Projet avancé

Codesign sur carte Intel DE10-Standard

SOULEYMANE SOUMAH
ANIS YAGOUB
FATIMA ENNACIRI

ENCADRANT :**M. PATRICE KADIONIK**

ANNÉE :**2022/2023**

Table des matières

1 Introduction

Ce rapport décrit le développement d'un design hardware sur le logiciel Quartus II et l'intégration d'un programme software développé à l'aide du logiciel Eclipse en C/C++. Il couvre les étapes de conception, de synthèse et de téléchargement du code sur notre cible matérielle (Carte DE-10 Standard de chez Intel). Les fonctionnalités du design, les choix de conception et les résultats obtenus sont également discutés dans ce rapport. Ce travail a été effectué dans le but de démontrer les capacités de Quartus II en matière de développement de designs hardware professionnels et de fournir une base solide pour des travaux pratiques futurs.

2 Prise en main du matériel et de différents logiciels

La conception conjointe matérielle et logicielle d'un design (description d'un circuit numérique en VHDL ou HDL à implémenter sur un FPGA) nécessite la prise en mais de plusieurs outils afin de faciliter les tâches. La première chose à connaître est d'abord la nature des composants constituant la carte FPGA, puis les logiciels à utiliser (Quartus Prime II pour le design hardware et l'IDE Eclipse pour le développement des programmes en C/C++).

2.1 Carte Intel DE10-Standard

Comme la montre la figure 35 dans l'annexe, les composants de ce système sont mis en œuvre à l'aide du FPGA et le système de processeur dur (HPS) à l'intérieur du chip SoC Cyclone V d'Intel. Le FPGA implémente deux processeurs Nios II et plusieurs ports périphériques : mémoire, modules de timer, entrée/sortie audio, entrée/sortie vidéo, PS/2, analogique-numérique, réception/transmission infrarouge et ports parallèles connectés aux commutateurs et aux lumières. Le HPS comprend un ARM Cortex A9 dual-core et un ensemble de périphériques[1].

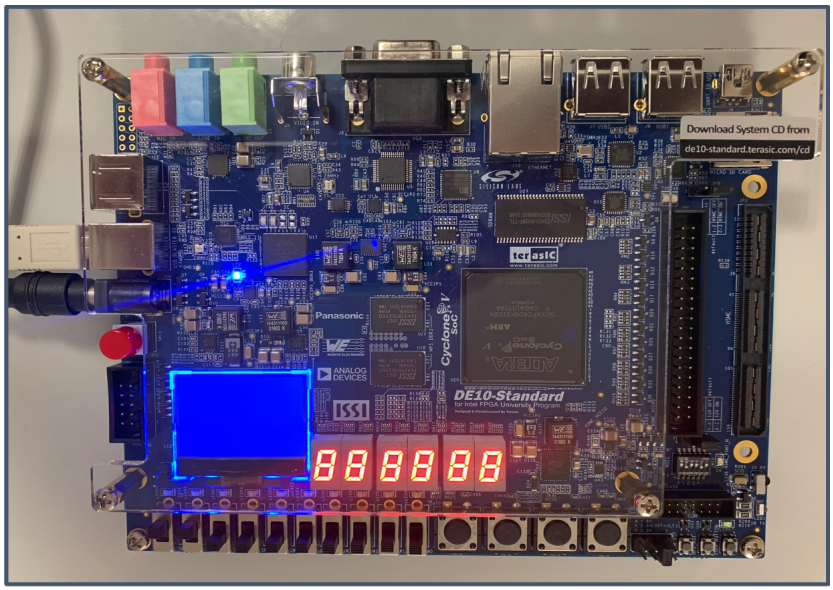


FIGURE 1 – DE10-Standard Board

2.2 Quartus Prime II

Altera Complete Design Suite est un package qui fournit un ensemble complet d'outils nécessaires à la conception et la mise en œuvre de circuits logiques numériques à l'aide des dispositifs FPGA d'Altera. Le logiciel Quartus II est le principal logiciel utilisé pour créer des designs de référence avec l'environnement de développement intégré de processeur embarqué à cœur logiciel Nios II.

2.3 IDE Eclipse

L'IDE Eclipse est le logiciel utilisé pour le développement de la partie logicielle du design hardware. Il permet de générer le *Board Support Package* qui contient un ensemble de données et de fonctions permettant à un système d'exploitation embarqué de fonctionner sur la cible matérielle.

3 Blocs IP utilisés

Les principaux blocs utilisés pour faire un design de base permettent de créer un système de référence opérationnel. Il est composé d'un processeur pour l'exécution des instructions, de la mémoire pour le stockage des données, système d'horloge, JTAG, et Interval timer. Il peut contenir des bloc IP supplémentaires afin d'enrichier sa fonctionnalité.

3.1 Processeur Nios II

Le processeur Intel Nios II est un processeur 32 bits qui peut être implémenté dans un périphérique FPGA Intel. Deux versions du processeur Nios II sont disponibles, désignées économie (/e) et rapide (/f). Le DE10-Standard inclut deux instances de la version Nios II/f, configurées avec la prise en charge matérielle en virgule flottante.

3.2 PLL

La PLL (Phase-Locked Loop) permet d'asservir la phase ou la fréquence de sortie d'un système sur la fréquence ou la phase du signal d'entrée. Dans note système, la mémoire SDRAM a une horloge de fréquence de 50Mhz mais avec un déphasage par rapport au système embarqué. On utilise donc la PLL pour générer un système d'hotloge compatible aux caractéristiques de la SDRAM.

3.3 JTAG

Le port JTAG permet la communication entre la carte DE10-Standard et l'ordinateur afin de transférer les fichiers de programme à la carte. La programmation JTAG permet le téléchargement direct de la configuration bitsream dans le Soc Cyclone V de FPGA. Les informations sont configurées tant que la carte FPGA est sous tension.

3.4 Interval Timers

La carte FPGA contient un module timer utilisé par le processeur Nios II. Il est composé de 6 registres de 16bits mappés aux adresses entre 0xFF202000 et 0xFF202004.

3.5 System ID

Le module *System ID* fournit une valeur spécifique au système DE10-Standard. L'ordinateur hôte permet de contrôler la carte en effectuant une lecture du port JTAG et puis une vérification de la valeur de l'identificateur renvoyé pour confirmer que le DE10-Standard Computer a été correctement téléchargé sur la carte DE10-Standard avant de tenter d'exécuter le code compilé pour le système.

3.6 GPIO

Pour la configuration des périphériques d'entrée-sortie de la carte (spécificment l'afficheur 7-segments, les Leds, les boutons poussoirs, et les switches), on utilise des ports parallèles d'entrée/sortie.

- Les 10 switches de la carte sont connectés à *input parallel port*. C'est un registre de données en mode lecture seule codé sur 10bits et mappé à l'adresse 0xFF200040.

- Les boutons poussoirs sont connectés à un *input parallel port* contenant 3 registres de 4bits d'adresse de base 0xFF200050. Le premier registre permet d'enregistrer les données lues par KEY_{3-0}, et les deux autres sont utilisés par les interruptions et les exceptions.

- Les Leds sont contrôlés par un *output parallel port* qui contient un registre de 10bits d'adresse de base 0xFF200000.

- L'afficheur 7-segments de la carte est connecté aux deux ports parallèles, chacun contenant un registre de données en écriture seule 32 bits. Le registre à l'adresse 0xFF200020 lit les chiffres HEX3 à HEX0, et le registre à l'adresse 0xFF200030 lit les chiffres HEX5 et HEX4. Les deux registres peuvent lire les données qui contrôlent les segments de chaque afficheur en fonction de l'emplacement des bits.

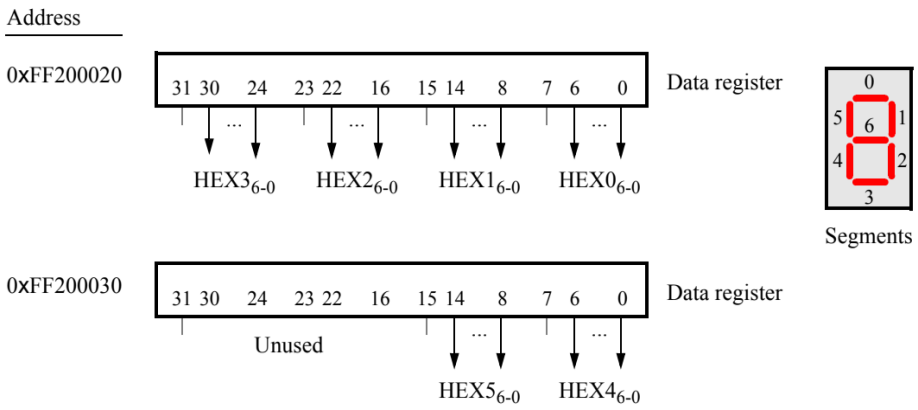


FIGURE 2 – Emplacement des bits pour l'afficheur 7-segments

3.7 Interruptions GPIO

4 Design de base

Le design de base est composé d'éléments qui permettent d'avoir une architecture fonctionnelle. Nous avons donc utilisé une PLL pour générer la clock, le processuer Nios II, une mémoire RAM, le JTAG UART, l'interval Timer, et à la fin quelques périphériques entrée-sortie (Switch, bouton poussoir, afficheur 7-segment, LED). Le fichier Qsys du projet décrit la liaison entre ces composants. Pour les configurer sur **Platform Designer** on suit les étapes suivantes :

- Ajouter le bloc **system and SDRAM Clocks for DE-series Boards**
- Faire la configuration suivante :

FIGURE 3 – Configuration de PLL

- Faire un click droit sur le nom du composant et le renommer en sys_sdram_pll_0.

- Faire un click droit sur ref_clk → Connections : System_PLL.ref_clk → export as : system_pll_ref_clk.

- Refaire la même procédure pour ref_reset, sdram_clk.

