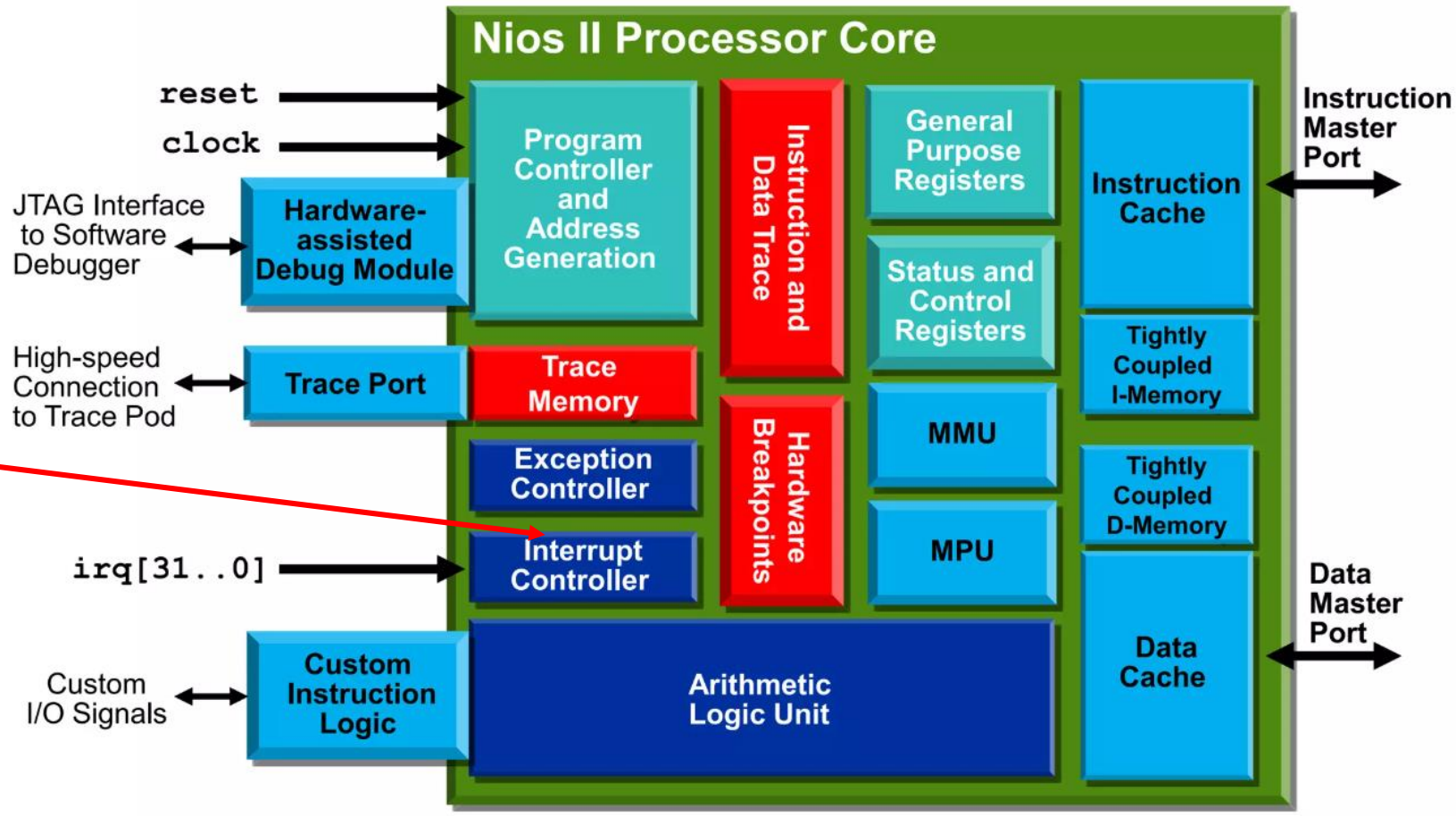
- **Program controller and address generation** : responsable de **l'exécution séquentielle** des instructions et de la gestion du **flux du programme**.




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

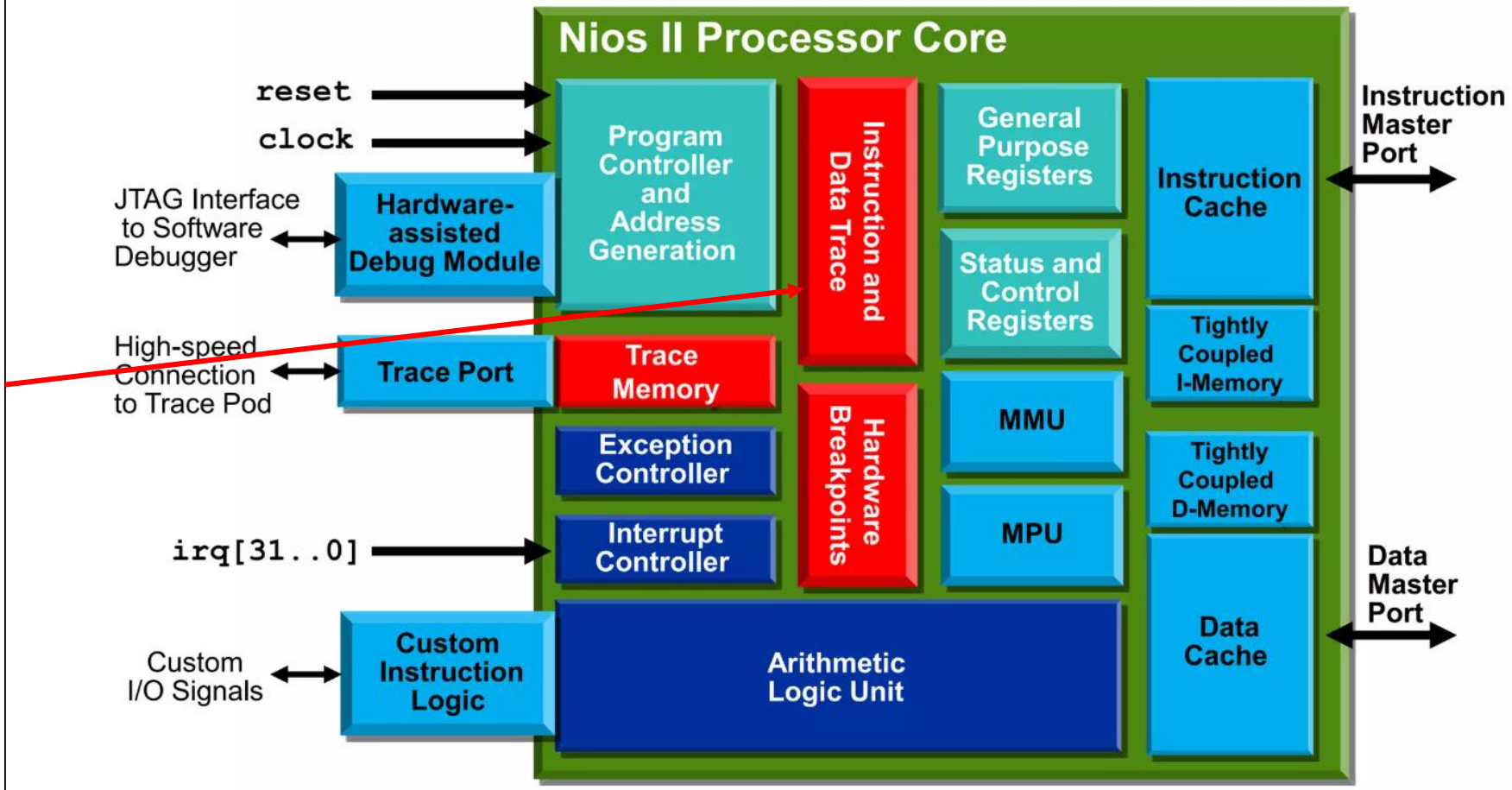
- **Exception controller and interrupt controller:** sont des mécanismes essentiels pour gérer les événements imprévus ou asynchrones qui surviennent pendant l'exécution d'un programme **(32 interruptions sont prisent en charge)**




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

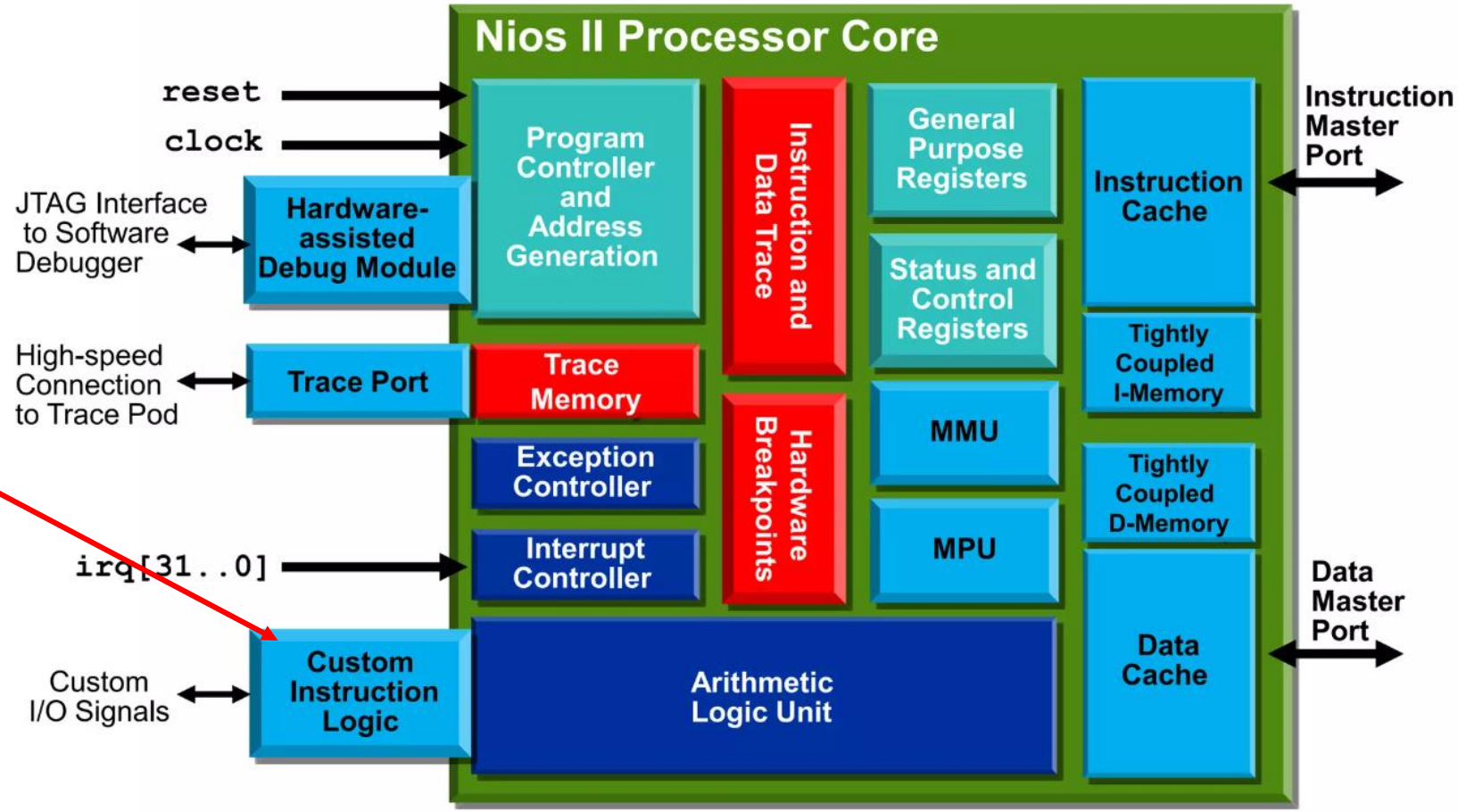
- **Instruction and data trace**: sont des mécanismes qui permettent de stocker les instructions exécutées et les données obtenues dans une mémoire (trace memory). Cela permet de suivre le flux d'exécution du programme




*Source: Intel (Altera Nios II Architecture)*

# Le NIOS II Embedded Processor architecture

- **Custom instruction logique:** sont fonctionnalités qui permettent d'ajouter des instructions spécifiques personnalisées (autres que les autres 53 instructions proposées par Intel) pour accélérer l'exécution
- (Exemple XOR avec décalage)