- Renommer le signal exporté de sdram_clk en sdram_clk

- Ajouter l'IP **Nios II processor** et faire les configurations ci-dessous :

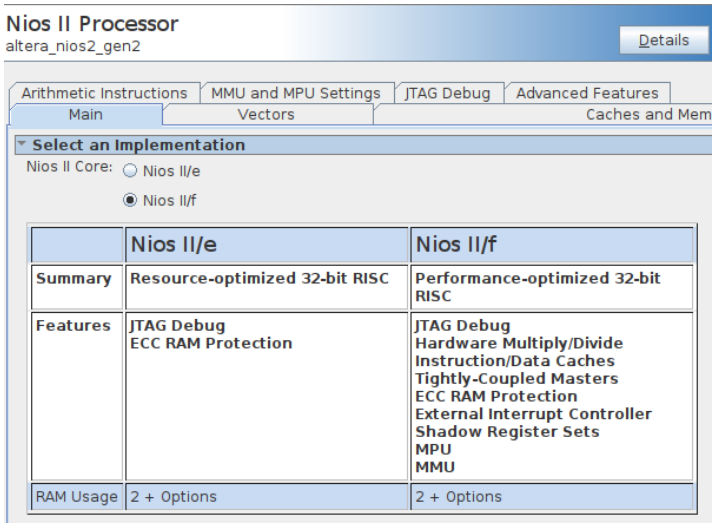


FIGURE 4 – Configuration du processeur Nios II (main)

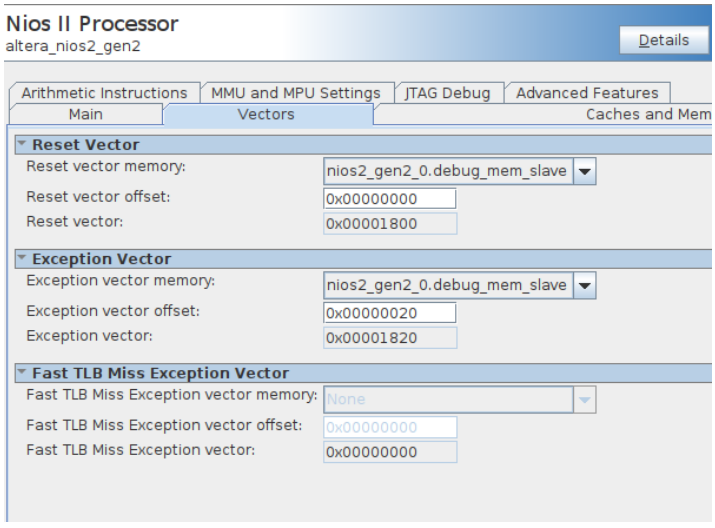


FIGURE 5 – Configuration du processeur Nios II (vectors)

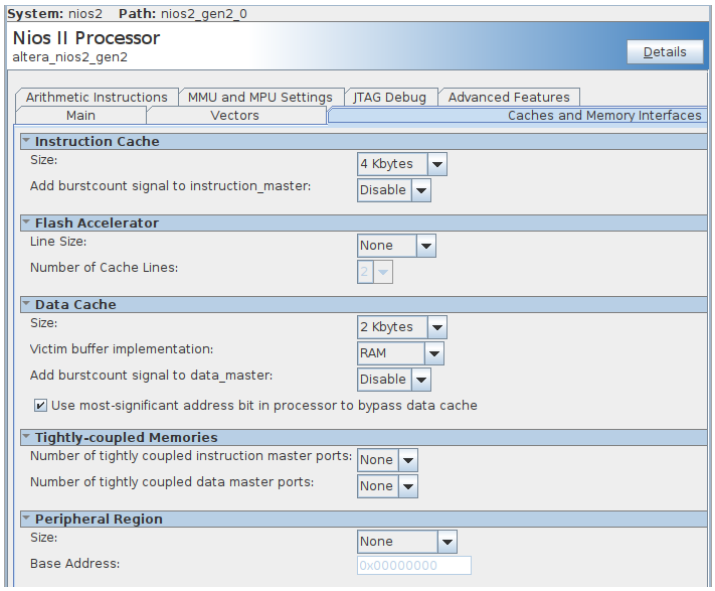


FIGURE 6 – Configuration du processeur Nios II (cache and memory interfaces)

- Faire un click droit sur le nom du composant et le renommer le bloc en nios2_gen2_0
- Ajouter la mémoire **On-Chip Memory (RAM or ROM) Intel FPGA IP** et suivre la configuration ci-dessous :

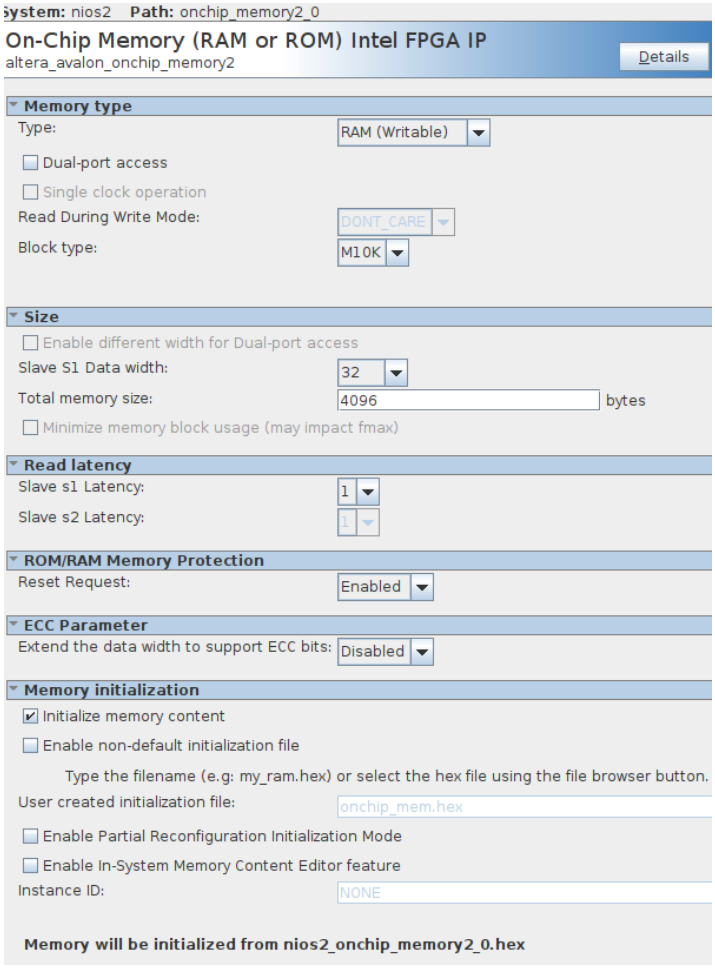


FIGURE 7 – Configuration de la RAM

- Ajouter l'IP **JTAG UART Intel FPGA IP** :

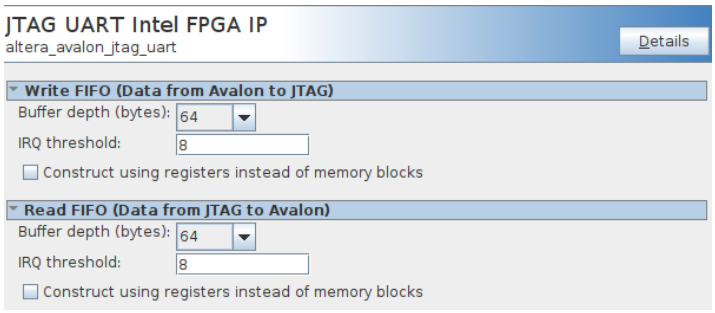


FIGURE 8 – Configuration de JTAG UART

- Le renommer en JTAG_UART

- Ajouter le bloc **Interval Timer Intel FPGA IP** :

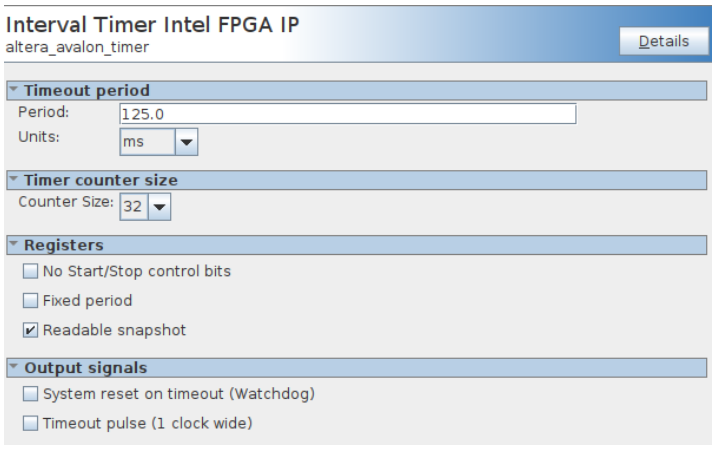


FIGURE 9 – Configuration d'interval Timer

- Le renommer en interval_timer.

- Ajouter l'IP **PIO (Parallel I/O) Intel FPGA IP** et faire la configuration ci-dessous :

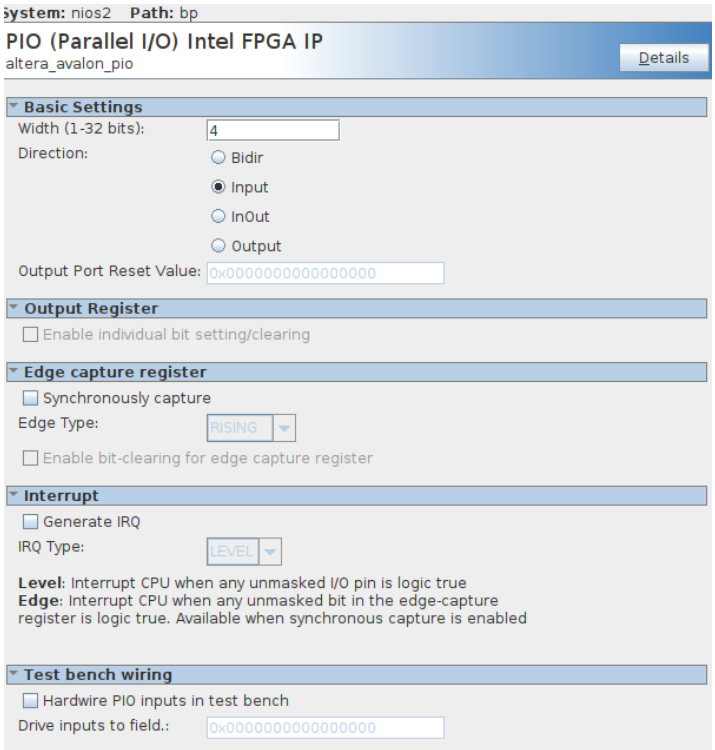


FIGURE 10 – Configuration des boutons poussoirs

- Renommer le bloc en **bp**

- Double click sur le signal external_connection dans la case **Export** et le renommer **bp_connexion**.

- Ajouter l'IP **PIO (Parallel I/O) Intel FPGA IP** et faire la configuration ci-dessous :

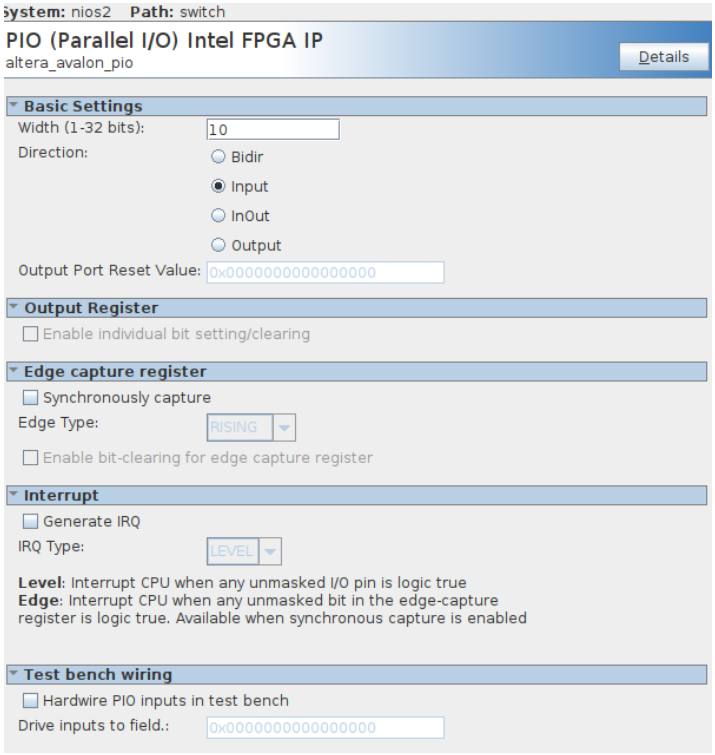


FIGURE 11 – Configuration des switches

- Renommer le bloc en **switch**
- Double click sur le signal external_connection dans la case **Export** et le renommer **slider_switches**.
- Ajouter l'IP **PIO (Parallel I/O) Intel FPGA IP** et faire la configuration ci-dessous :

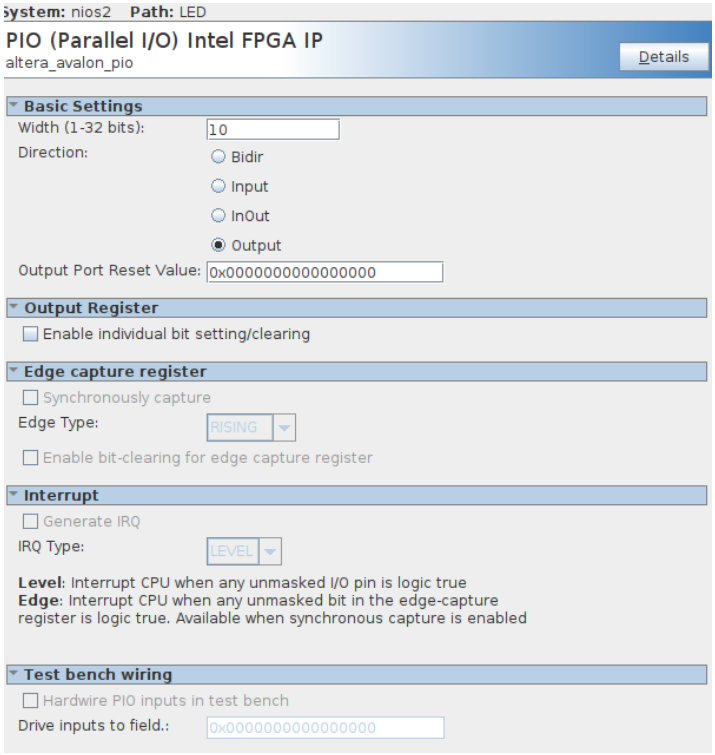


FIGURE 12 – Configuration des LEDs

- Renommer le bloc en **LED**
- Double click sur le signal external_connection dans la case **Export** et le renommer **leds**.

- Ajouter l'IP **SEG7_IF** et faire la configuration ci-dessous :

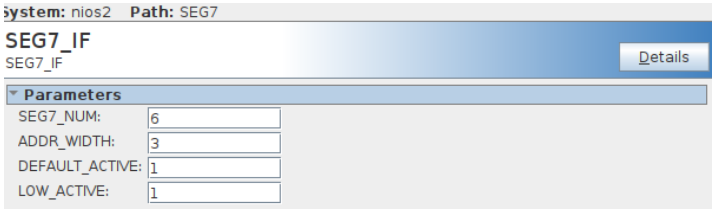


FIGURE 13 – Configuration de l'afficheur 7 segments

- Renommer le bloc en **SEG7**
- Double click sur le signal Conduit dans la case **Export** et le renommer **seg7**.

Après l'ajout de ces blocs, on relie leurs signaux de la façon suivante :

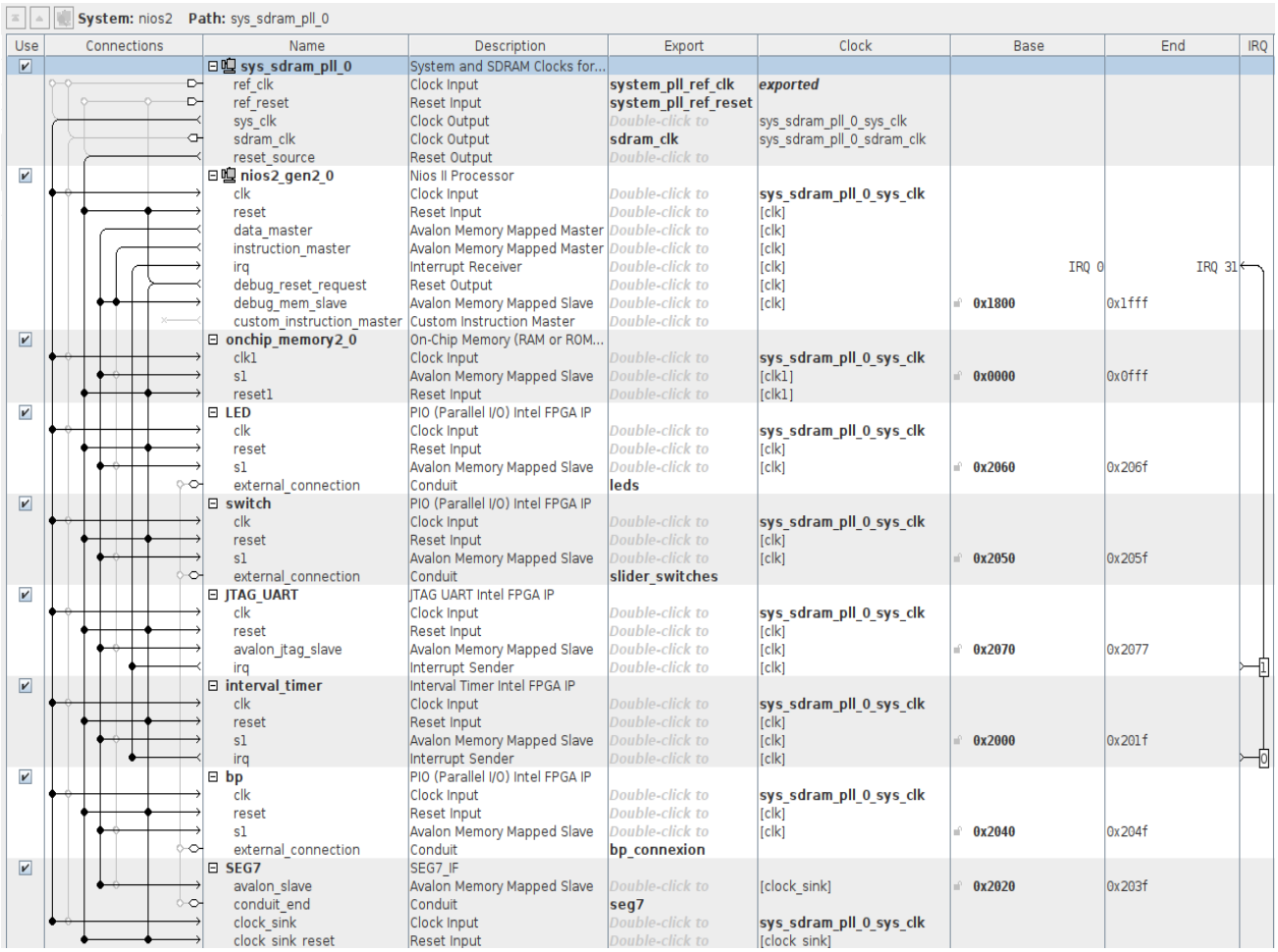


FIGURE 14 – Qsys du design de base

Les horloges et les signaux reset de tous les composants sont reliées respectivemment à la même horloge et au même reset (ceux de sys_sdram_pll_0). Les signaux *slave* qui reçoivent les instructions à exécuter sont reliés au signal *master* du processuer Nios II (signal *data_master*).

5 VGA Subsystem

Le sous-système qui permet de contrôler l'affichage sur l'écran utilise le port VGA de la carte DE10-Standard. Le sous-system que nous allons utiliser est fourni dans le projet **Computer_system** développé par Intel. Il est composé d'un ensemble d'IPs qui permettent au concepteur de dessiner des formes en couleurs et d'afficher du texte sur l'écran.

Dans ce qui suit nous allons détailler les différents blocs qui constituent notre sous-système et leur fonctions :

5.1 Clock source

Le premier composant que nous utilisons est l'horloge. Elle est identique à l'horloge du système (50Mhz) et définit la fréquence de fonctionnement de la majorité des blocs de notre sous-système.

5.2 Mémoire SDRAM

Un contrôleur SDRAM dans le FPGA fournit une interface avec la RAM dynamique synchrone (SDRAM) de 64 Mo sur la carte DE10-Standard, qui est organisée en 32M x 16 bits. Il est accessible par le processeur Nios II et est mappé à l'espace d'adressage 0x00000000 à 0x03FFFFFF.

5.3 VGA_Clk)

Nous utilisons une seconde horloge que nous connectons au contrôleur VGA cette horloge est cadencée à 25 Mhz en raisons de la résolution que nous avons choisi (640x480) ainsi que la fréquence de rafraîchissement de l'écran à 60Hz.

5.4 VGA pixel DMA

Le contrôleur DMA utilise une interface mappée en mémoire pour lire les images vidéo d'une mémoire externe. Puis il envoie ces images vidéo via son interface de flux Avalon. L'interface *Avalon memory-mapped slave* est utilisée pour communiquer avec les registres internes du contrôleur. Le contrôleur Pixel Buffer DMA peut utiliser soit un adressage consécutif ou l'adressage en mode X-Y pour lire et écrire les images depuis et en destination de la mémoire.

5.5 Dual-clock FIFO

La dual clock FIFO bufferise les données vidéo et permet le transfert d'un flux de données entre deux domaines d'horloges différents. Les données sont bufferisées dans une mémoire de type FIFO. Ensuite, les données sont lues à partir de la FIFO à la fréquence d'horloge de sortie et diffusées hors du bloc IP.

5.6 RGB Resampler

Le RGB Reasampler convertit les flux vidéo entre les formats d'espace colorimétrique RGB.
Bien que le bloc IP puisse convertir n'importe quel format RGB, la conversion au format Niveaux de gris 8 bits doit être évitée. La manière utilisée pour convertir au format en niveaux de gris est très rudimentaire.

5.7 Scaler

Le bloc IP scaler modifie la résolution d'un flux vidéo. Le scaler convertit la résolution d'un flux entrant en ajoutant ou en supprimant des colonnes/lignes entières de pixels.

5.8 char buf subsystem

Ce bloc fourni par Intel dans le projet computer system est composé d'autres bloc IP que nous n'allons pas détailler.
Le Character Buffer pour l'affichage VGA convertit les caractères ASCII en représentation graphique pour l'affichage. Un programme s'exécutant sur le processeur Nios II peut envoyer des codes de caractères ASCII à l'interface Avalon du Character Buffer,nommé *avalon_char_slave*. Le bloc IP stocke les caractères dans sa mémoire. Le contrôleur DMA lit le caractères ASCII de la mémoire et les envoie au character renderer. Le bloc render convertit le caractères ASCII dans leur représentation graphique et les envoie via une interface Avalon Streaming.