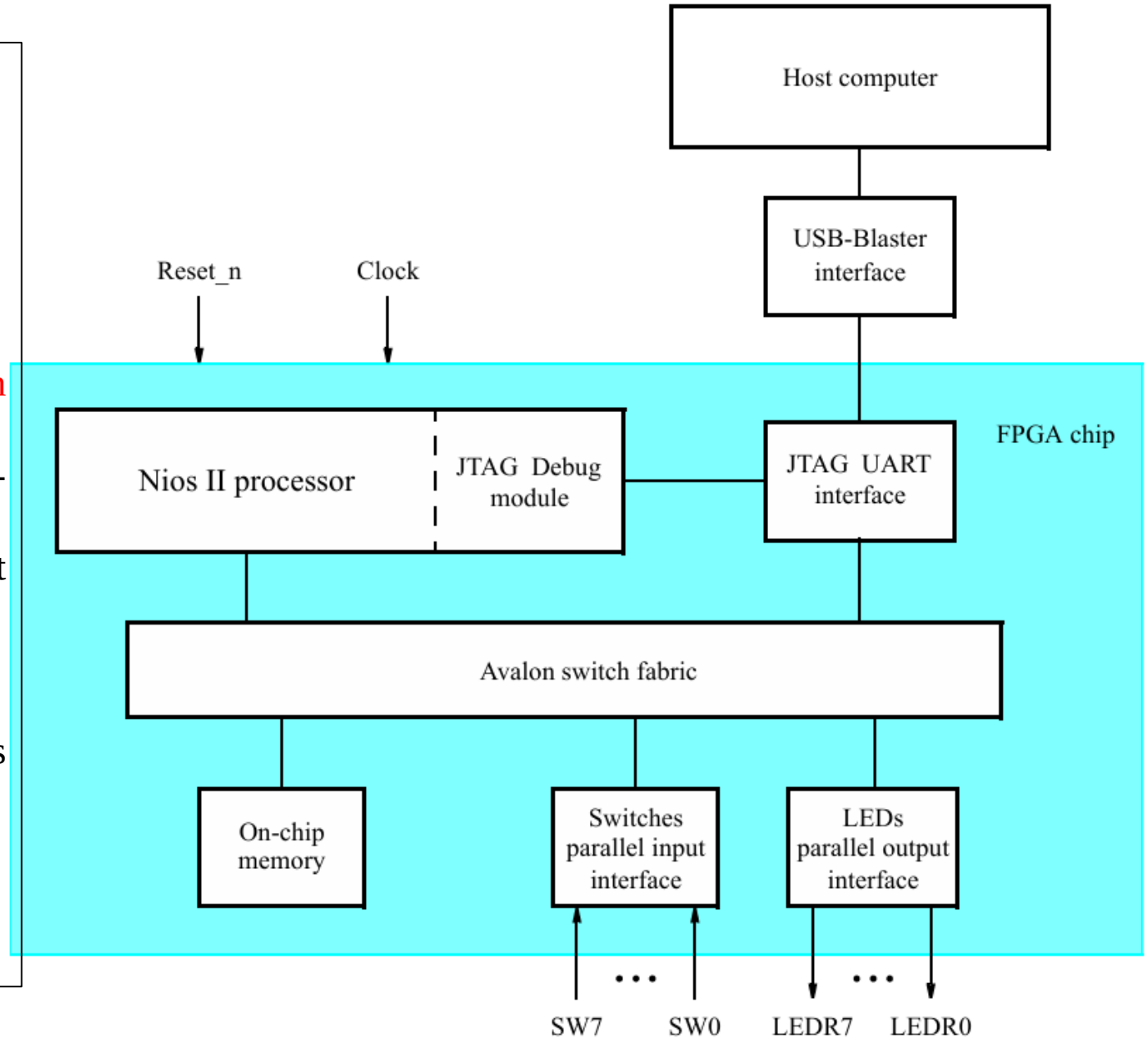

*Source: Intel (Altera Nios II Architecture)*

# Application:

# Conception et implémentation d'un SoC sur une carte FPGA (DE2-115)

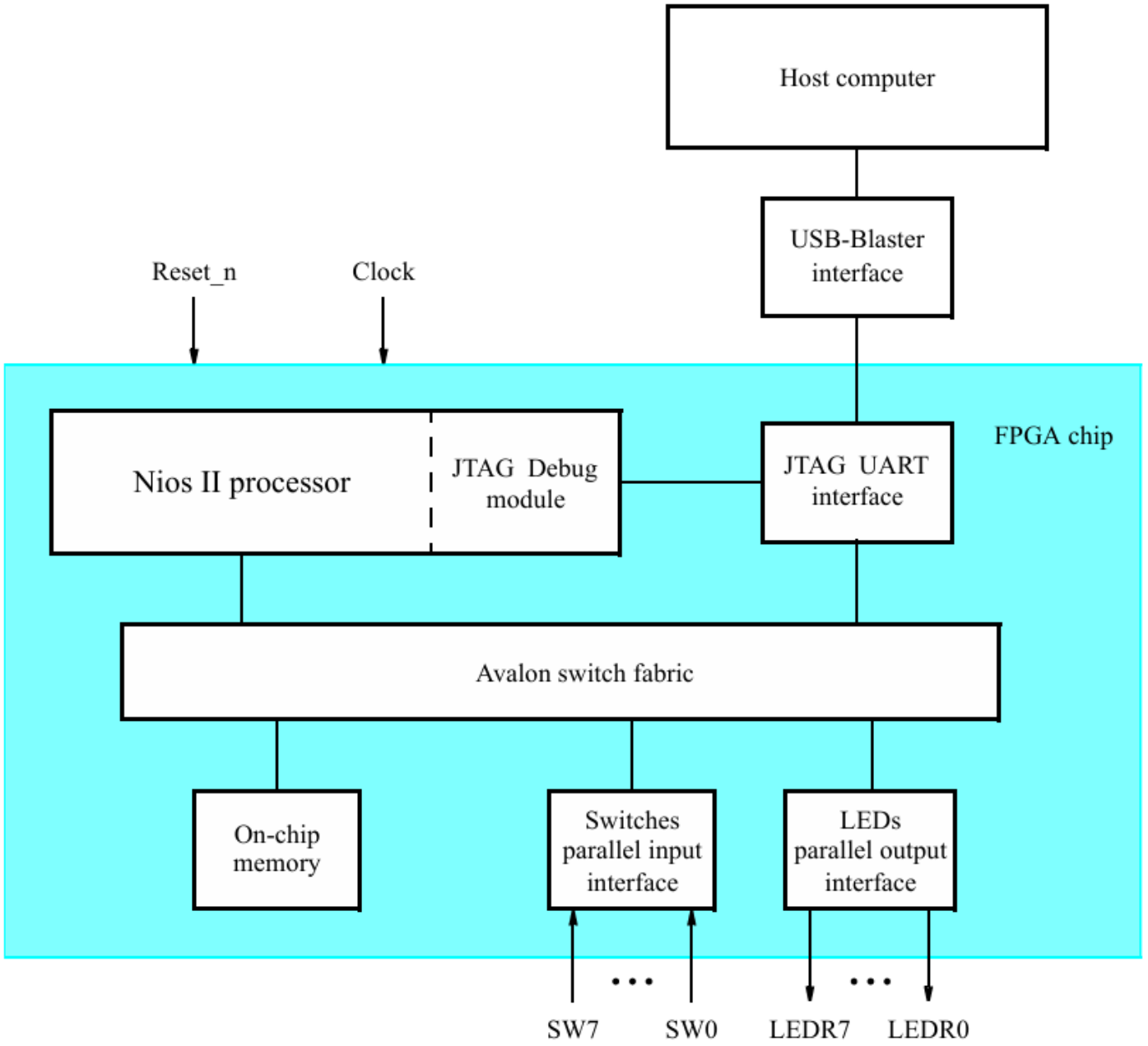
# Description du SoC

- Le SoC est constitué:
- D'un Processeur Soft Nios II
- D'une mémoire de type RAM
- D'un système d'interconnexion appelé Avalon Switch fabric (un bus de communication flexible (Master-Slave) propre à Intel (Altera) permettant d'interconnecter plusieurs périphériques
- Des Switches d'entrée (inputs) et des LEDs de sorties (output)
- Voir slide suivant….



# Description du SoC

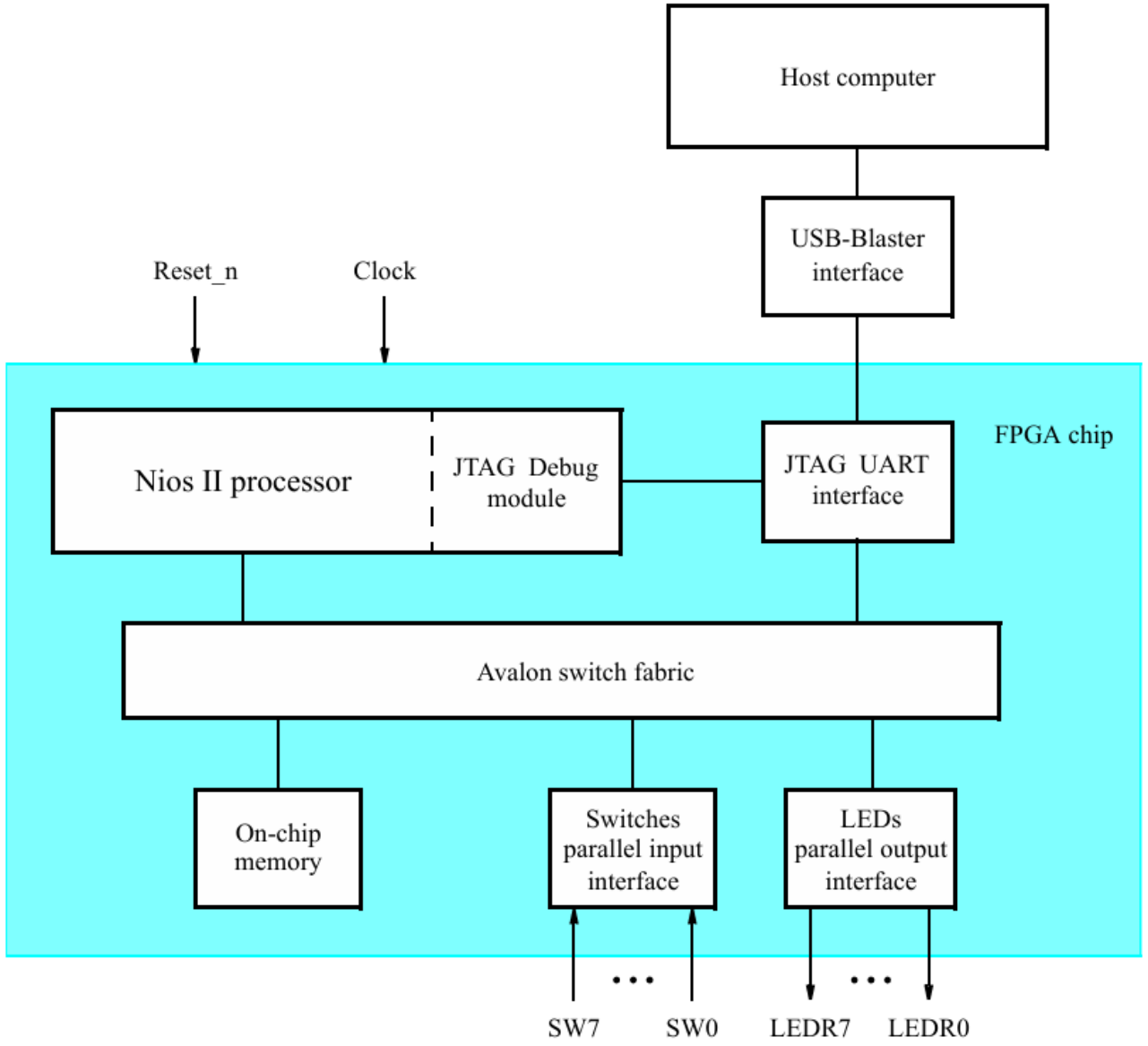
- Le SoC est constitué:
- D'une interface JTAG UART permettant de connecter la carte FPGA à l'ordinateur
- Un module JTAG Debug Module permettant d'interconnecter le processeur Nios II avec le PC pour des tâches de débogage



# Description du fonctionnement SoC

Huit commutateurs sur la carte DE2-SoC, SW7- SW0, sont utilisés pour allumer ou éteindre huit LED, LEDR7 - 0.

Pour obtenir l'opération souhaitée, le motif à huit bits correspondant à l'état des commutateurs doit être envoyé au port de sortie pour activer les LED. Cela sera fait en faisant exécuter au processeur Nios II un programme stocké dans la mémoire intégrée. Un fonctionnement continu est nécessaire, de sorte que lorsque les commutateurs sont basculés, les lumières changent en conséquence.

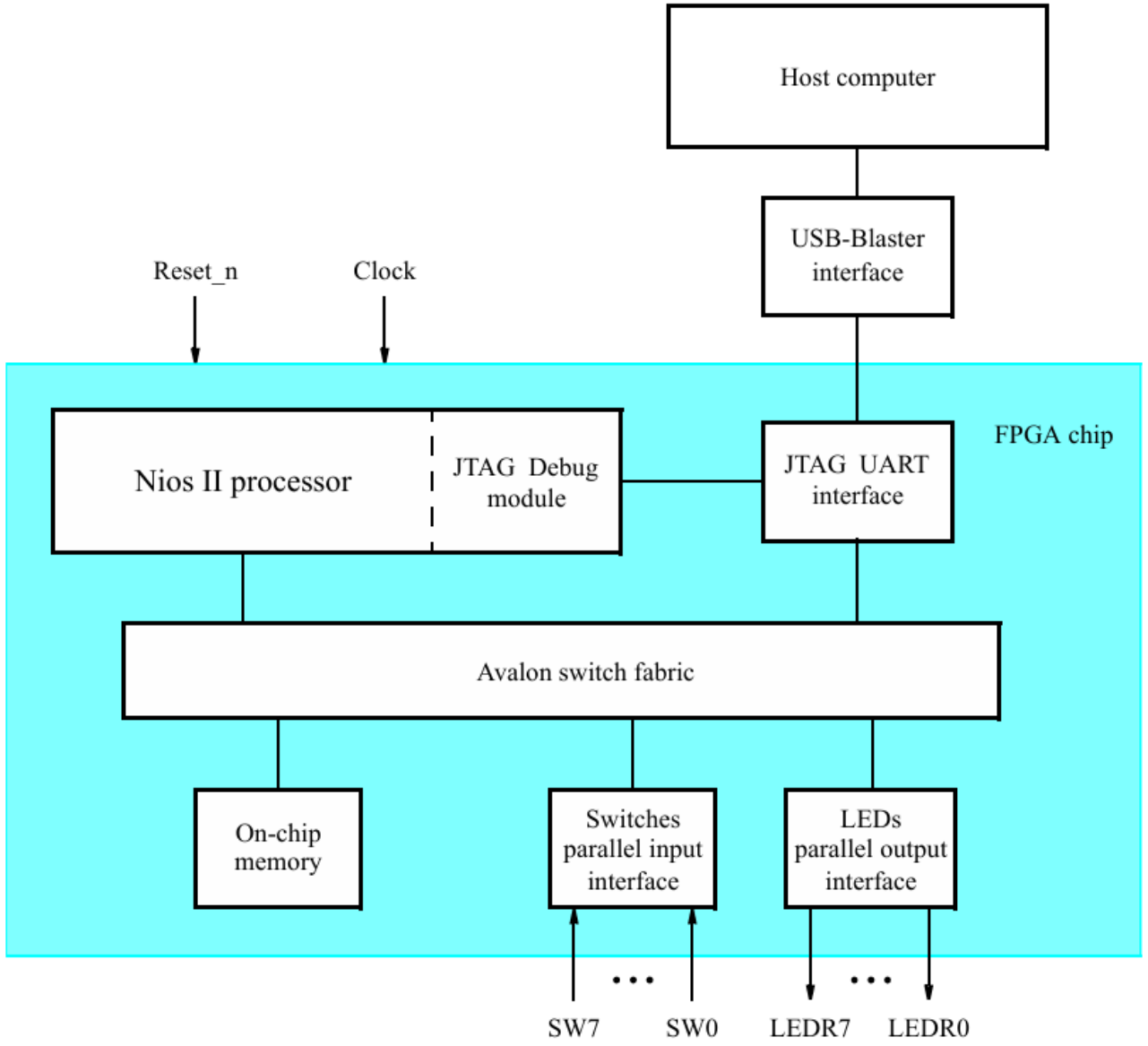


# Implémentation du SoC

Pour implémenter le SoC on va utiliser l'outil Plateform Designer de Intel

Via la plateforme, il faut instancier les éléments suivants:

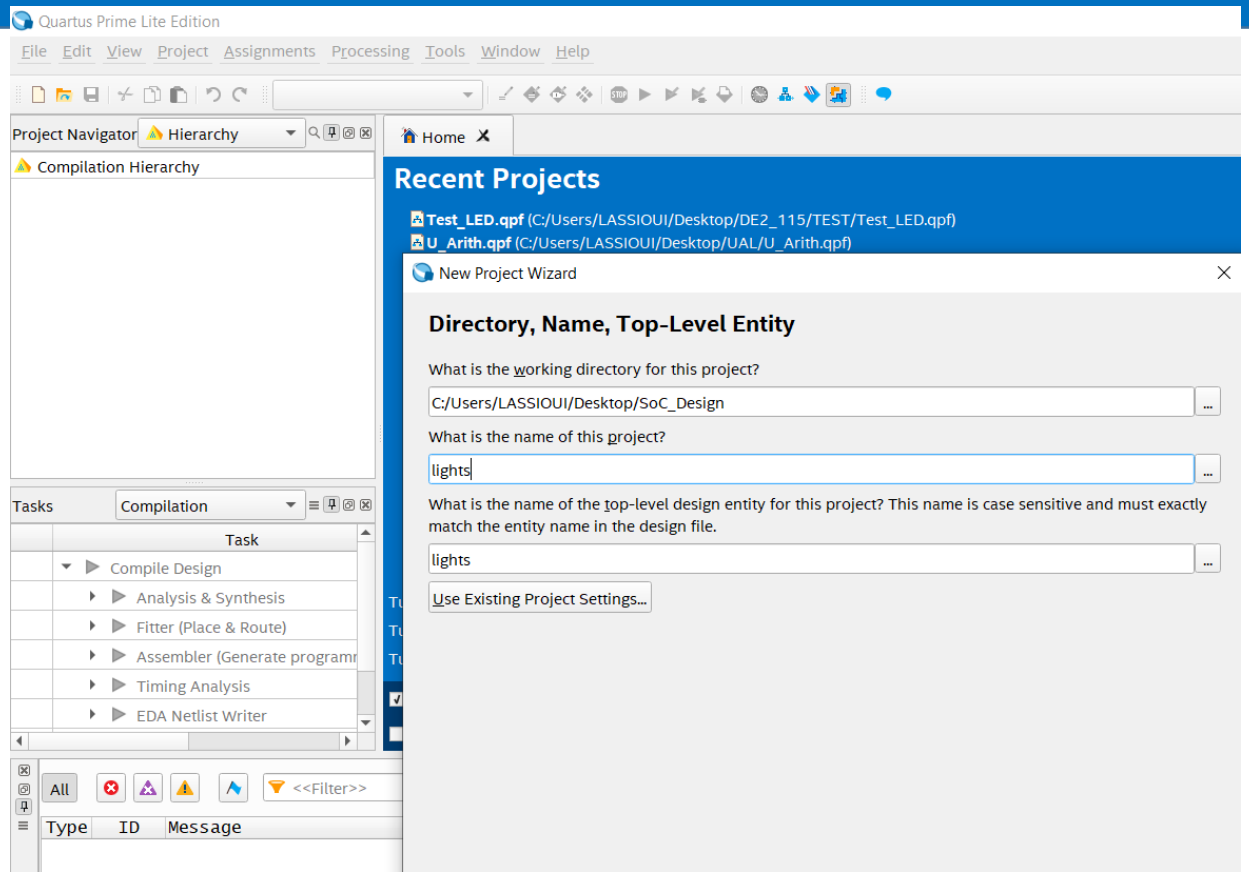
- ❑ Le processeur Nios II
- ❑ Une mémoire de 4K-Byte
- ❑ Deux interfaces parallèles d'entrées/sorties
- ❑ Un module JTAG de communication avec le PC