5.9 Alpha blender

Le bloc IP alpha blender combine deux flux vidéo en un seul. Les deux flux entrants sont appelés *foreground* et *background* sont mélangés ensemble pour créer un seul flux en sortie. Le *foreground* doit être au format 40-bit RGBA, tandis que le *background* doit être au format 30-bit RGB.

5.10 VGA controller

Le bloc IP VGA controller génère les signaux de timing requis par le convertisseur numérique analogique VGA sur la carte DE10-standard. Les données sont fournis au contrôleur VGA via son interface avalon streaming. Le contrôleur prend les données en entrées, ajoute les timing VGA adéquats et ensuite, envoie ces informations au convertisseur numérique analogique VGA sur la carte DE10-standard.

6 Design de référence pour l'affichage sur VGA

Dans cette partie nous allons voir comment construire un design pas à pas pour afficher du texte sur un moniteur VGA.

— ajouter l'IP **system and SDRAM Clocks for DE-series Boards**

— Configurer le bloc comme sur la figure suivante :

FIGURE 15 – Configuration de l'horloge système

- Faire un click droit sur le nom du composant et le renommer le bloc en System_PLL.
  Faire un click droit sur ref_clk → Connections : System_PLL.ref_clk → export as : system_pll_ref_clk.

- Répéter la même procédure pour ref_reset, sdram_clk.

- Renommer le signal exporté de sdram_clk en sdram_clk.

— Ajouter L'IP **Video clocks for DE-series Boards**.

- Configurer le bloc comme sur la figure suivante :

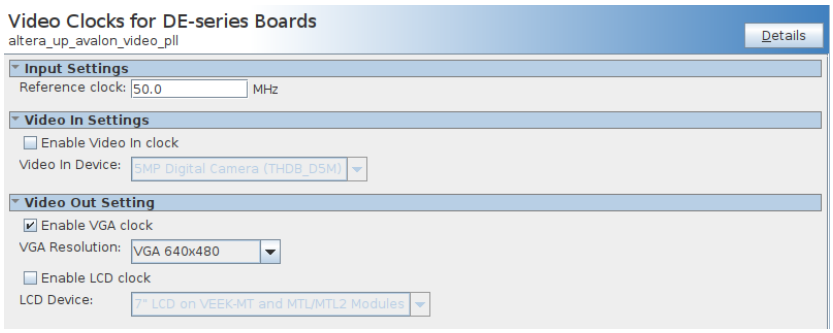


FIGURE 16 – configuration de l'horloge vidéo

- Faire un click droit sur le nom du composant et le renommer le bloc en Video_PLL.

- Faire un click droit sur ref_clk → Connections : Video_PLL.ref_clk → export as : video_pll_ref_clk.

- Répéter la même procédure pour ref_reset.

— Ajouter l'IP **Nios II processor** :

— Faire un click droit sur le nom du composant et le renommer le bloc en Nios2.

— Connecter le bloc comme sur la figure suivante :

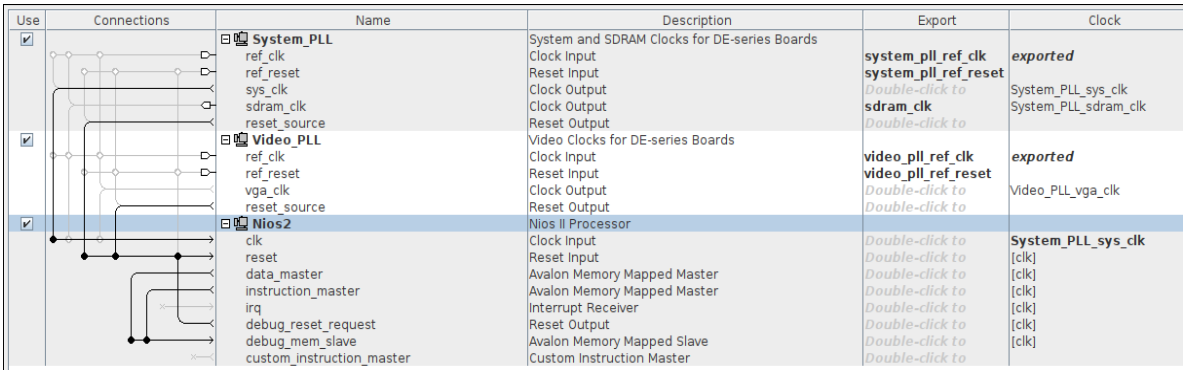


FIGURE 17 – connections du Nios2

— Ajouter l'IP **JTAG UART intel FPGA IP** :

— Faire un click droit sur le nom du composant et le renommer le bloc en JTAG_UART.

— Configurer le bloc comme sur la figure suivante :

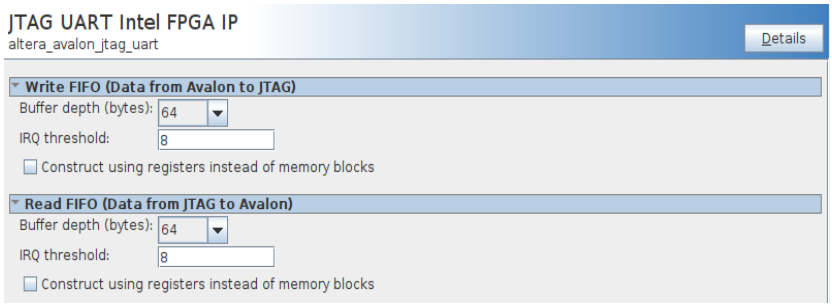


FIGURE 18 – configuration du JTAG

— Connecter le bloc comme sur la figure suivante :

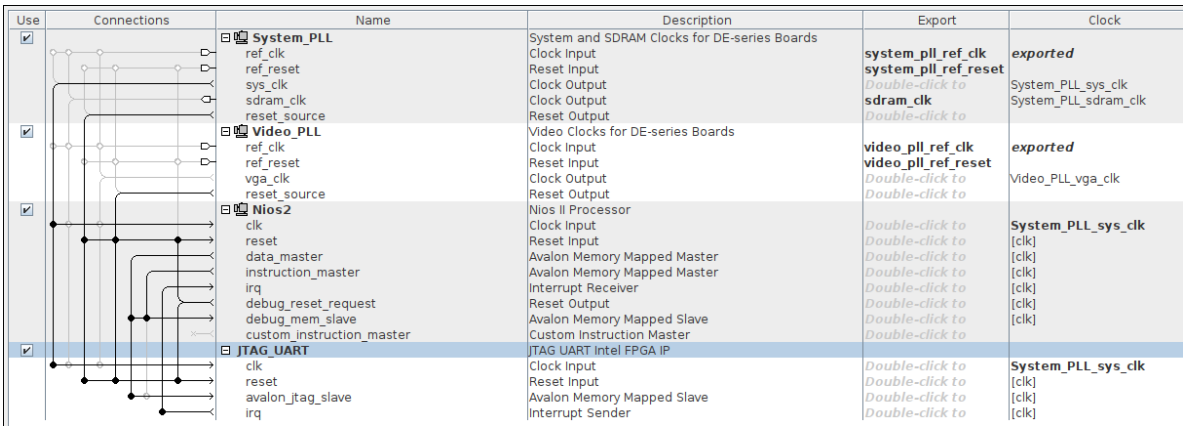


FIGURE 19 – Connections du JTAG

— Ajouter l'IP **System ID peripheral Intel FPGA IP** :

  — Faire un click droit sur le nom du composant et le renommer le bloc en SysID.

  — Configurer le bloc comme sur la figure suivante :

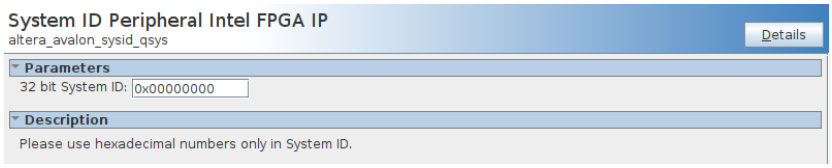
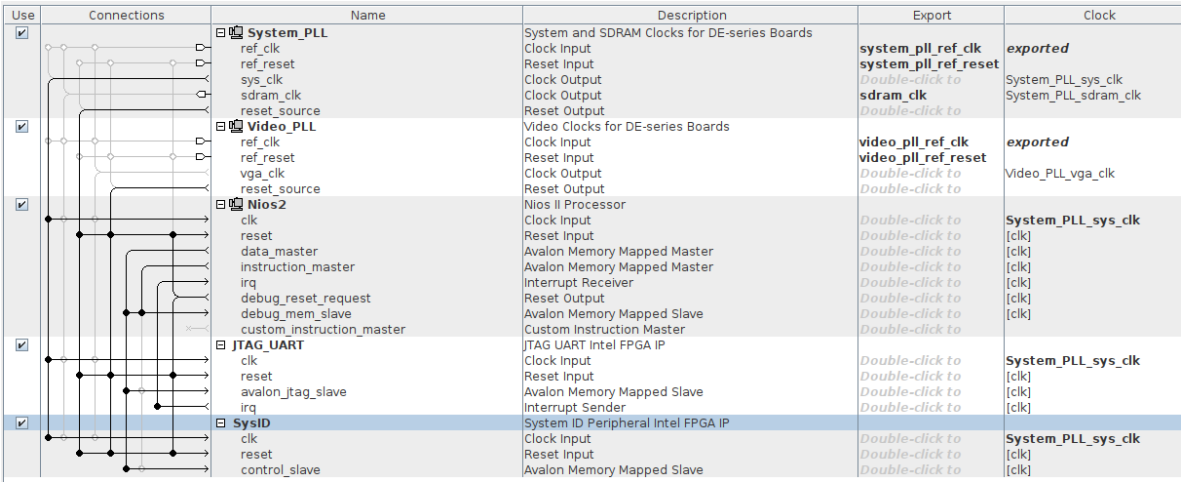


FIGURE 20 – Configuration System ID

  — Connecter le bloc comme sur la figure suivante :



Use	Connections	Name	Description	Export	Clock
☑		**System_PLL**	System and SDRAM Clocks for DE-series Boards		
		ref_clk	Clock Input	**system_pll_ref_clk**	***exported***
		ref_reset	Reset Input	**system_pll_ref_reset**	
		sys_clk	Clock Output	*Double-click to*	System_PLL_sys_clk
		sdram_clk	Clock Output	**sdram_clk**	System_PLL_sdram_clk
		reset_source	Reset Output	*Double-click to*	
☑		**Video_PLL**	Video Clocks for DE-series Boards		
		ref_clk	Clock Input	**video_pll_ref_clk**	***exported***
		ref_reset	Reset Input	**video_pll_ref_reset**	
		vga_clk	Clock Output	*Double-click to*	Video_PLL_vga_clk
		reset_source	Reset Output	*Double-click to*	
☑		**Nios2**	Nios II Processor		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		data_master	Avalon Memory Mapped Master	*Double-click to*	[clk]
		instruction_master	Avalon Memory Mapped Master	*Double-click to*	[clk]
		irq	Interrupt Receiver	*Double-click to*	[clk]
		debug_reset_request	Reset Output	*Double-click to*	[clk]
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		custom_instruction_master	Custom Instruction Master	*Double-click to*	
☑		**JTAG_UART**	JTAG UART Intel FPGA IP		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		irq	Interrupt Sender	*Double-click to*	[clk]
☑		**SysID**	System ID Peripheral Intel FPGA IP		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		control_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]

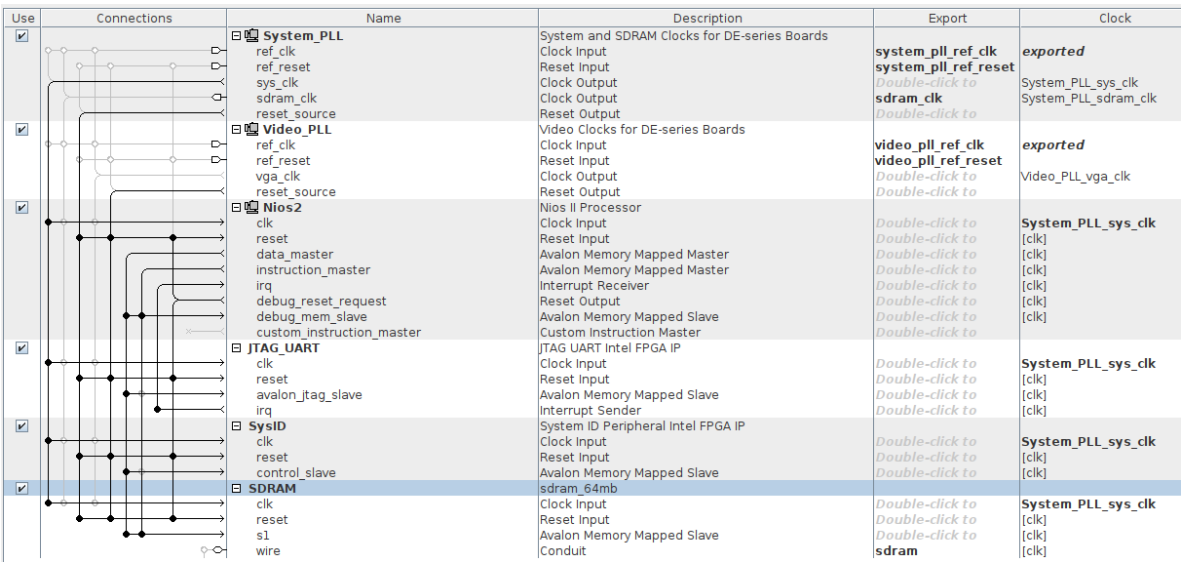
FIGURE 21 – Connections dusystem ID

— Ajouter l'IP **sdram_64mb** :

— Faire un click droit sur le nom du composant et le renommer le bloc en SDRAM.

— Faire un click droit sur wire → Connections : System_PLL.wire → export as : sdram_64_mb

— Renommer le signal exporté en sdram.



Use	Connections	Name	Description	Export	Clock
☑		**System_PLL**	System and SDRAM Clocks for DE-series Boards		
		ref_clk	Clock Input	**system_pll_ref_clk**	***exported***
		ref_reset	Reset Input	**system_pll_ref_reset**	
		sys_clk	Clock Output	*Double-click to*	System_PLL_sys_clk
		sdram_clk	Clock Output	**sdram_clk**	System_PLL_sdram_clk
		reset_source	Reset Output	*Double-click to*	
☑		**Video_PLL**	Video Clocks for DE-series Boards		
		ref_clk	Clock Input	**video_pll_ref_clk**	***exported***
		ref_reset	Reset Input	**video_pll_ref_reset**	
		vga_clk	Clock Output	*Double-click to*	Video_PLL_vga_clk
		reset_source	Reset Output	*Double-click to*	
☑		**Nios2**	Nios II Processor		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		data_master	Avalon Memory Mapped Master	*Double-click to*	[clk]
		instruction_master	Avalon Memory Mapped Master	*Double-click to*	[clk]
		irq	Interrupt Receiver	*Double-click to*	[clk]
		debug_reset_request	Reset Output	*Double-click to*	[clk]
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		custom_instruction_master	Custom Instruction Master	*Double-click to*	
☑		**JTAG_UART**	JTAG UART Intel FPGA IP		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		irq	Interrupt Sender	*Double-click to*	[clk]
☑		**SysID**	System ID Peripheral Intel FPGA IP		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		control_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
☑		**SDRAM**	sdram_64mb		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		wire	Conduit	**sdram**	[clk]

FIGURE 22 – Connections SDRAM

— Ajouter l'IP **On-Chip Memory** :

— Faire un click droit sur le nom du composant et le renommer le bloc en On-chip_SRAM.

— Configurer le bloc comme sur la figure suivante :

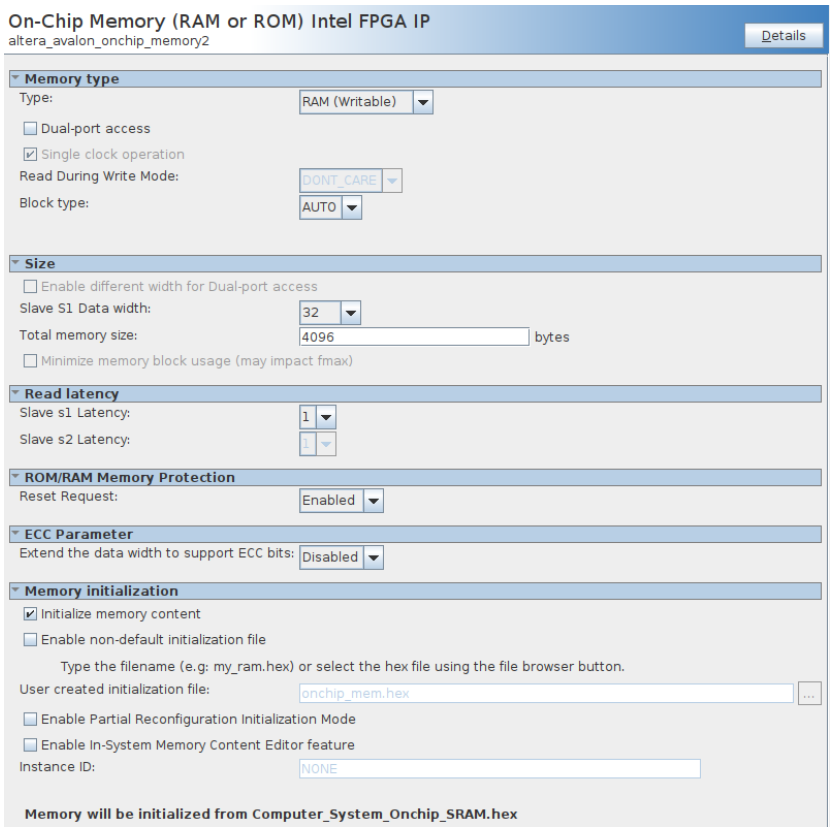


FIGURE 23 – Configuration de la mémoire

— Connecter le bloc comme sur la figure suivante :

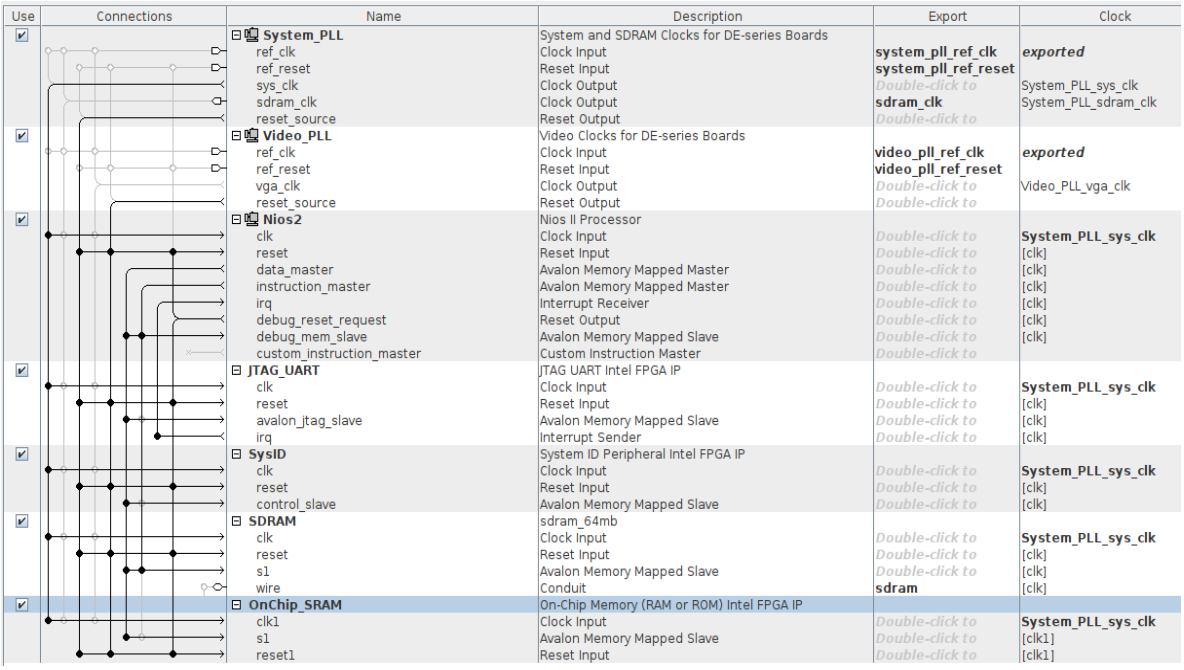
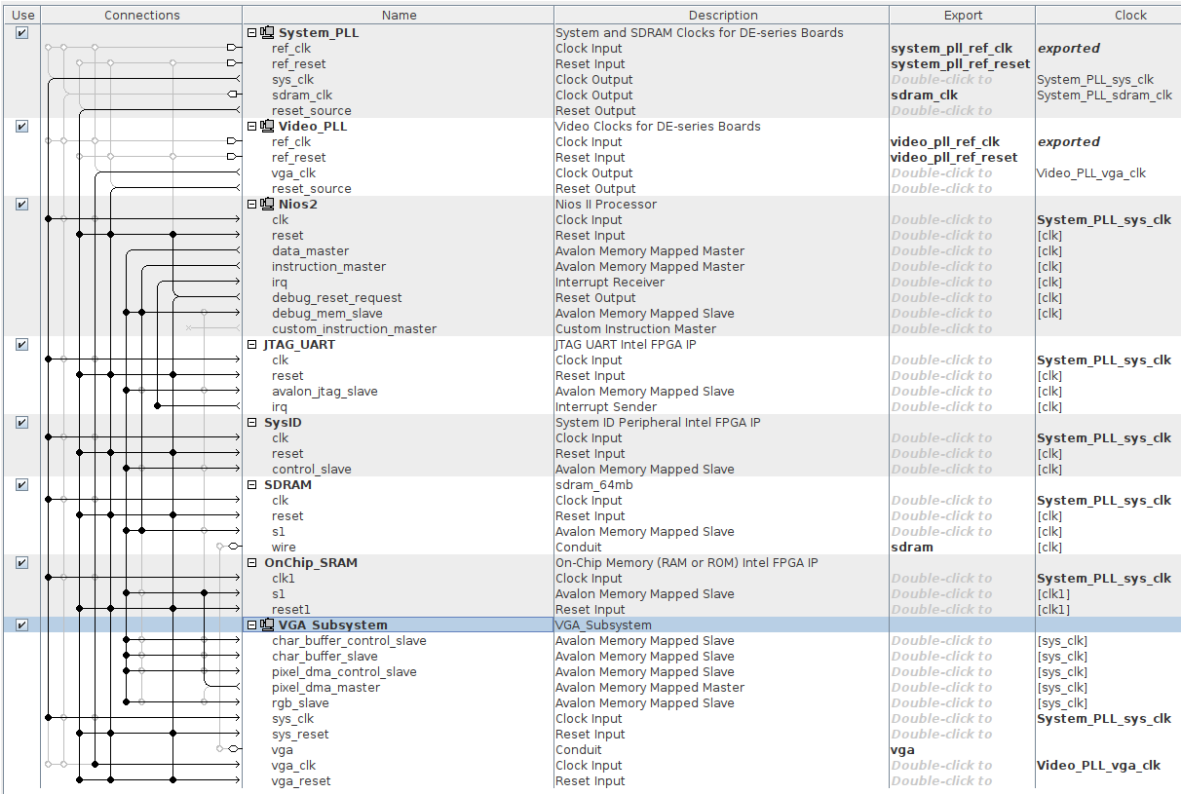