# Implémentation du SoC

**Étape 1:** Création d'un projet dans Quartus Prime

**Étape 2:** Choix de la famille du circuit FPGA disponible sur le DE2-115 board

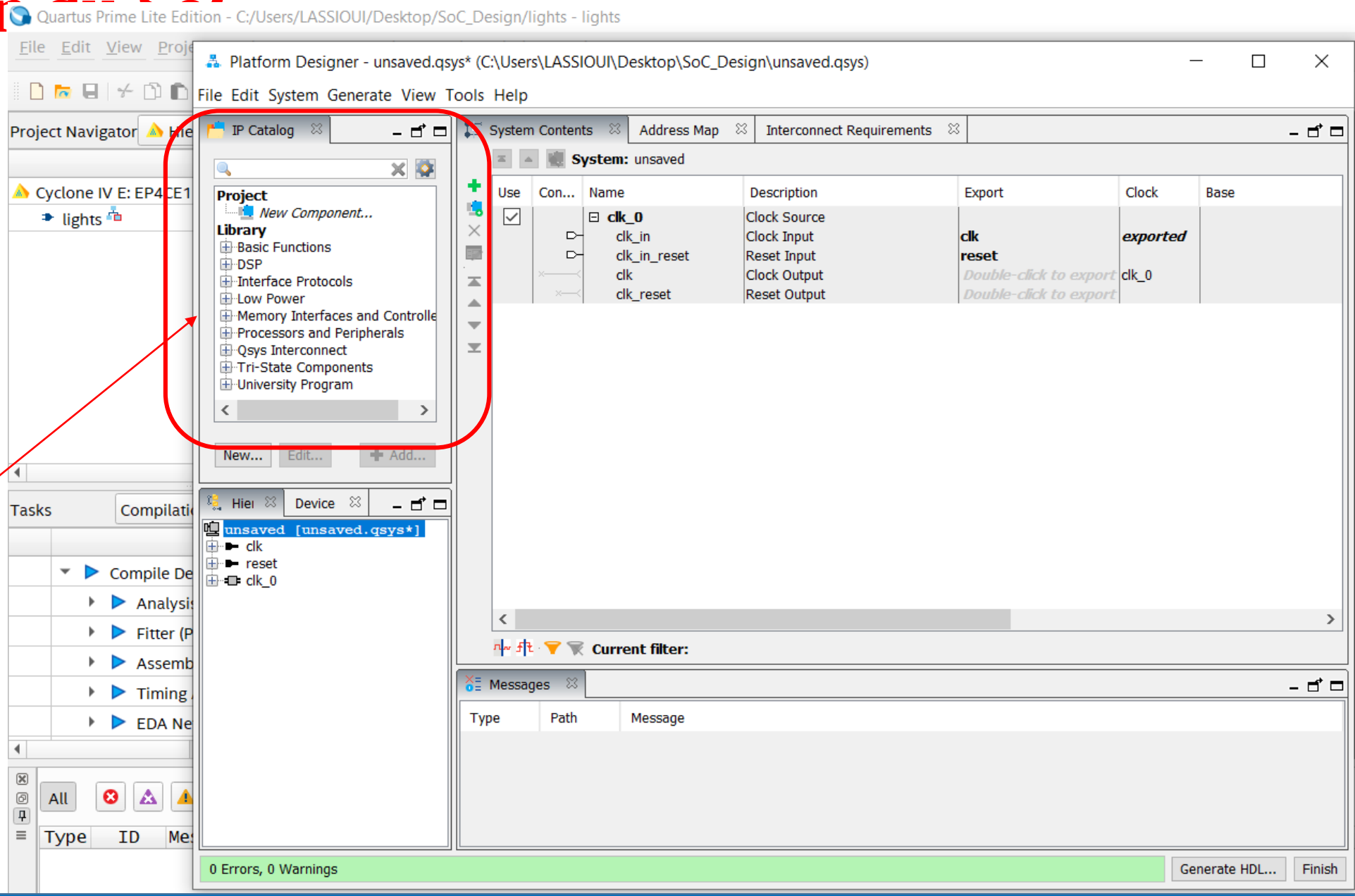



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| EP4CE115F23C8L | 1.0V | 114480 | 281 | 281 | 3981312 | 532 | 4 |
| EP4CE115F23C9L | 1.0V | 114480 | 281 | 281 | 3981312 | 532 | 4 |
| EP4CE115F23I7 | 1.2V | 114480 | 281 | 281 | 3981312 | 532 | 4 |
| EP4CE115F23I8L | 1.0V | 114480 | 281 | 281 | 3981312 | 532 | 4 |
| EP4CE115F29C7 | 1.2V | 114480 | 529 | 529 | 3981312 | 532 | 4 |
| EP4CE115F29C8 | 1.2V | 114480 | 529 | 529 | 3981312 | 532 | 4 |
| EP4CE115F29C8L | 1.0V | 114480 | 529 | 529 | 3981312 | 532 | 4 |
| EP4CE115F29C9L | 1.0V | 114480 | 529 | 529 | 3981312 | 532 | 4 |
| EP4CE115F29I7 | 1.2V | 114480 | 529 | 529 | 3981312 | 532 | 4 |

# Implémentation du SoC

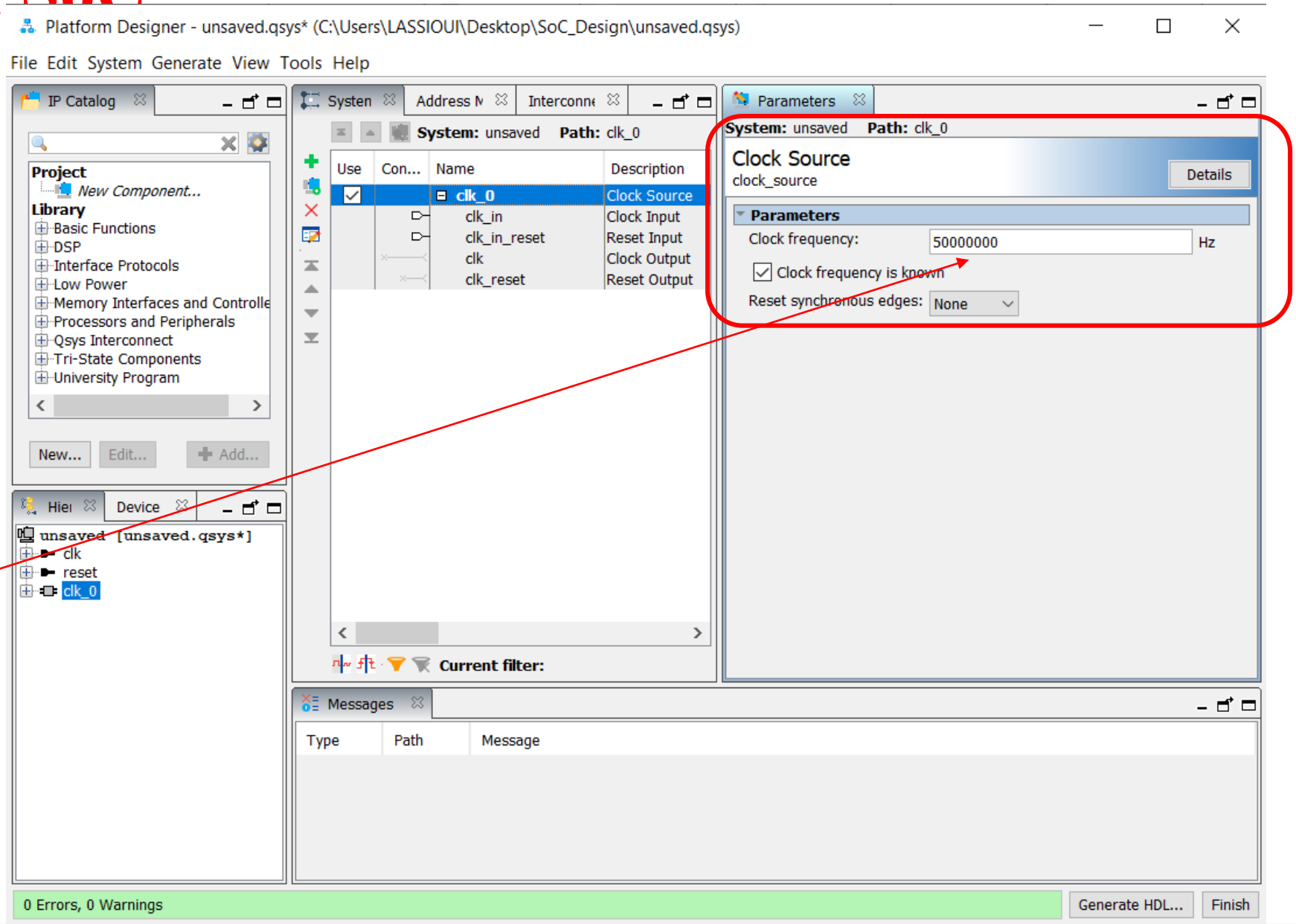
**Étape 3:** Aller vers Tools et lancer l'outil Plateform Designer

**La fenêtre IP Catalag va nous permettre d'ajouter les blocs IP à notre conception**



# Implémentation du SoC

L'exécution du SoC est cadencer par un signal d'horloge clk_0. Dans notre cas, on va utiliser le signal d'horloge fourni par l'oscillateur interne de la carte DE2-115 (50MHz)

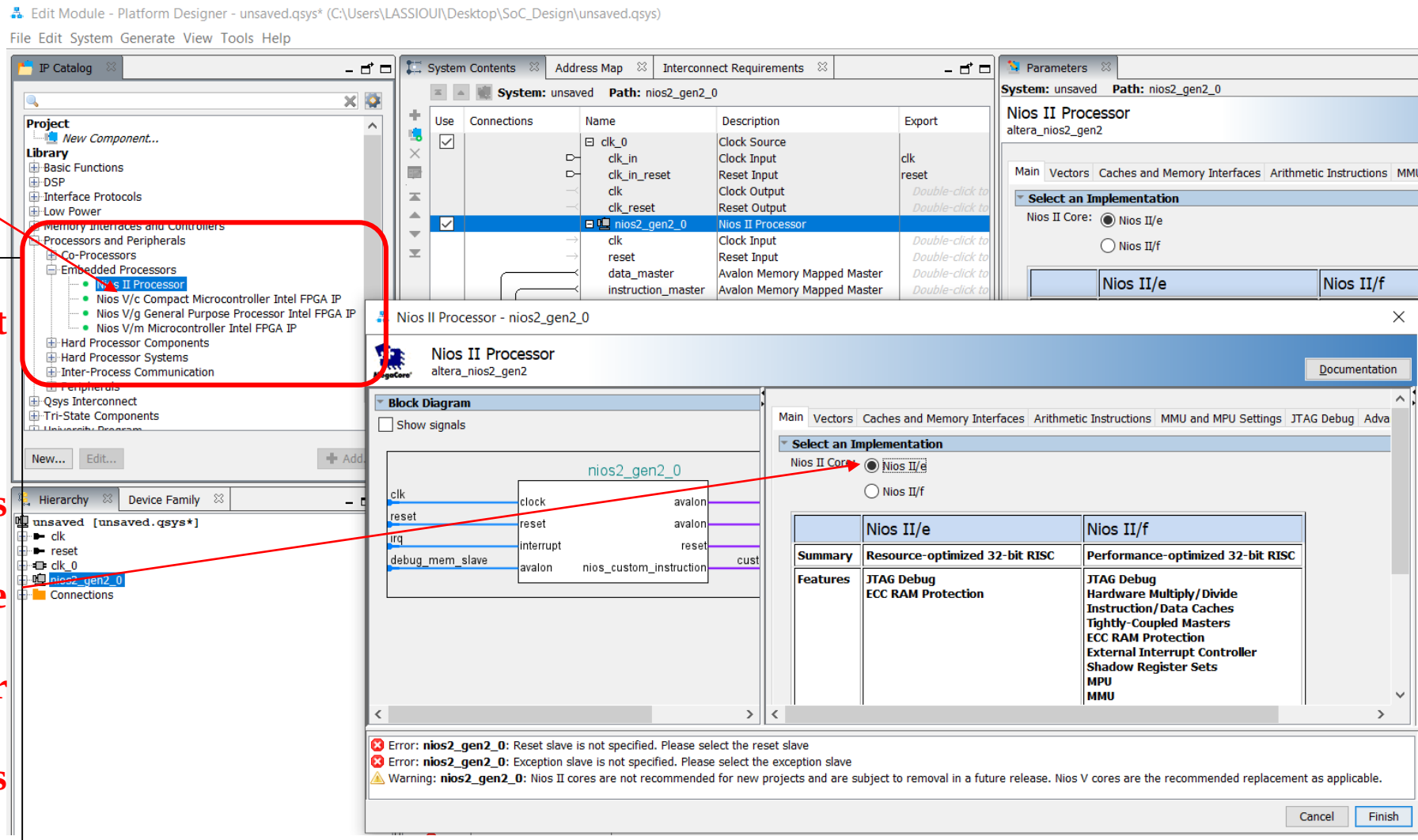


# Implémentation du SoC

**Étape 4:** Ajouter le Processeur Nios II

**Choisir la version Nios II/e (on peut l'utiliser sans licence)**

**Le processeur possède deux entrées Reset and Interrupt (quand l'une de ces entrées est activée, le processeur commence l'exécution à des adresses spécifiques de la mémoire**

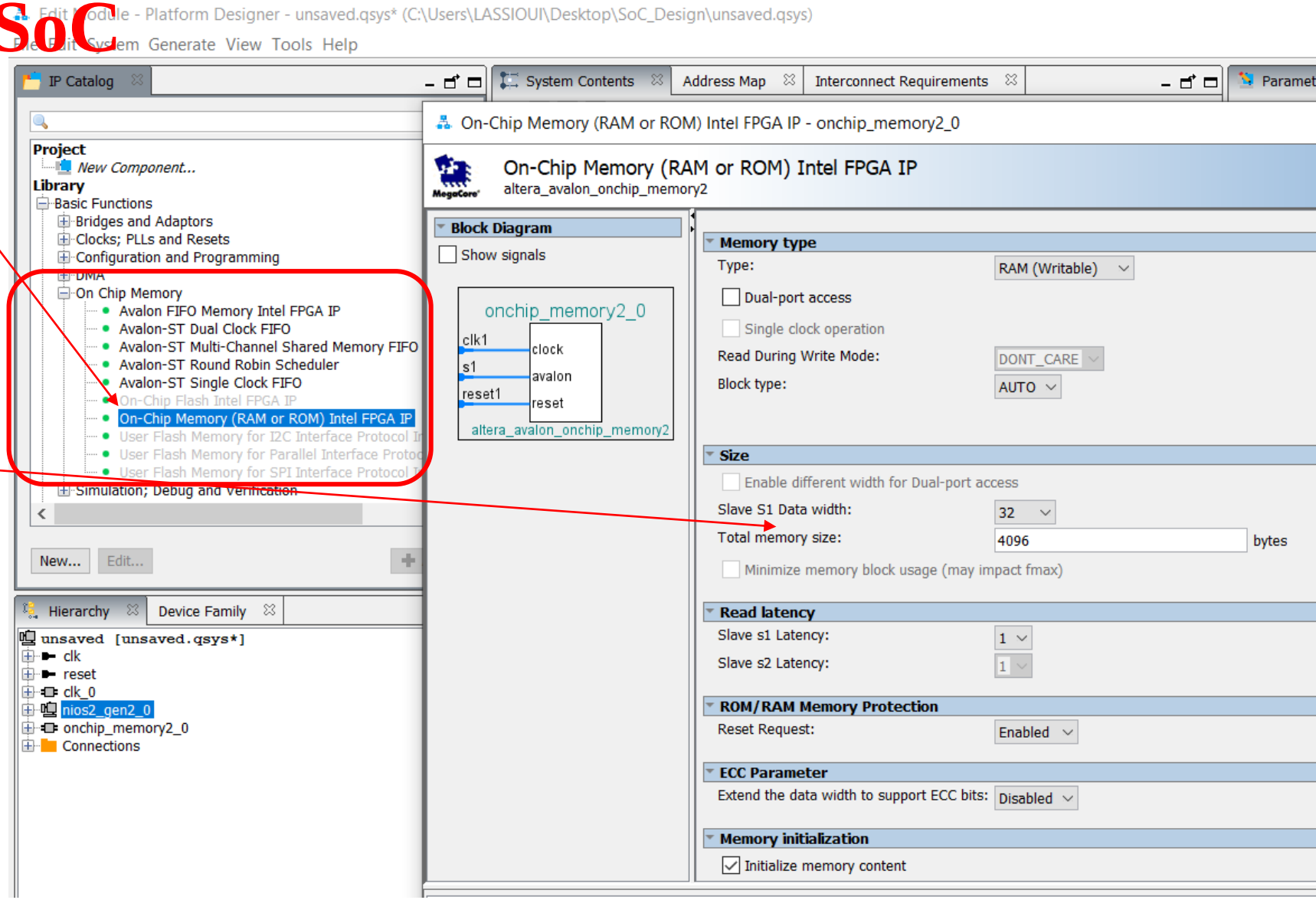


# Implémentation du SoC

**Étape 5:** Ajouter la mémoire (basic functions : On-Chip memory

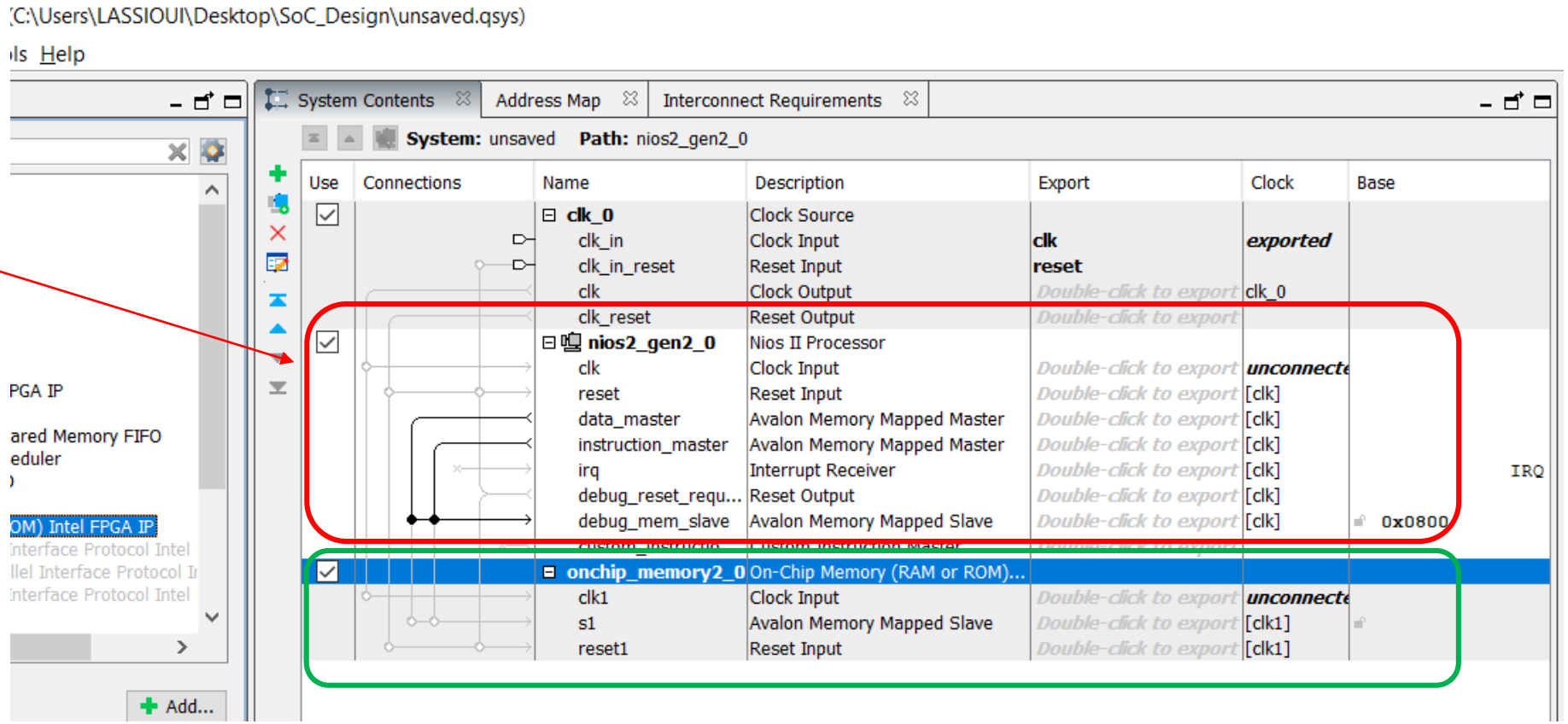
**Assurer que le bus est de 32 bits et la taille est de 4K-Bytes**

**La mémoire est de type RAM**



# Implémentation du SoC

**Observer que la mémoire et le processeur ont été ajouté à la conception mais aucune connexion n'existe entre les deux**

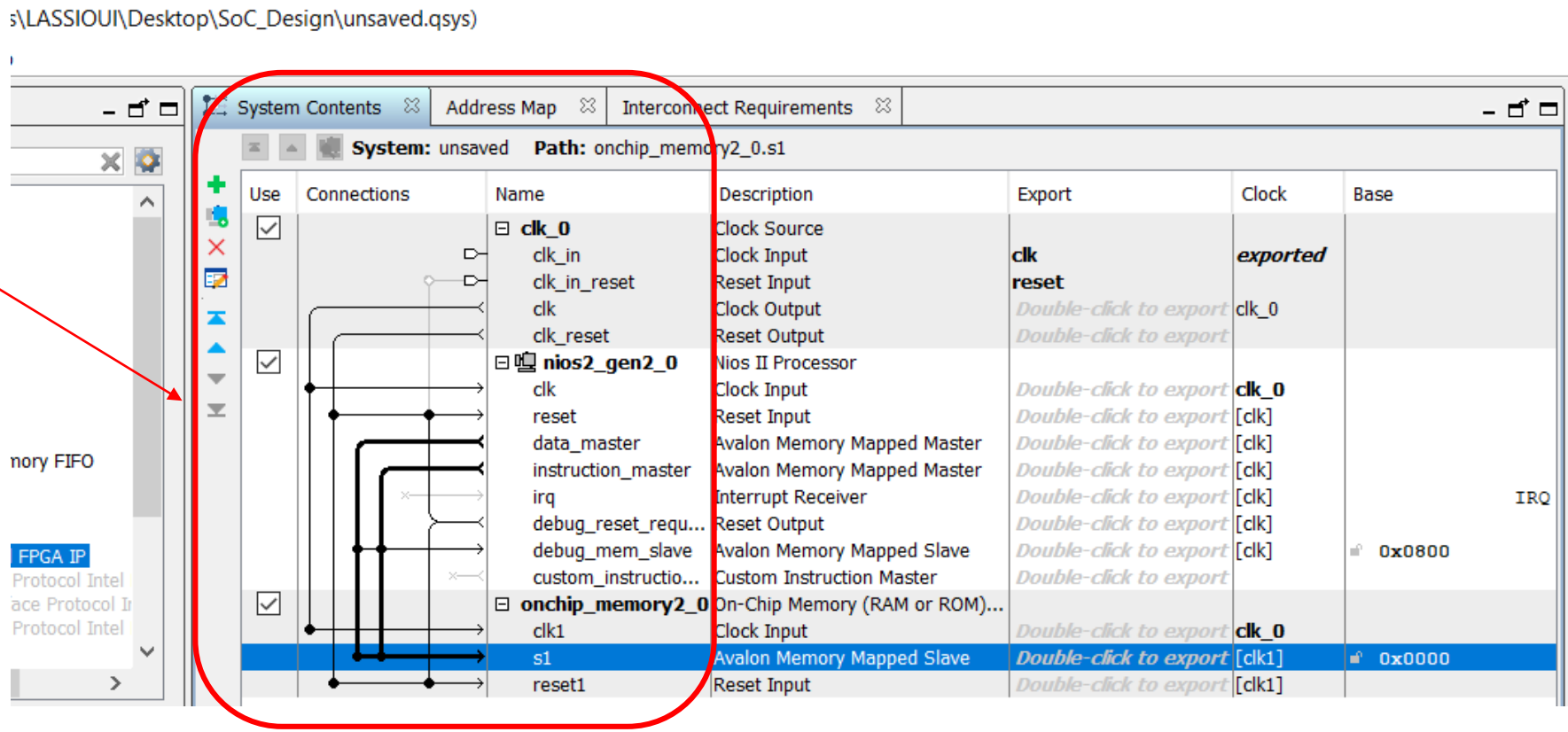



**Les connexions à faire:**

- ❑ **Clk out vers clk_in du processeur et de la mémoire**
- ❑ **Reset out (clk and Jtag) vers Reset in Mémoire et Nios II**
- ❑ **S1 de la mémoire vers Data_Master et Instruction_master du Nios II**

# Implémentation du SoC

La figure suivante illustre les connexions effectuées




**Les connexions à faire:**

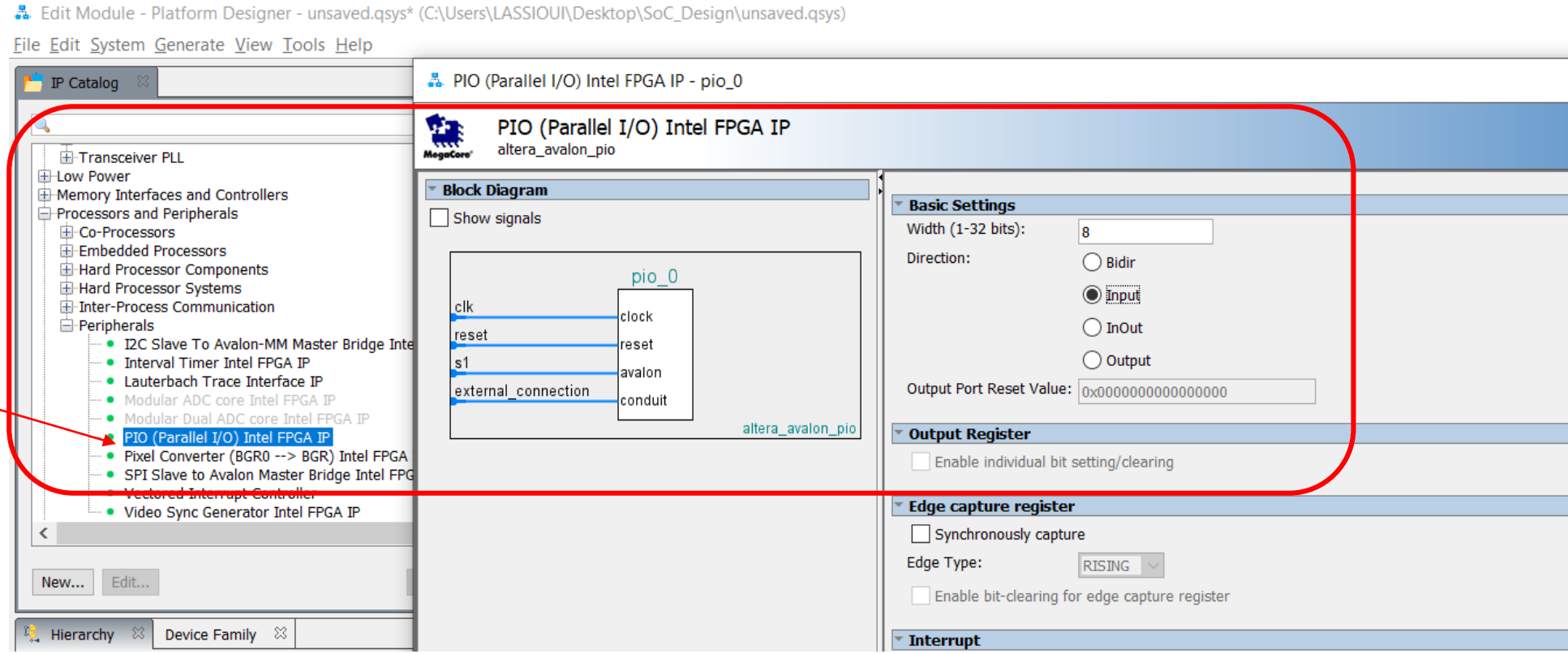
- ❑ **Clk out vers clk_in du processeur et de la mémoire**
- ❑ **Reset out (clk and Jtag) vers Reset in Mémoire et Nios II**
- ❑ **S1 de la mémoire vers Data_Master et Instruction_master du Nios II**

# Implémentation du SoC

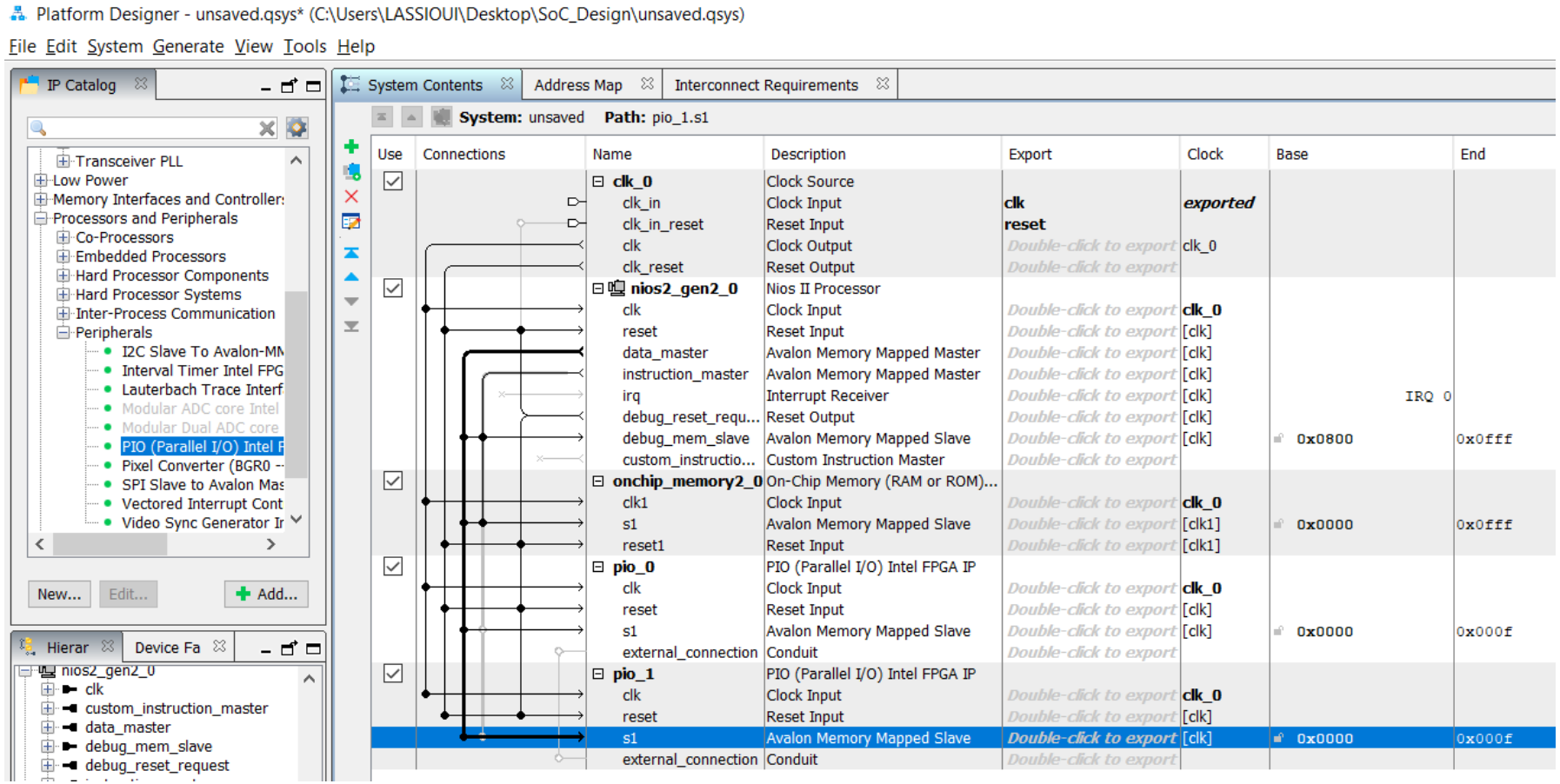
**Étape 6: Ajout des interfaces de communications**