FIGURE 24 – Connections mémoire

— Ajouter l'IP **VGA_Subsystem** :

  — Faire un click droit sur le nom du composant et le renommer le bloc en VGA_Subsystem.

  — Connecter le bloc comme sur la figure suivante :



Use	Connections	Name	Description	Export	Clock
☑		**System_PLL**	System and SDRAM Clocks for DE-series Boards		
		ref_clk	Clock Input	**system_pll_ref_clk**	***exported***
		ref_reset	Reset Input	**system_pll_ref_reset**	
		sys_clk	Clock Output	*Double-click to*	System_PLL_sys_clk
		sdram_clk	Clock Output	**sdram_clk**	System_PLL_sdram_clk
		reset_source	Reset Output	*Double-click to*	
☑		**Video_PLL**	Video Clocks for DE-series Boards		
		ref_clk	Clock Input	**video_pll_ref_clk**	***exported***
		ref_reset	Reset Input	**video_pll_ref_reset**	
		vga_clk	Clock Output	*Double-click to*	Video_PLL_vga_clk
		reset_source	Reset Output	*Double-click to*	
☑		**Nios2**	Nios II Processor		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		data_master	Avalon Memory Mapped Master	*Double-click to*	[clk]
		instruction_master	Avalon Memory Mapped Master	*Double-click to*	[clk]
		irq	Interrupt Receiver	*Double-click to*	[clk]
		debug_reset_request	Reset Output	*Double-click to*	[clk]
		debug_mem_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		custom_instruction_master	Custom Instruction Master	*Double-click to*	
☑		**JTAG_UART**	JTAG UART Intel FPGA IP		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		avalon_jtag_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		irq	Interrupt Sender	*Double-click to*	[clk]
☑		**SysID**	System ID Peripheral Intel FPGA IP		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		control_slave	Avalon Memory Mapped Slave	*Double-click to*	[clk]
☑		**SDRAM**	sdram_64mb		
		clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		reset	Reset Input	*Double-click to*	[clk]
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk]
		wire	Conduit	**sdram**	[clk]
☑		**OnChip_SRAM**	On-Chip Memory (RAM or ROM) Intel FPGA IP		
		clk1	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		s1	Avalon Memory Mapped Slave	*Double-click to*	[clk1]
		reset1	Reset Input	*Double-click to*	[clk1]
☑		**VGA_Subsystem**	VGA_Subsystem		
		char_buffer_control_slave	Avalon Memory Mapped Slave	*Double-click to*	[sys_clk]
		char_buffer_slave	Avalon Memory Mapped Slave	*Double-click to*	[sys_clk]
		pixel_dma_control_slave	Avalon Memory Mapped Slave	*Double-click to*	[sys_clk]
		pixel_dma_master	Avalon Memory Mapped Master	*Double-click to*	[sys_clk]
		rgb_slave	Avalon Memory Mapped Slave	*Double-click to*	[sys_clk]
		sys_clk	Clock Input	*Double-click to*	**System_PLL_sys_clk**
		sys_reset	Reset Input	*Double-click to*	
		vga	Conduit	**vga**	
		vga_clk	Clock Input	*Double-click to*	**Video_PLL_vga_clk**
		vga_reset	Reset Input	*Double-click to*	

FIGURE 25 – Connections VGA

— Configurer le NIOS 2.

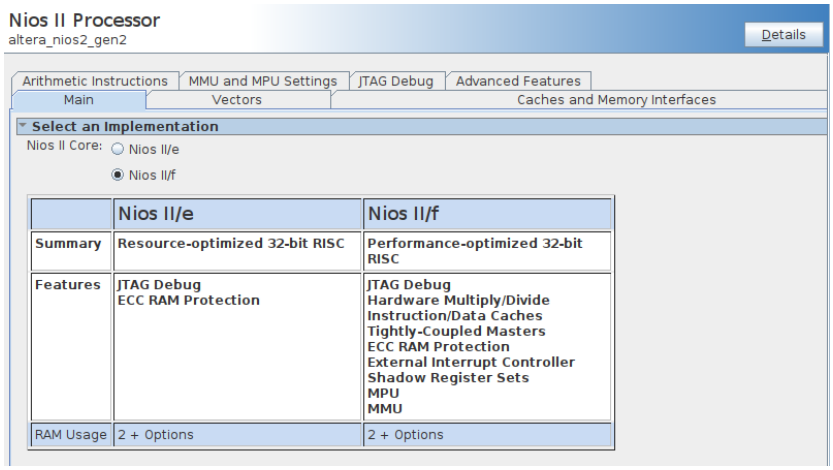


FIGURE 26 – Configuration du Nios2 (main)

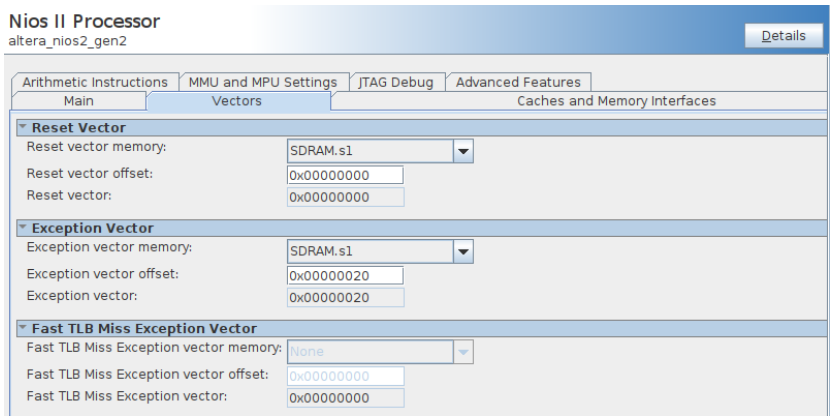


FIGURE 27 – Configuration du Nios2 (vectors)

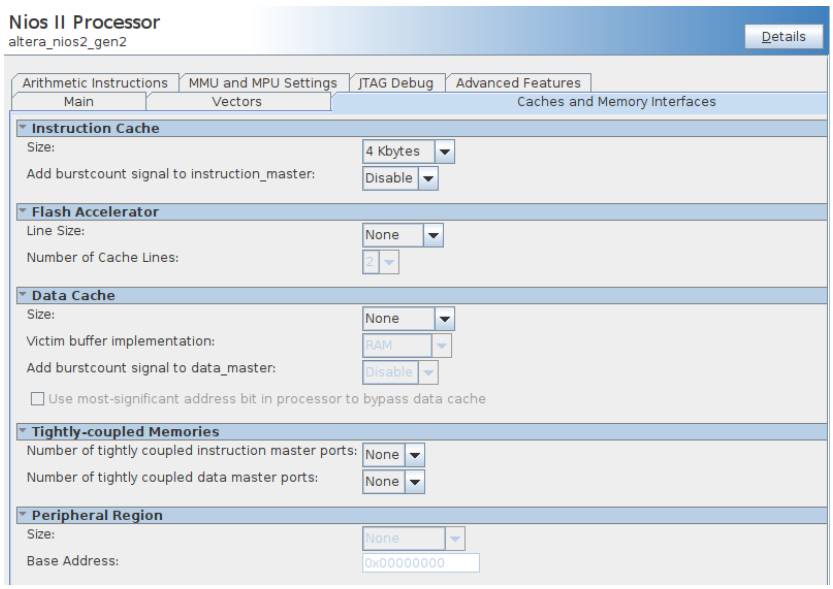


FIGURE 28 – Configuration du Nios2 (cahe and memory interface)

— Aller dans system→ assign base adresses.
— Aller dans system→ create global reset network.
— Appuyer Generate .
— Modifier le top level comme indiqué en annexes.
— Appuyer sur compile et charger le design dans la carte.

7 Développement de la partie logicielle

7.1 Génération de BSP

Le *Board Support Package* est un paquet logiciel contenant le bootloader, le HAL (*Hardware Abstraction Layer* : définit les fonctions standardisées de manipulation du hardware) ou un OS embarqué (pour notre cas c'est le µ*C*/*OS* II), et les pilotes. Il peut aussi contenir des composants logiciels supplémentaires pour ajouter d'autres fonctionnalités au système.

Deux types de BSP peuvent être générées : HAL BSP pour une programmation sans utilisation d'un RTOS, et RT BSP en utilisant le noyau µ*C*/*OS* II. La génération des BSP implique la génération d'un fichier *system.h* directement via un fichier d'extension .sopinfo spécifique au design : c'est le fichier qui contient toutes les données qui permettent la communication entre le hard et le soft.

7.2 GPIO(Led, switch, BP)

Nous nous basons sur les données et les fonctions générées par le BSP pour implémenter un programme qui contrôle les périphériques sur la carte. Pour cela, nous incluons le fichier *system.h* qui contient les adresses sous formes de constantes, en plus de deux fonctions IORD et IOWR pour lire et écrire sur les ports GPIO. Nous avons comme exemple les fonctions suivantes :

- Pour les LEDs :

void BSP_init(void) : initialisation des valeurs stockées dans les adresses associées aux LEDs.

void BSP_setLED(unsigned char lite) : allumer une LED ;

void BSP_clrLED(unsigned char lite) : éteindre une LED.

- Pour les switches :

int BSP_SW (unsigned char s) : retourne 1 si le switch est mis sur 1 et 0 sinon.

- Pour les boutons poussoirs :

int BSP_BP (unsigned char s) : retourne 1 si le bouton poussoir est appuyé et 0 sinon.

- Pour les 7-segments :

void SEG7_Clear(void) : désactiver les afficheurs 7-segments ;

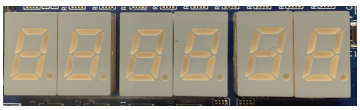


FIGURE 29 – void SEG7_Clear(void)

void SEG7_Full(void) : activer tous les segments de l'afficheur ;

FIGURE 30 – void SEG7_Full(void)

void SEG7_Number(void) : afficher "543210" ;

FIGURE 31 – void SEG7_Number(void)

void SEG7_Hex(alt_u32 Data) : afficher Data en hexadécimal ;

FIGURE 32 – void SEG7_Hex(78)

void SEG7_Dec(alt_u32 Data) : afficher Data en décimal.

FIGURE 33 – void SEG7_Dec(33)

7.3 Les interruptions

8 Afficher du texte sur un moniteur VGA

Dans cette partie nous allons décrire le code en C que nous avons écrit afin d'afficher du texte sur un moniteur VGA.
Le code repose en grande partie sur les fonctions et fichiers générés par le BSP. Le fichier *altera_up_avalon_video_dma_controller.h* contient la définition de la structure *alt_up_video_dma_dev*. Le fichier *system.h* donne le nom complet du périphérique *Char_buf_DMA* ce qui nous permet de l'ouvrir. Enfin en utilisant la fonction générée par le BSP *alt_up_video_dma_draw_string* nous écrivons un message sur l'écran.

```c
#include <stdio.h>
#include "../tst1_bsp/alt_sys_init.c"
int main()
{
    alt_up_video_dma_dev * char_buffer;

    char_buffer = alt_up_video_dma_open_dev("/dev/
   VGA_Subsystem_Char_Buf_Subsystem_Char_Buf_DMA");

    if (char_buffer == NULL)
    {
        printf("Error:Character buffer is not defined\n");
    }
    else
    {
        printf("Character buffer is ready to use\n");
    }

//================================================
    alt_up_video_dma_draw_string(char_buffer,"ENSEIRB MATMECA"
   ,30,10,0);
  return 0;
}
```

Listing 1 – Code pour afficher du texte

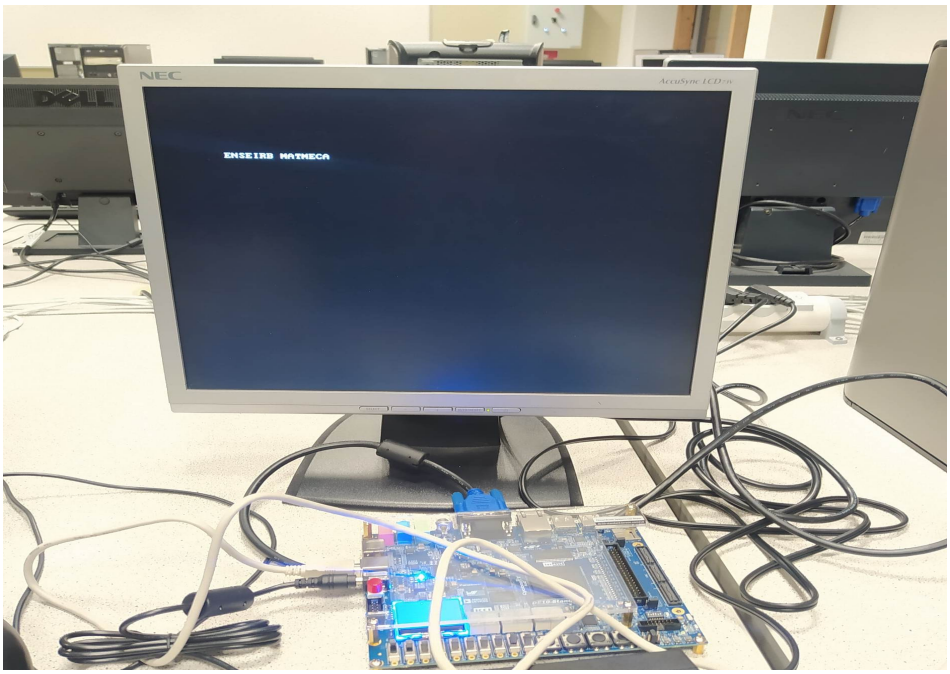


FIGURE 34 – Affichage de texte sur moniteur VGA

9 Jeu de conversion

Afin de vérifier le bon fonctionnement de notre design, nous le testons par l'implémentation d'un jeu de conversion. C'est un jeu que nous avons développé en première année en VHDL et nous l'avons repris pour le re-développer en c++ sur la carte DE10-Srandard. L'objectif de ce jeu est de :

- Générer un un nombre aléatoire entre 0 et 100 et l'afficher en décimal sur les trois afficheurs 7-segments en droite (la génération se fait en appuyant sur le BP1),
- L'utilisateur doit convertir ce nombre en binaire en utilisant les switches,
- Après la confirmation de la conversion par l'utilisateur(en appuyant sur le BP2), le score s'affiche sur l'afficheur 7 segment situé à gauche,
- Le joueur n'a que 5 tentatives et doit re-exécuter le programme dans le cas où il veut retenter de jouer.

Nous avons utilisé les fonctions déjà générées par le BSP pour coder ce jeu en bare-metal (pas besoin d'utiliser les caractéristiques du noyau temps réel pour ce développement).

10 Conclusion

En conclusion, ce rapport a décrit le développement d'un design hardware basé sur le logiciel Quartus II. Les étapes de conception, de simulation, de synthèse et de téléchargement du code sur la carte DE10-Standard ont été détaillées, ainsi que les fonctionnalités

du design, les choix de conception et les résultats obtenus après le test par l'implémentation du jeu de conversion. Il a été démontré que Quartus II est un outil puissant pour le développement de designs hardware professionnels. Ce projet a fourni une base solide et a permis de mettre en évidence les avantages de l'utilisation de Quartus II dans le développement de designs hardware. Ce rapport est un outil précieux pour les ingénieurs et les développeurs qui souhaitent utiliser Quartus II pour leurs projets futurs.

11 Annexes

11.1 Block diagram of the DE10-Standard Computer :

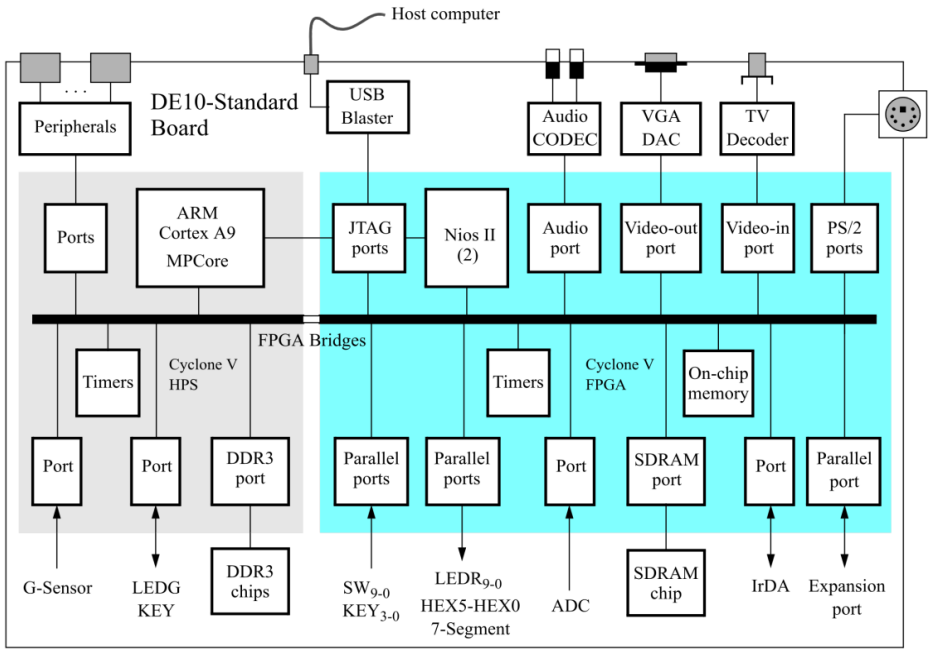


FIGURE 35 – Block diagram of the DE10-Standard Computer

11.2 Code source du jeu

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "../jeu_bm_bsp/system.h"
#include "../jeu_bm_bsp/HAL/inc/io.h"
#include "../jeu_bm_bsp/HAL/inc/priv/alt_busy_sleep.h"

#define SEG7_SET(index, seg_mask)    IOWR(SEG7_BASE,index,seg_mask)
#define SEG7_NUM     6

static unsigned char szMap[] = {
        63, 6, 91, 79, 102, 109, 125, 7,127, 111, 119, 124, 57, 94, 121, 113};
       //0,1,2,3, 4, 5, 6, 7, 8, 9, 'a', 'b', 'c', 'd', 'e', 'f'};


void SEG7_Clear(void){
    int i;
    for(i=0;i<SEG7_NUM;i++){
        SEG7_SET(i, 0x00);
    }
}

void SEG7_Full(void){
    int i;
    for(i=0;i<SEG7_NUM;i++){
```

```
27          SEG7_SET(i, 0xFF);
28      }
29  }
30
31  void SEG7_Number(void){
32      int i;
33      for(i=0;i<SEG7_NUM;i++){
34          SEG7_SET(i, szMap[i]);
35      }
36  }
37
38  void SEG7_Hex(alt_u32 Data, alt_u8 point_mask){
39      alt_u8 mask = 0x01;
40      alt_u8 seg_mask;
41      int i;
42
43      //
44      seg_mask = 0;
45      for(i=0;i<SEG7_NUM;i++){
46          seg_mask = szMap[Data & 0x0F];
47          Data >>= 4;
48          if (point_mask & mask)
49              seg_mask |= 0x80;
50          mask <<= 1;
51          SEG7_SET(i, seg_mask);
52      }
53  }
54
55  void SEG7_Decimal_al(alt_u32 Data, alt_u8 point_mask){
56      alt_u8 mask = 0x01;
57      alt_u8 seg_mask;
58      int i;
59
60      //
61      seg_mask = 0;
62      for(i=0;i<3;i++){
63          seg_mask = szMap[Data%10];
64          Data /= 10;
65          if (point_mask & mask)
66              seg_mask |= 0x80;
67          mask <<= 1;
68          SEG7_SET(i, seg_mask);
69      }
70
71  }
72
73  void SEG7_Decimal(alt_u32 Data, alt_u8 point_mask){
74      alt_u8 mask = 0x01;
75      alt_u8 seg_mask;
76      int i;
77
78      //
79      seg_mask = 0;
80      for(i=0;i<SEG7_NUM;i++){
81          seg_mask = szMap[Data%10];
82          Data /= 10;
83          if (point_mask & mask)
84              seg_mask |= 0x80;
85          mask <<= 1;
86          SEG7_SET(i, seg_mask);
87      }
88
89  }
90
91  void BSP_init(void){
```