**Choisir la direction input et la taille 8**

**De la même façon, ajouter le périphérique d'interface de sortie**



# Implémentation du SoC

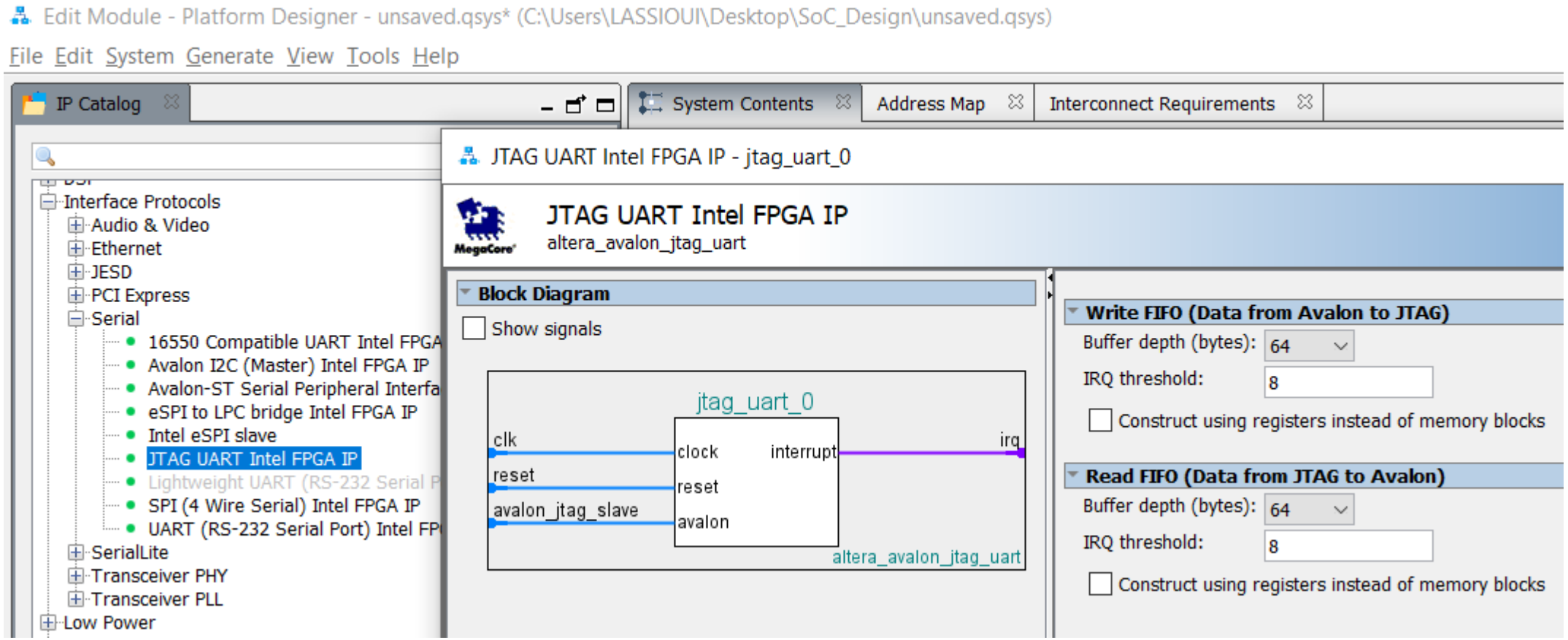



**Les connexions à faire:**

- ❑ **Clk out vers clk_in PIO**
- ❑ **Reset out (clk and Jtag) vers Reset in du PIO**
- ❑ **S1 du PIO vers Data_Master du Nios II**

# Implémentation du SoC

**Étape 7: Ajout de l'interface JTAG**



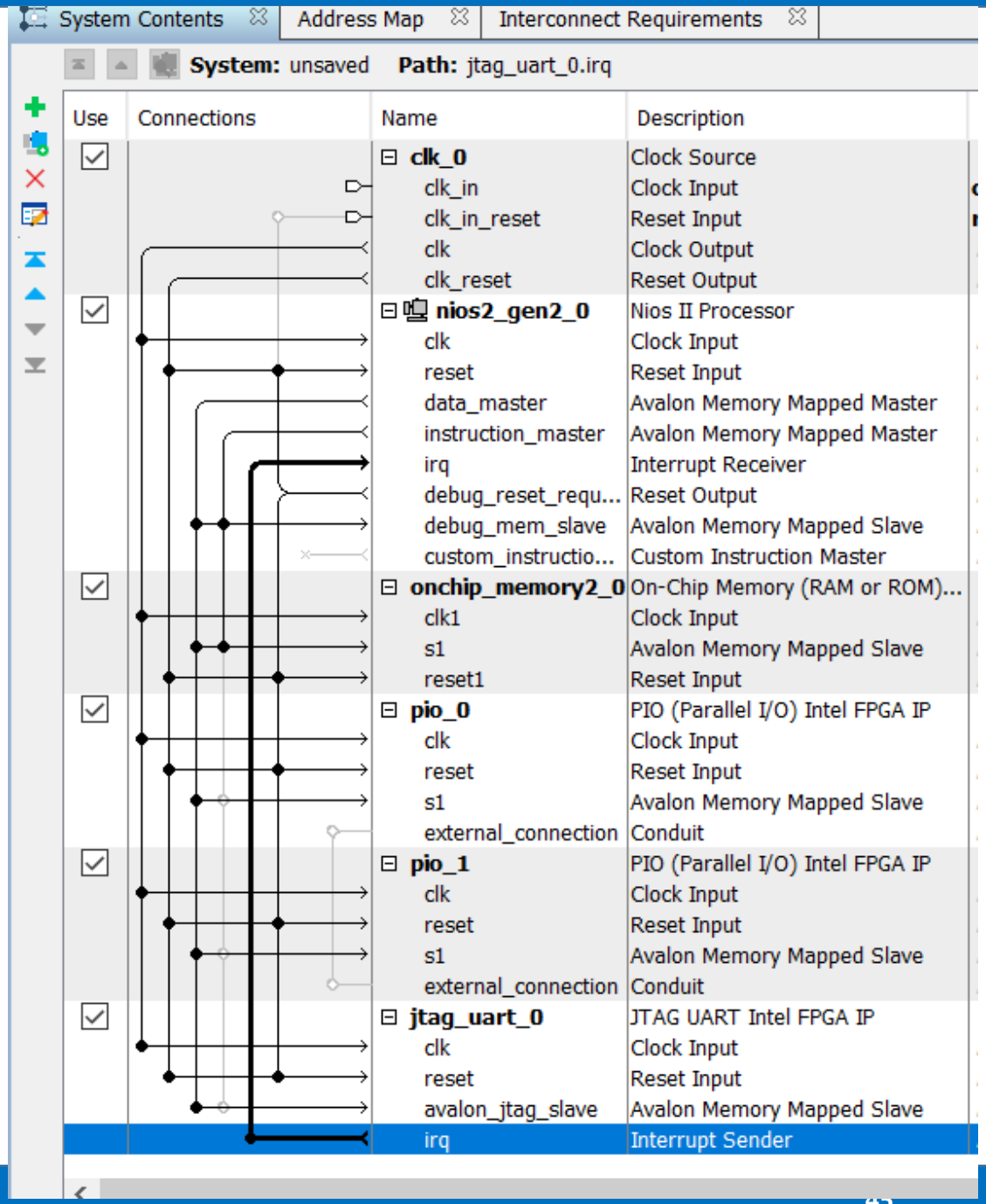
# Implémentation du SoC

**Étape 8:** **Ajouter les connexions suivantes:**

**Clk out vers clk Jtag**

**Reset out (clk et nios) vers reset JTAG**

**Irq (interrupt requets) vers Irq Nios II**

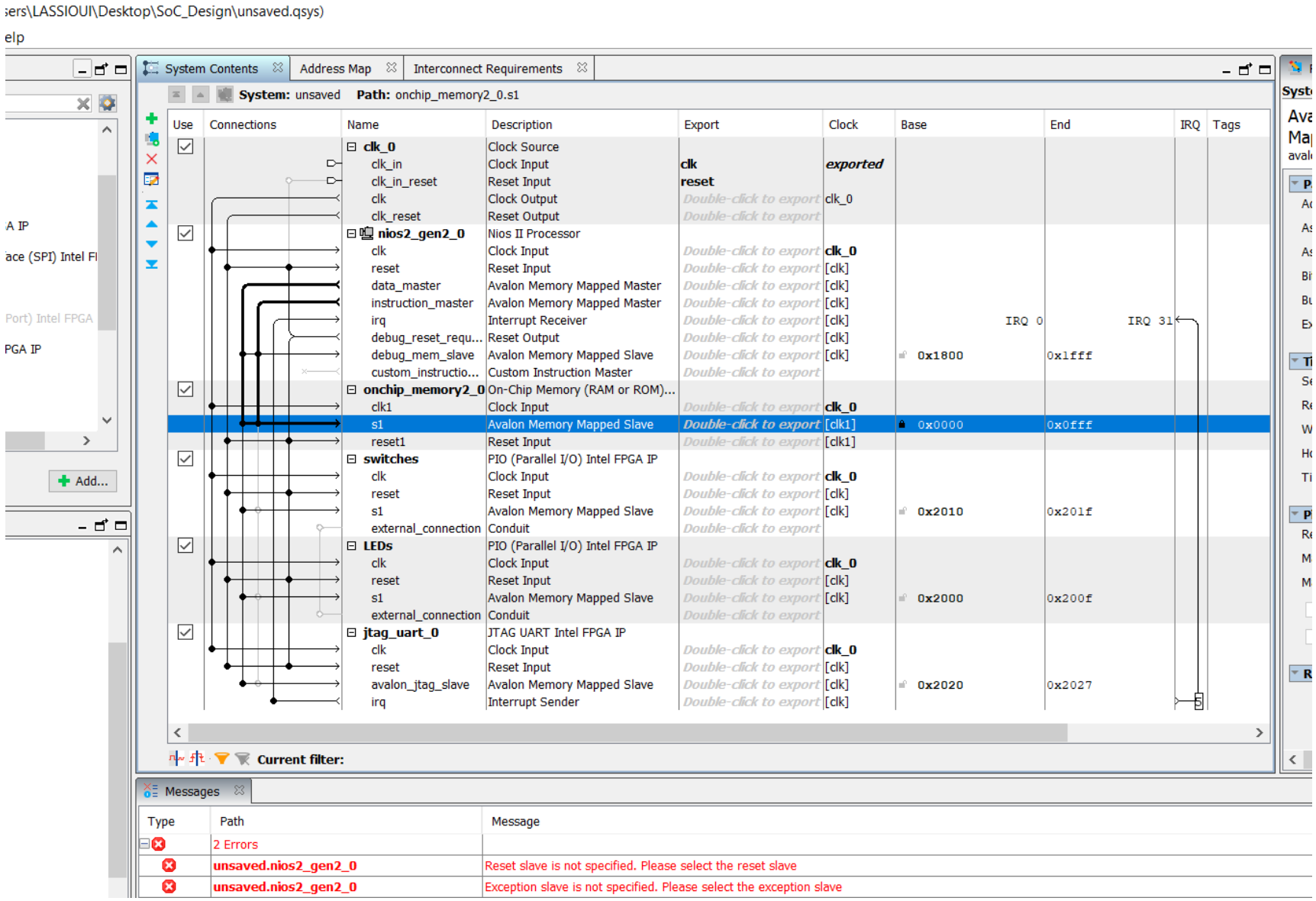


# Implémentation du SoC

**Étape 8:** l'objectif de cette étape est de spécifier une adresse de base de la mémoire

Cliquer sur On-chip memory puis choisir l'adresse de base suivante: **0x00000000**

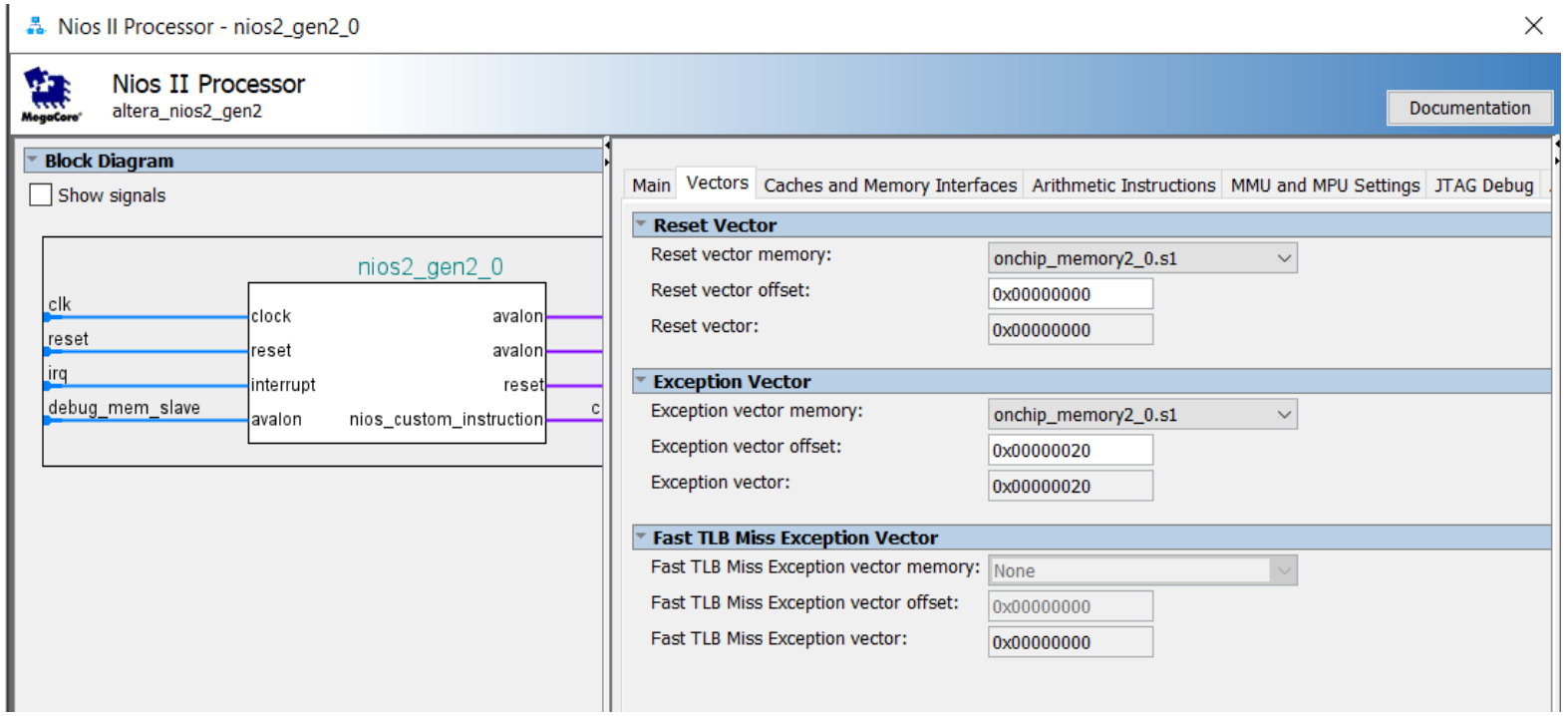
Puis allez vers **System -> Assign Base Adresses**



# Implémentation du SoC

**Étape 9: Spécifier la mémoire On-Chip Memory comme étant la mémoire de base utilisée par le NIOS II lors des opérations de Reset et d'interruption**

**Cliquez droit sur Nios II -> Edit -> Vectors comme le montre la figure suivante**



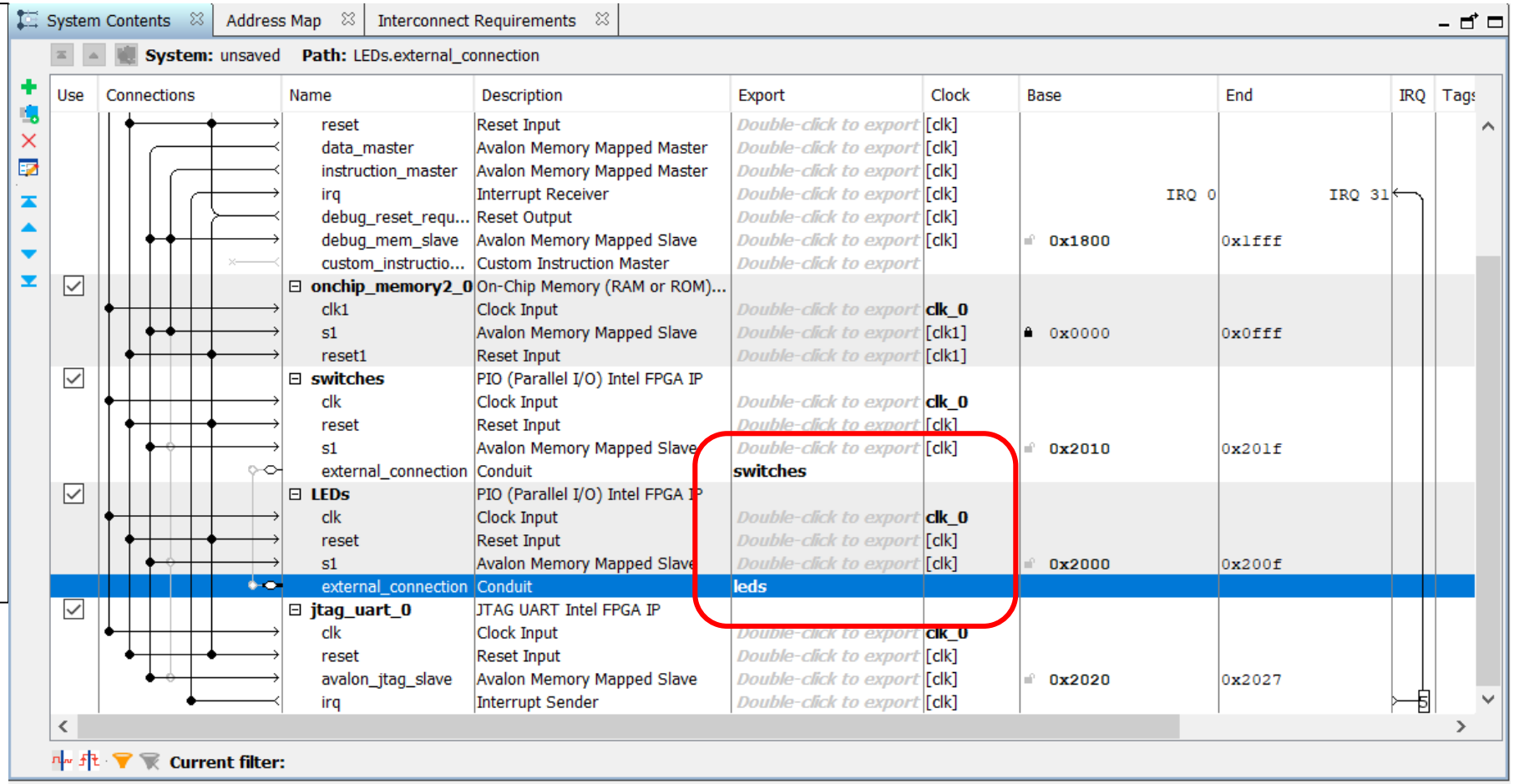

**À ce stade, tous les messages d'erreurs disparaitront**

# Implémentation du SoC

**Étape 10: Spécifier les connexions avec les périphériques externes**

**Cliquez sur LEDs puis double-clic sur export (nommer le LEDS**

**Même chose pour les switches**

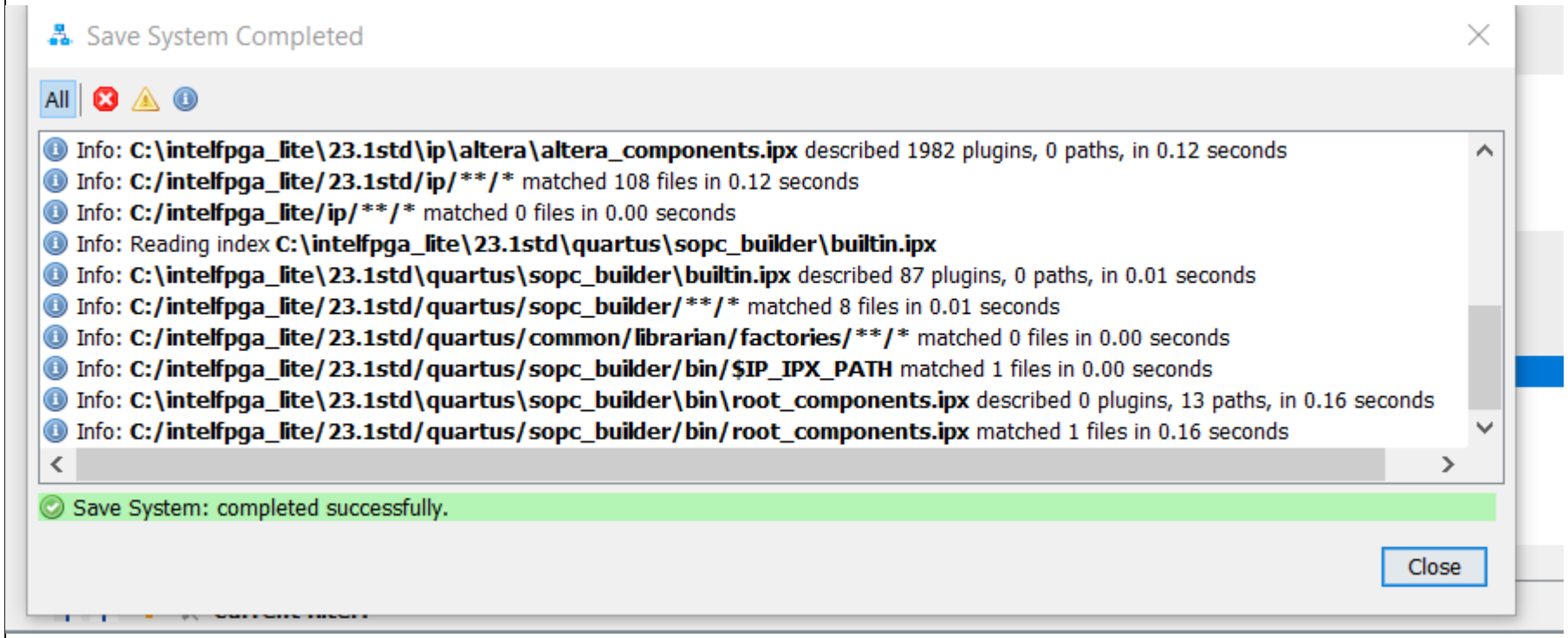


# Implémentation du SoC

**Étape 11: enregistrer le système sous le nom nios_system dans le même répertoire du projet**

**La figure suivante doit être générée**

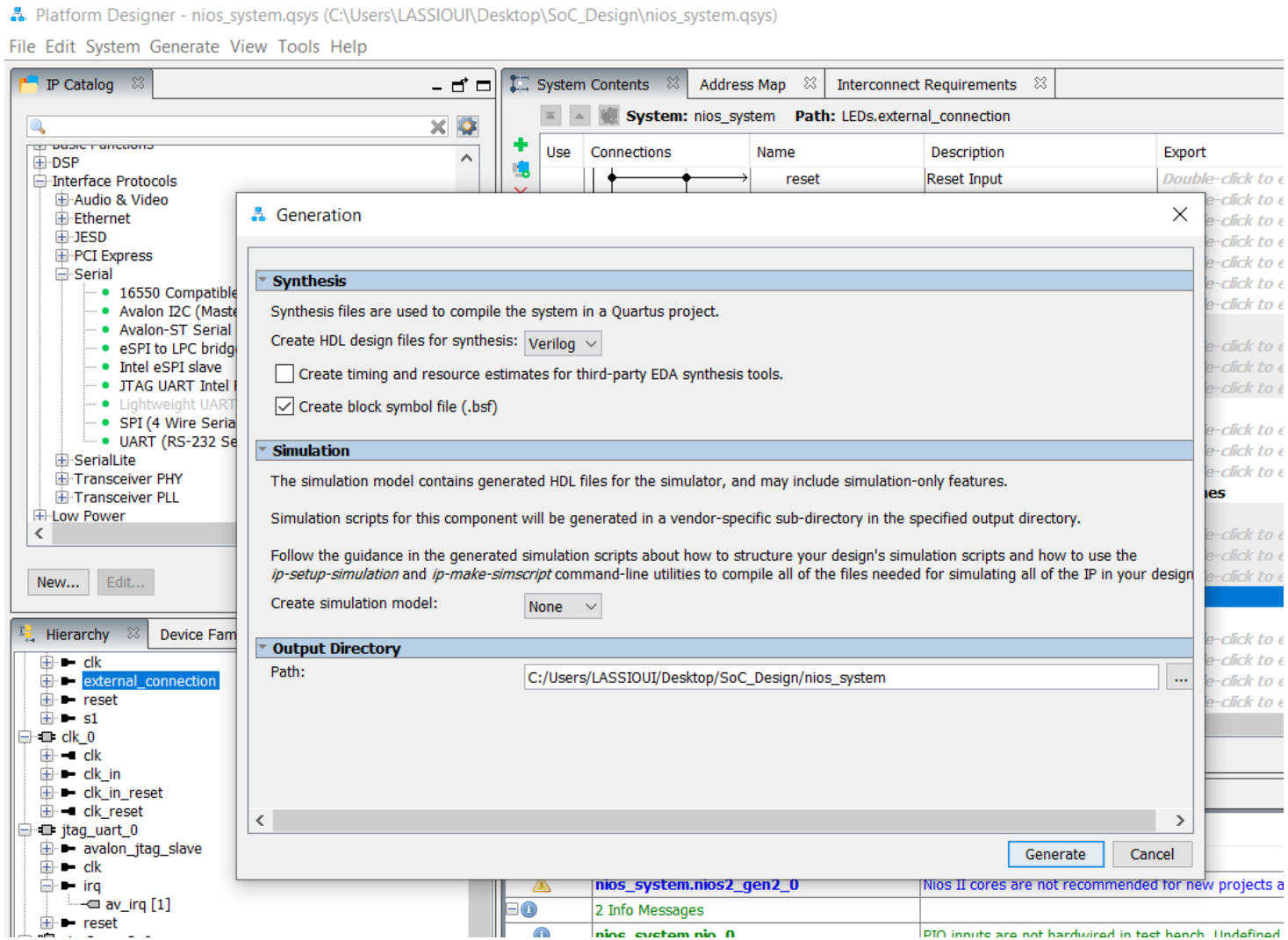
**Sinon, il faut revoir les étapes précédentes**



# Implémentation du SoC

**Étape 12: Génération du code HDL du SoC**

**Allez vers Generate -> HDL**



# Implémentation du SoC

**Intégrer le code HDL généré dans un fichier VHDL**

# Implémentation d

Le code généré est très large, ce qui nous interesse est de savoir les noms du module pour l'instancier dans le code VHDL

C:\Users\LASSIOUI\Desktop\SoC_Design\nios_system\synthesis\nios_system.v - Notepad++

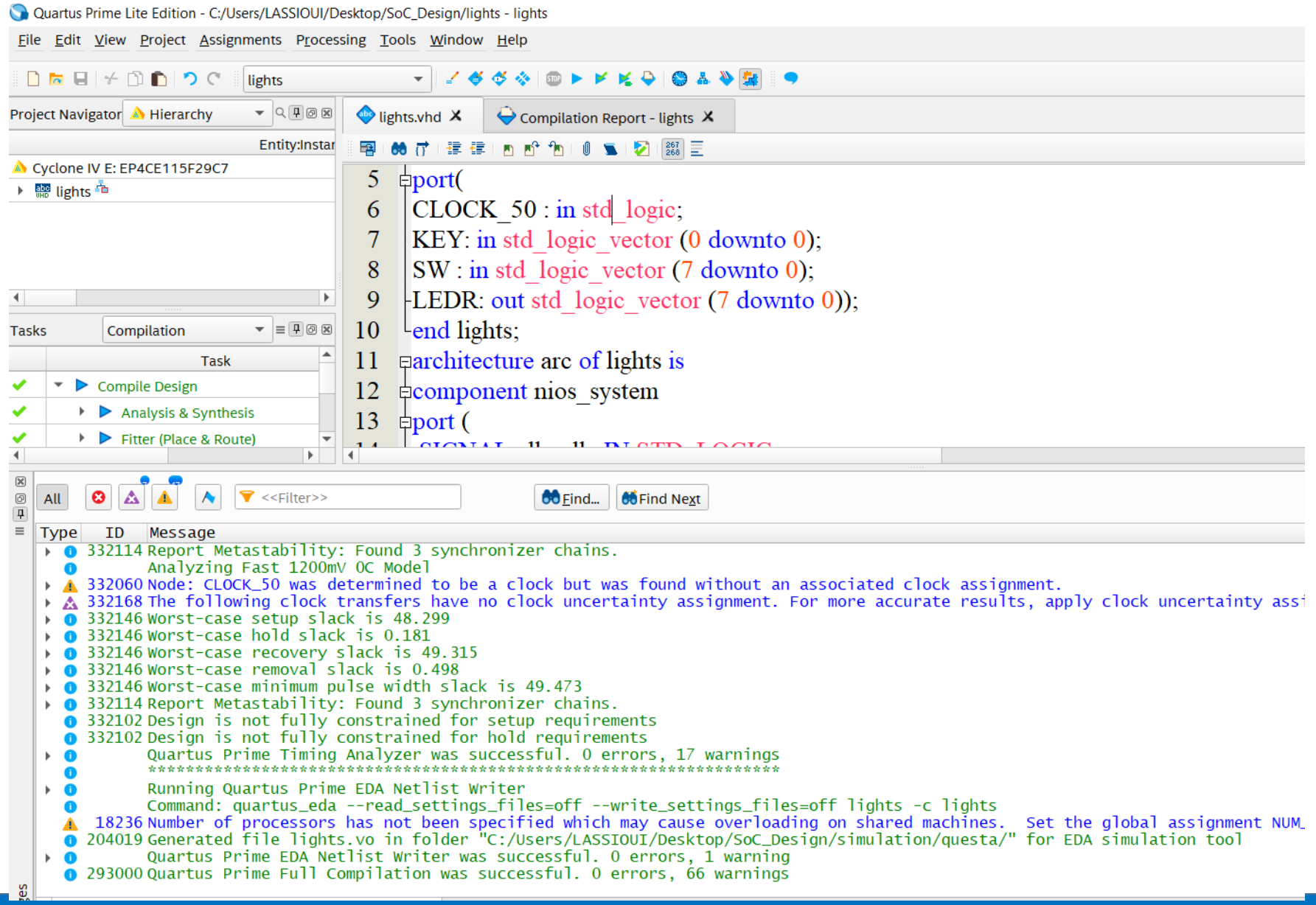
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?

and_gate.vhd | and_gate_tb.vhd | new 1 | new 2 | new 3 | new 4 | new 5 | new 6 | nios_system.v

```
1  // nios_system.v
2
3  // Generated using ACDS version 23.1 991
4
5  `timescale 1 ps / 1 ps
6  module nios_system (
7          input  wire        clk_clk,          //      clk.clk
8          output wire [7:0]  leds_export,      //     leds.export
9          input  wire        reset_reset_n,    //    reset.reset_n
10         input  wire [7:0]  switches_export   // switches.export
11     );
12
13     wire  [31:0] nios2_gen2_0_data_master_readdata;
14     wire         nios2_gen2_0_data_master_waitrequest;
15     wire         nios2_gen2_0_data_master_debugaccess;
16     wire  [13:0] nios2_gen2_0_data_master_address;
17     wire   [3:0] nios2_gen2_0_data_master_byteenable;
18     wire         nios2_gen2_0_data_master_read;
19     wire         nios2_gen2_0_data_master_write;
20     wire  [31:0] nios2_gen2_0_data_master_writedata;
21     wire  [31:0] nios2_gen2_0_instruction_master_readdata;
22     wire         nios2_gen2_0_instruction_master_waitrequest;
23     wire  [12:0] nios2_gen2_0_instruction_master_address;
24     wire         nios2_gen2_0_instruction_master_read;
25     wire         mm_interconnect_0_jtag_uart_0_avalon_jtag_slave_chi
26     wire  [31:0] mm_interconnect_0_jtag_uart_0_avalon_jtag_slave_rea
```