```
92      IOWR(LED_BASE,0,0x00);
93  }
94
95  int BSP_SW(unsigned char s){
96      int a = IORD(SWITCH_BASE,0);
97      if(s == 1){
98          if(a & 0x01 == 0x01)
99              return 1;
100         else
101             return 0;
102     }
103     if(s == 2){
104         if((a & 0x02) == 0x02)
105             return 1;
106         else
107             return 0;
108     }
109     if(s == 3){
110         if((a & 0x04) == 0x04)
111             return 1;
112         else
113             return 0;
114     }
115     if(s == 4){
116         if((a & 0x08) == 0x08)
117             return 1;
118         else
119             return 0;
120     }
121     if(s == 5){
122         if((a & 0x10) == 0x10)
123             return 1;
124         else
125             return 0;
126     }
127     if(s == 6){
128         if((a & 0x20) == 0x20)
129             return 1;
130         else
131             return 0;
132     }
133     if(s == 7){
134         if((a & 0x40) == 0x40)
135             return 1;
136         else
137             return 0;
138     }
139     if(s == 8){
140         if((a & 0x80) == 0x80)
141             return 1;
142         else
143             return 0;
144     }
145     if(s == 9){
146         if((a & 0x100) == 0x100)
147             return 1;
148         else
149             return 0;
150     }
151     if(s == 10){
152         if((a & 0x200) == 0x200)
153             return 1;
154         else
155             return 0;
156     }
```

```
157 }
158
159 void BSP_setLED(unsigned char lite){
160     int a;
161     a = IORD(LED_BASE,0);
162     if(lite==1){
163         a = a | 0x01;
164         IOWR(LED_BASE,0,a);
165     }
166     if(lite==2){
167         a = a | 0x02;
168         IOWR(LED_BASE,0,a);
169     }
170     if(lite==3){
171         a = a | 0x04;
172         IOWR(LED_BASE,0,a);
173     }
174     if(lite==4){
175         a = a | 0x08;
176         IOWR(LED_BASE,0,a);
177     }
178     if(lite==5){
179         a = a | 0x10;
180         IOWR(LED_BASE,0,a);
181     }
182     if(lite == 6){
183         a = a | 0x20;
184         IOWR(LED_BASE,0,a);
185     }
186     if(lite==7){
187         a = a | 0x40;
188         IOWR(LED_BASE,0,a);
189     }
190     if(lite==8){
191         a = a | 0x80;
192         IOWR(LED_BASE,0,a);
193     }
194     if(lite==9){
195         a = a | 0x100;
196         IOWR(LED_BASE,0,a);
197     }
198     if(lite==10){
199         a = a | 0x200;
200         IOWR(LED_BASE,0,a);
201     }
202 }
203
204 void BSP_clrLED(unsigned char lite){
205     int a;
206     a = IORD(LED_BASE,0);
207     if(lite==1){
208         a = a & ~(0x01);
209         IOWR(LED_BASE,0,a);
210     }
211     if(lite==2){
212         a = a & ~(0x02);
213         IOWR(LED_BASE,0,a);
214     }
215     if(lite==3){
216         a = a & ~ 0x04;
217         IOWR(LED_BASE,0,a);
218     }
219     if(lite==4){
220         a = a & ~ 0x08;
221         IOWR(LED_BASE,0,a);
```

```
222     }
223     if(lite==5){
224         a = a & ~ 0x10;
225         IOWR(LED_BASE,0,a);
226     }
227     if(lite == 6){
228         a = a & ~ 0x20;
229         IOWR(LED_BASE,0,a);
230     }
231     if(lite==7){
232         a = a & ~ 0x40;
233         IOWR(LED_BASE,0,a);
234     }
235     if(lite==8){
236         a = a & ~ 0x80;
237         IOWR(LED_BASE,0,a);
238     }
239     if(lite==9){
240         a = a & ~ 0x100;
241         IOWR(LED_BASE,0,a);
242     }
243     if(lite==10){
244         a = a & ~ 0x200;
245         IOWR(LED_BASE,0,a);
246     }
247 }
248
249 int BSP_BP(unsigned char s){
250     int a = IORD(BP_BASE,0);
251         if(s == 1){
252             if(a & 0x01 == 0x01)
253                 return 0;
254             else
255                 return 1;
256         }
257         else if(s == 2){
258             if((a & 0x02) == 0x02)
259                 return 0;
260             else
261                 return 1;
262         }
263         else if(s == 3){
264             if((a & 0x04) == 0x04)
265                 return 0;
266             else
267                 return 1;
268         }
269         else if(s == 4){
270             if((a & 0x08) == 0x08)
271                 return 0;
272             else
273                 return 1;
274         }
275
276         else{
277             return 0;
278         }
279 }
280
281 int swtoint(void){
282     int i;
283     int n;
284     n=0;
285     for(i=1; i<11; i++){
286         if(BSP_SW(i)){
```

```
287             n+= pow(2, i-1);
288         }
289     }
290     printf("%d\n", n);
291     return n;
292 }
293 
294 int main(void){
295   int i, score;
296   score = 0;
297   srand(time(NULL));
298   SEG7_Decimal(0, 0);
299   while(1){
300     if(BSP_BP(1)==1){
301         printf("BP1 pressed\n");
302         i = rand() % 100;
303         printf("val al atoire1 : %d \n", i);
304         SEG7_Decimal_al(i, 0);
305         alt_busy_sleep(500000);
306     }
307     if(BSP_BP(2)==1){
308             printf("BP2 pressed\n");
309             int j=swtoint();
310             printf("val al atoire2 : %d \n", i);
311             if(i==j){
312                 score += 1;
313                 SEG7_SET(4, szMap[score]);
314                 printf("bravo ! \n");
315             }
316             else
317                 printf("echec \n");
318             //OSTimeDlyHMSM(0, 0, 0, 500);
319         alt_busy_sleep(500000);
320         }
321         if(score==5){
322             break;
323         }
324   }
325   return 0;
326 }
```

code.c

11.3 Module DE10-Standard Computer

```
module DE10_Standard_Computer
//=======================================================
//  PARAMETER declarations
//=======================================================
// #(
// )


//=======================================================
//  PORT declarations
//=======================================================
(
///////////////////////////////////////
// FPGA Pins
///////////////////////////////////////

// Clock pins
input                       CLOCK_50,
input                       CLOCK2_50,
input                       CLOCK3_50,
input                       CLOCK4_50,

// ADC
output                      ADC_CONVST,
output                      ADC_SCLK,
output                      ADC_SDI,
input                       ADC_SDO,

// Audio
input                       AUD_ADCDAT,
inout                       AUD_ADCLRCK,
inout                       AUD_BCLK,
output                      AUD_DACDAT,
inout                       AUD_DACLRCK,
output                      AUD_XCK,

// SDRAM
output      [12: 0] DRAM_ADDR,
output      [ 1: 0] DRAM_BA,
output              DRAM_CAS_N,
output              DRAM_CKE,
output              DRAM_CLK,
output              DRAM_CS_N,
inout       [15: 0] DRAM_DQ,
output              DRAM_LDQM,
output              DRAM_RAS_N,
output              DRAM_UDQM,
output              DRAM_WE_N,

// I2C Bus for Configuration of the Audio and Video-In Chips
output              FPGA_I2C_SCLK,
inout               FPGA_I2C_SDAT,

// 40-pin headers
inout       [35: 0] GPIO,

// Seven Segment Displays
output      [ 6: 0] HEX0,
output      [ 6: 0] HEX1,
output      [ 6: 0] HEX2,
output      [ 6: 0] HEX3,
output      [ 6: 0] HEX4,
output      [ 6: 0] HEX5,
```

```
64
65  // IR
66  input                IRDA_RXD,
67  output               IRDA_TXD,
68
69  // Pushbuttons
70  input        [ 3: 0] KEY,
71
72  // LEDs
73  output       [ 9: 0] LEDR,
74
75  // PS2 Ports
76  inout                PS2_CLK,
77  inout                PS2_DAT,
78
79  inout                PS2_CLK2,
80  inout                PS2_DAT2,
81
82  // Slider Switches
83  input        [ 9: 0] SW,
84
85  // Video-In
86  input                TD_CLK27,
87  input        [ 7: 0] TD_DATA,
88  input                TD_HS,
89  output               TD_RESET_N,
90  input                TD_VS,
91
92  // VGA
93  output       [ 7: 0] VGA_B,
94  output               VGA_BLANK_N,
95  output               VGA_CLK,
96  output       [ 7: 0] VGA_G,
97  output               VGA_HS,
98  output       [ 7: 0] VGA_R,
99  output               VGA_SYNC_N,
100 output               VGA_VS
101 );
102
103 //=======================================================
104 //  REG/WIRE declarations
105 //=======================================================
106
107 wire         [31: 0] hex3_hex0;
108 wire         [15: 0] hex5_hex4;
109
110 assign HEX0 = ~hex3_hex0[ 6: 0];
111 assign HEX1 = ~hex3_hex0[14: 8];
112 assign HEX2 = ~hex3_hex0[22:16];
113 assign HEX3 = ~hex3_hex0[30:24];
114 assign HEX4 = ~hex5_hex4[ 6: 0];
115 assign HEX5 = ~hex5_hex4[14: 8];
116
117 //=======================================================
118 //  Structural coding
119 //=======================================================
120
121
122     Computer_System u0 (
123             // SDRAM
124         .sdram_clk_clk                          (DRAM_CLK),
125         .sdram_addr                             (DRAM_ADDR),
126         .sdram_ba                               (DRAM_BA),
127         .sdram_cas_n                            (DRAM_CAS_N),
128         .sdram_cke                              (DRAM_CKE),
```

```
129         .sdram_cs_n                         (DRAM_CS_N),
130         .sdram_dq                           (DRAM_DQ),
131         .sdram_dqm                          ({DRAM_UDQM,DRAM_LDQM}),
132         .sdram_ras_n                        (DRAM_RAS_N),
133         .sdram_we_n                         (DRAM_WE_N),                    //                      .we_n
134
135         .system_pll_ref_clk_clk             (CLOCK_50),
136         .system_pll_ref_reset_reset         (1'b0),
137
138             // VGA Subsystem
139         .vga_CLK                            (VGA_CLK),
140         .vga_BLANK                          (VGA_BLANK_N),
141         .vga_SYNC                           (VGA_SYNC_N),
142         .vga_HS                             (VGA_HS),
143         .vga_VS                             (VGA_VS),
144         .vga_R                              (VGA_R),
145         .vga_G                              (VGA_G),
146         .vga_B                              (VGA_B),
147
148
149          .video_pll_ref_clk_clk             (CLOCK2_50),
150          .video_pll_ref_reset_reset         (1'b0),
151     );
152
153
154 endmodule
```

Références

[1] Intel FPGA. *DE10-Standard Computer System with Nios II.*