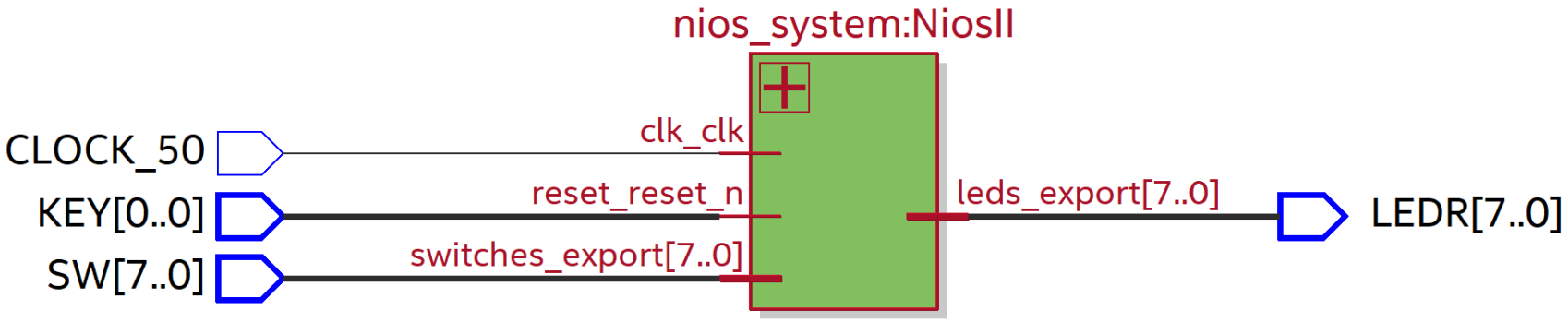
# Implémentation du SoC

Le code généré est très large, ce qui nous interesse est de savoir les noms du module pour l'instancier dans le code VHDL

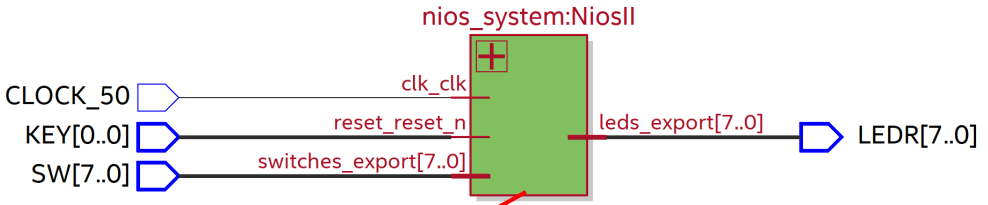


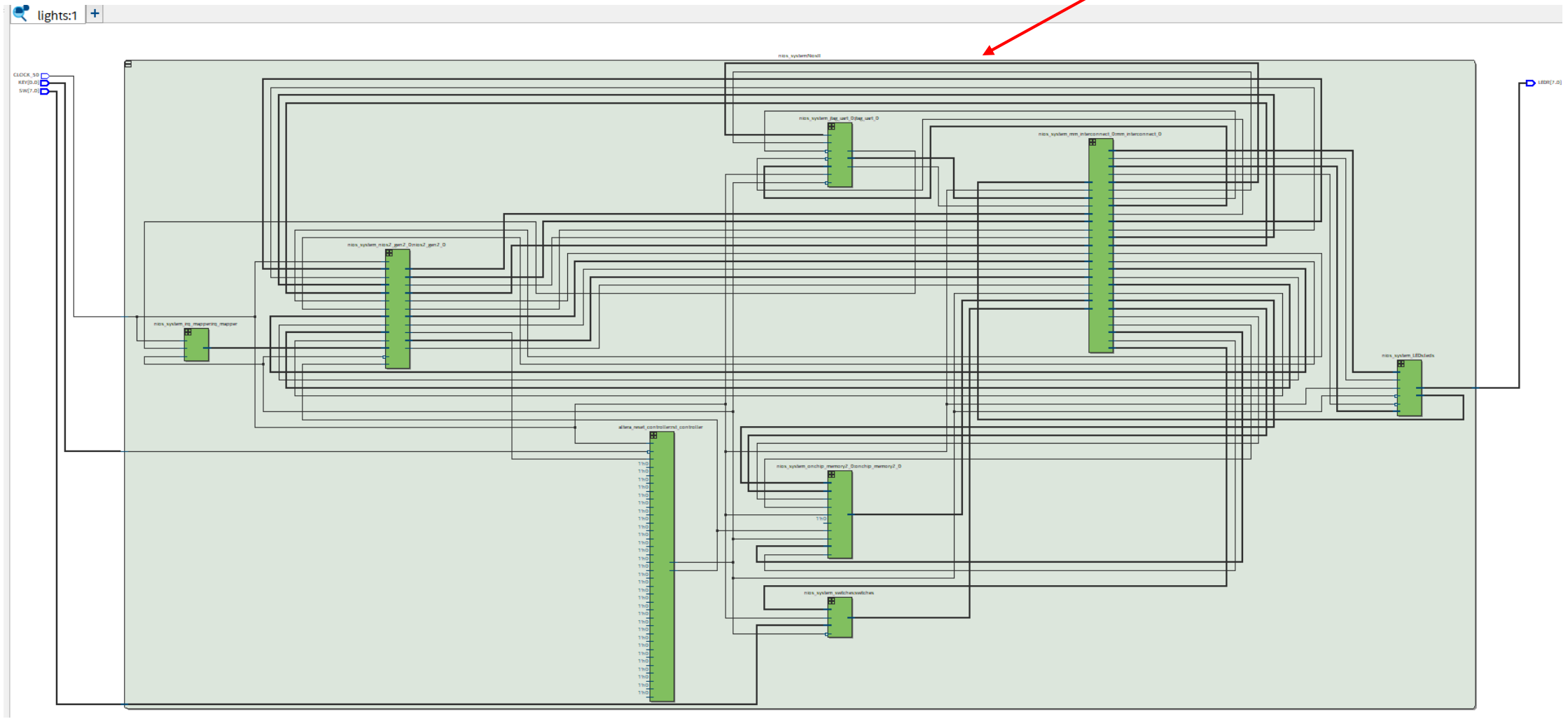
# Implémentation du SoC

**Après la compilation, le système généré est illustré dans la figure suivante**



# Implémentation du SoC

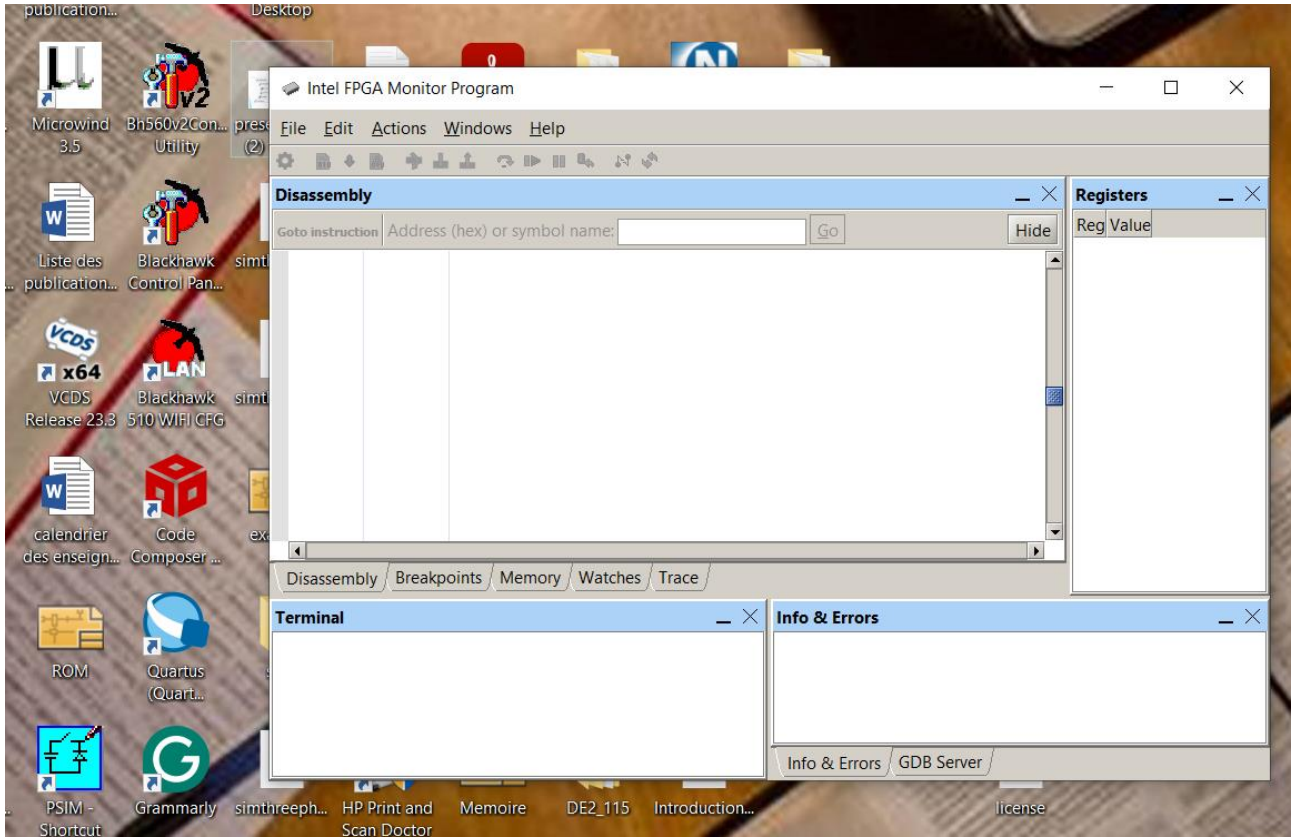


# Implémentation du SoC

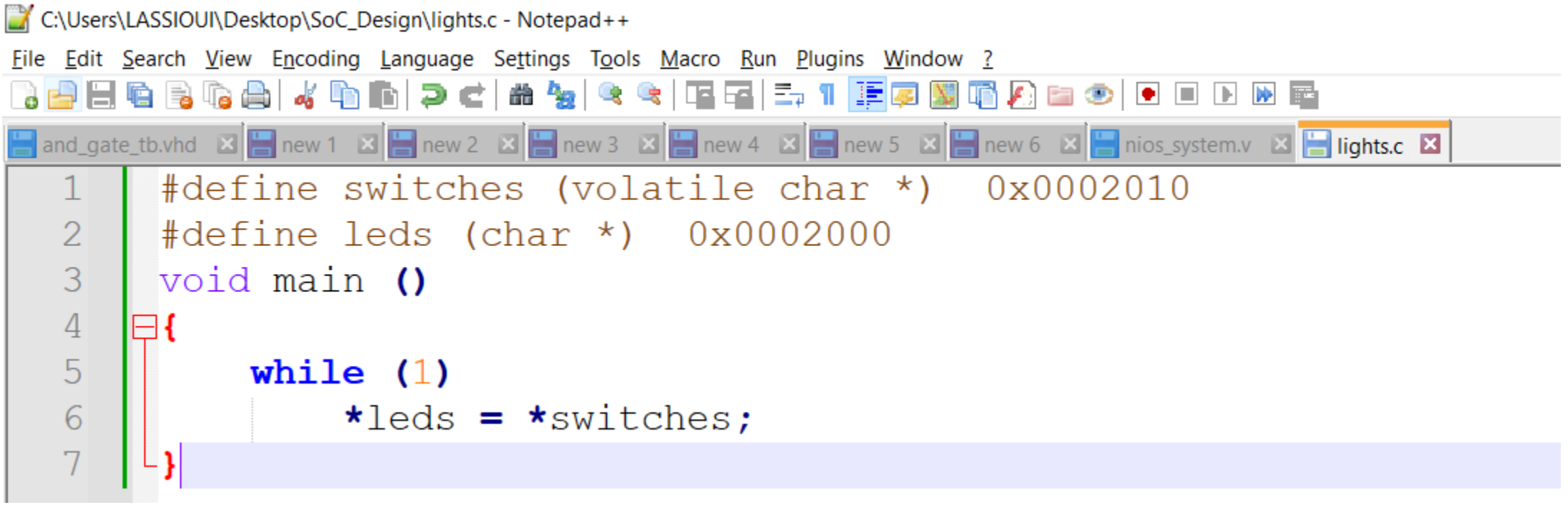
## FPGA Monitor program:

**Dans ce qui suit on va utiliser le logiciel FPGA Monitor Program pour charger la configuration dans le circuit FPGA et aussi pour programmer le Processeur Nios II**



# Implémentation du SoC

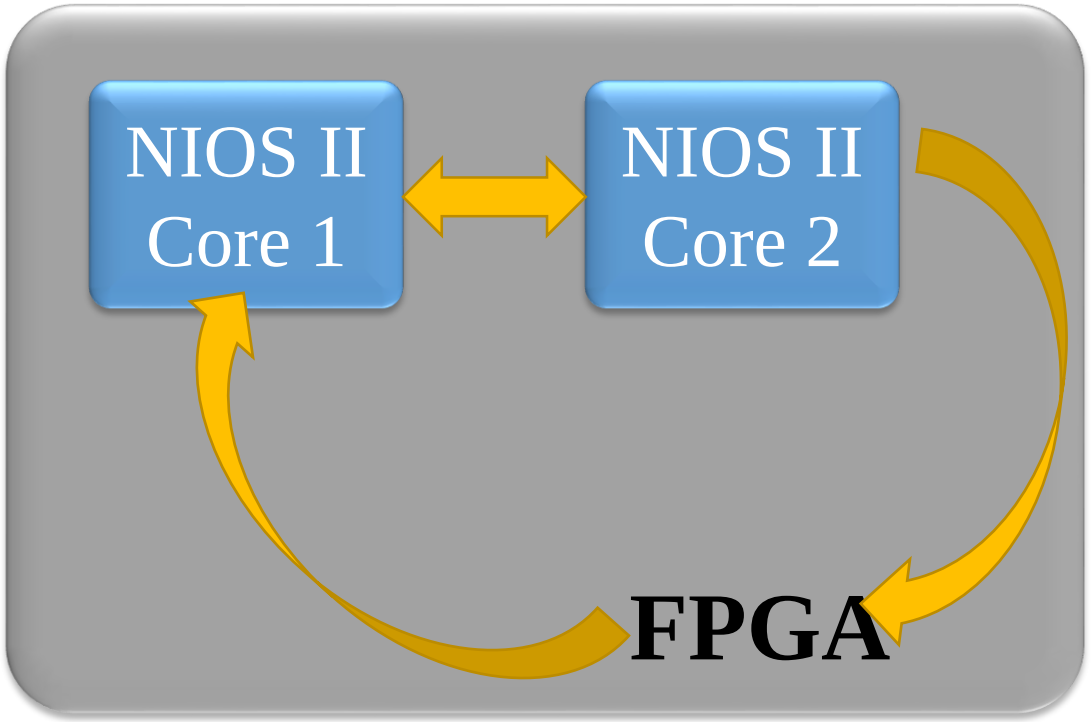
**Créer un fichier lights.c dans notepad++ et enregistrer le dans le répertoire de travail**



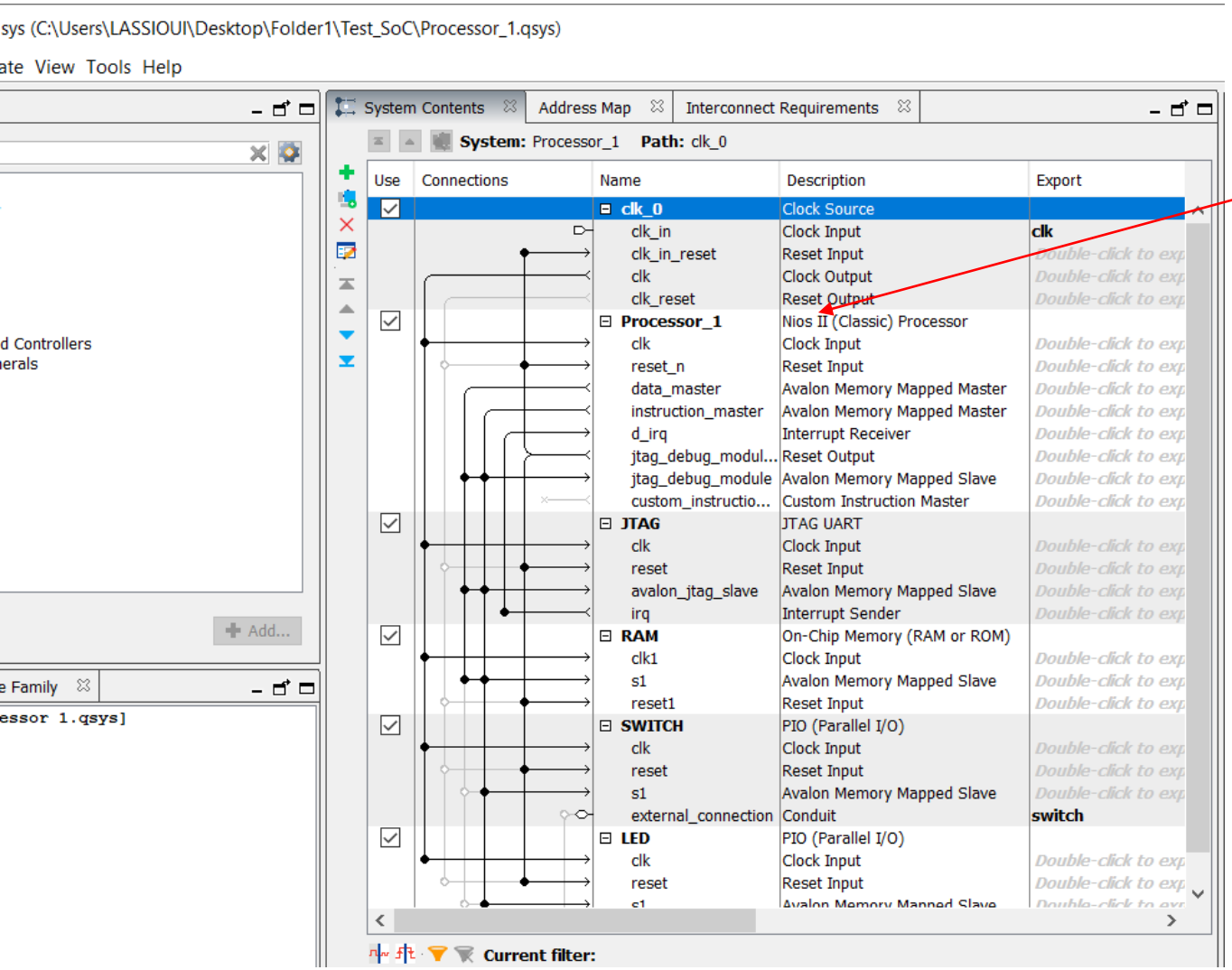
# Implémentation du SoC

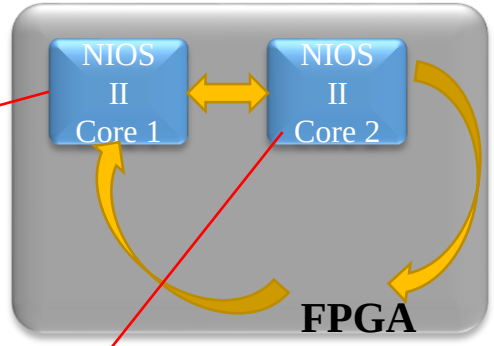
**Charger le fichier dans le circuit FPGA et vérifier son fonctionnement.**

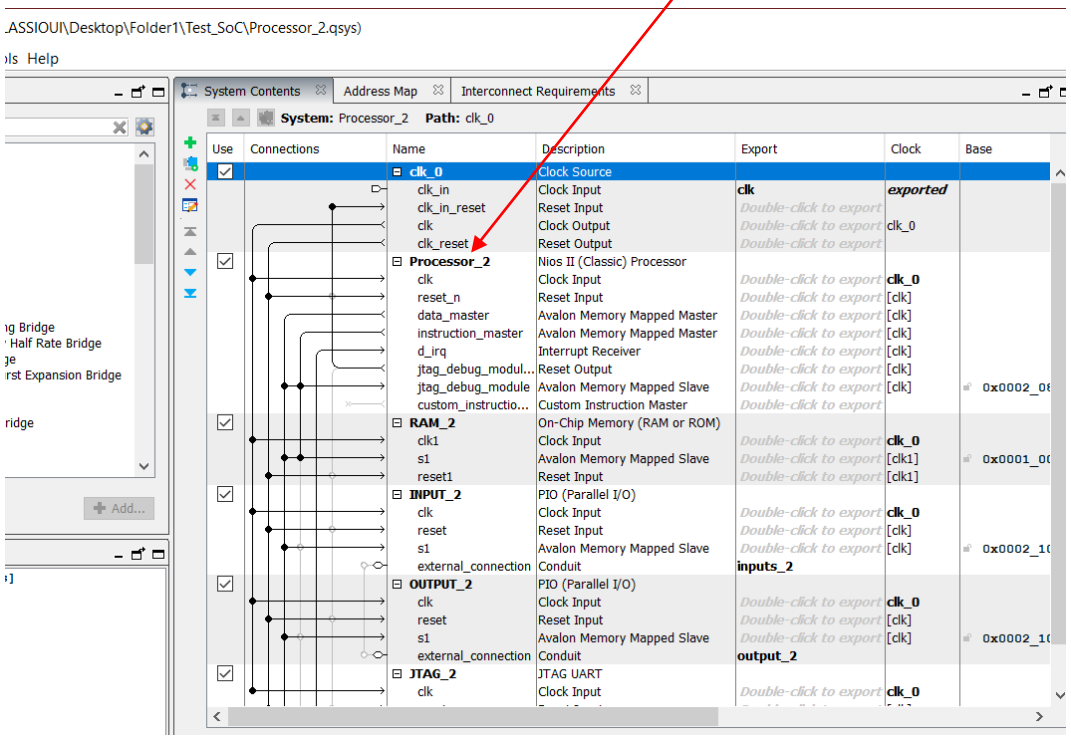
# Application II: Implémentation d’un SoC Dual Core



# Application II: Implémentation d'un SoC Dual Core

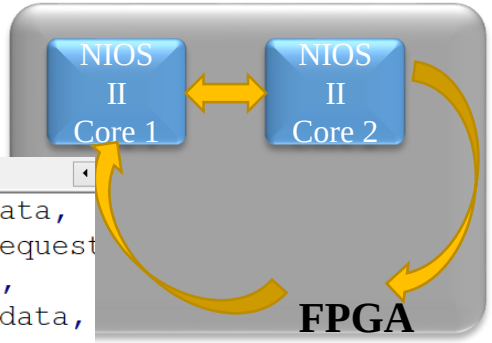


# Application II: Implémentation d'un SoC Dual Core

## Un extrait du code vhdl généré par Qsys




Processor_1_inst.vhd | new 8 | AND_GATE.vhd | AND_GATE_TB.vhd | test.php | test_db.php | Processor_2_inst.vhd | Processor_2.vhd

```
337             jtag_debug_module_readdata                => mm_interconnect_0_processor_2_jtag_debug_module_readdata,
338             jtag_debug_module_waitrequest             => mm_interconnect_0_processor_2_jtag_debug_module_waitrequest
339             jtag_debug_module_write                   => mm_interconnect_0_processor_2_jtag_debug_module_write,
340             jtag_debug_module_writedata               => mm_interconnect_0_processor_2_jtag_debug_module_writedata,
341             no_ci_readra                              => open
342         );
343
344     ram_2 : component Processor_2_RAM_2
345         port map (
346             clk        => clk_clk,                                -- clk1.clk
347             address    => mm_interconnect_0_ram_2_s1_address,     --   s1.address
348             clken      => mm_interconnect_0_ram_2_s1_clken,       --     .clken
349             chipselect => mm_interconnect_0_ram_2_s1_chipselect, --     .chipselect
350             write      => mm_interconnect_0_ram_2_s1_write,       --     .write
351             readdata   => mm_interconnect_0_ram_2_s1_readdata,    --     .readdata
352             writedata  => mm_interconnect_0_ram_2_s1_writedata,   --     .writedata
353             byteenable => mm_interconnect_0_ram_2_s1_byteenable, --     .byteenable
354             reset      => rst_controller_reset_out_reset,         -- reset1.reset
355             reset_req  => rst_controller_reset_out_reset_req,     --     .reset_req
356             freeze     => '0'                                     -- (terminated)
357         );
358
359     mm_interconnect_0 : component Processor_2_mm_interconnect_0
360         port map (
361             clk_0_clk_clk                                       => clk_clk,
362             Processor_2_reset_n_reset_bridge_in_reset_reset => rst_controller_reset_out_reset,
363             Processor_2_data_master_address                     => processor_2_data_master_address,
```

# Application II: Implémentation d'un SoC Dual Core

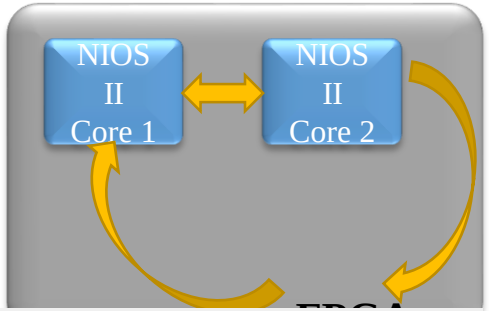
C:/Users/LASSIOUI/Desktop/Folder1/Test_SoC/Test_SoC - Test_SoC

Assignments Processing Tools Window Help

Test_SoC

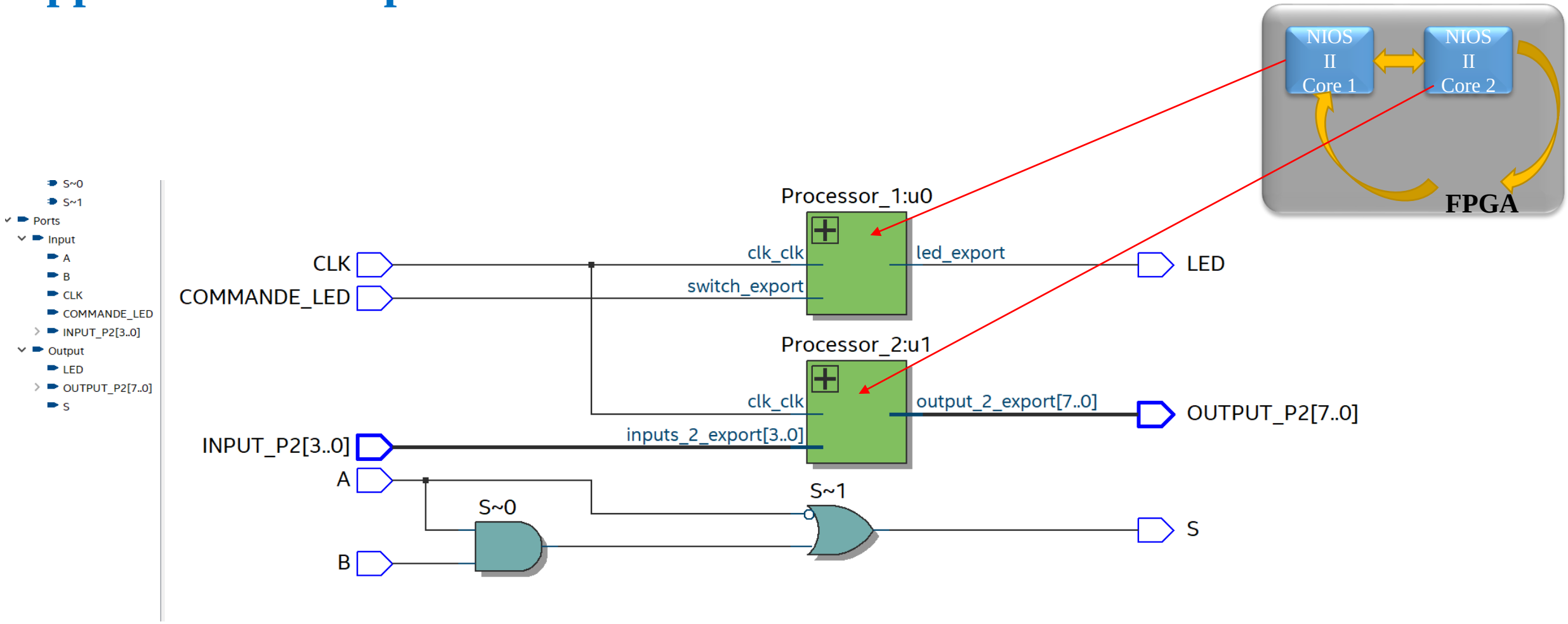
Test_SoC.vhd | Compilation Report - Test_SoC

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  entity Test_SoC is
4  port (
5  INPUT_P2 : IN std_logic_vector(3 downto 0);
6  OUTPUT_P2 : OUT std_logic_vector(7 downto 0) ;
7  A, B: in std_logic;
8  S: out std_logic;
9  CLK: in std_logic;
10 COMMANDE_LED: in std_logic;
11 LED: out std_logic);
12 end entity;
13 architecture arc of Test_SoC is
14
15 ---------------
16 component Processor_1 is
17       port (
18          clk_clk       : in  std_logic := 'X'; -- clk
19          switch_export : in  std_logic := 'X'; -- export
20          led_export    : out std_logic         -- export
21       );
22    end component Processor_1;
23
```




```
23
24    -------------------------------
25    component Processor_2 is
26       port (
27          clk_clk         : in  std_logic                     := 'X';
28          inputs_2_export : in  std_logic_vector(3 downto 0) := (others
29          output_2_export : out std_logic_vector(7 downto 0)
30       );
31    end component Processor_2;
32    -------------------------------
33 begin
34    u0 : component Processor_1
35       port map (
36          clk_clk       => CLK,       --    clk.clk
37          switch_export => COMMANDE_LED, -- switch.export
38          led_export    => LED     --    led.export
39       );
40       ----------------------------------------
41
42       u1 : component Processor_2
43       port map (
44          clk_clk         => CLK,         --      clk.clk
45          inputs_2_export => INPUT_P2, -- inputs_2.export
46          output_2_export => OUTPUT_P2  -- output_2.export
47       );
48       ----------------------------------------
49
50    S <= (A and B) or not(A);
51
52       end architecture;
53
```

# Application II: Implémentation d'un SoC Dual Core



# Application II: Implémentation d'un SoC Dual Core

## Programme en langage C du processeur N1

## Ce code permet de contrôler l'état de la led en utilisant le switch

Nios II - Processor_1/hello_world_small.c - Eclipse

File Edit Source Refactor Navigate Search Project Nios II Run Window Help

Project Explorer

- Processor_1
  - Includes
  - system
  - hello_world_small.c
  - create-this-app
  - Makefile
  - readme.txt
- Processor_1_bsp [Processor_1]
  - Includes
  - drivers
    - inc
      - altera_avalon_jtag_uart_fd.h
      - altera_avalon_jtag_uart_regs.h
      - altera_avalon_jtag_uart.h
      - altera_avalon_pio_regs.h
    - src
  - HAL
  - alt_sys_init.c
  - linker.h

hello_world_small.c | system.h | altera_avalon_pio_regs.h

```
 * "Small Hello World" example.
#include "system.h"
#include "stdio.h"
#include "altera_avalon_pio_regs.h"


int main()
{
  printf("Hello from Nios II!\n");

  /* Event loop never exits. */
  while (1)
  {
      IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, IORD_ALTERA_AVALON_PIO_DATA(SWITCH_BASE));

  }

  //return 0;
}
```