TE608 – Conception de Circuit Numérique 2

Didier Meier

didier.meier@intervenants.efrei.net

2024-2025

Description du projet

LogiGame

Mise en œuvre ludique d'un cœur de contrôleur sur FPGA

Objectifs du projet

- Concevoir en VHDL un cœur de microcontrôleur simple
- Synthétiser puis tester le microcontrôleur sur la carte de développement ARTY (intégrant un FPGA Artix-35T de Xilinx) en utilisant l'IDE Vivado de Xilinx
- Utiliser ce cœur de microcontrôleur pour exécuter un jeu interactif sur une carte de développement FPGA
- Concevoir en VHDL les blocs fonctionnels spécifiques complémentaires nécessaires au bon fonctionnement du jeu
- Utiliser le cœur de microcontrôleur précédemment créé ainsi que les blocs fonctionnels spécifiques pour programmer le jeu
- Synthétiser puis tester le jeu sur la carte de développement ARTY

Objectifs pédagogiques

- À travers le projet LogiGame, vous êtes amenés à concevoir un cœur de microcontrôleur en VHDL qui sera utilisé pour exécuter un jeu interactif.
- Ce jeu met en œuvre des notions fondamentales : datapath, logique séquentielle, mémoires, contrôleur d'instructions, temporisation et interface avec des éléments physiques de la carte FPGA (LEDs, boutons, switches).

Description du jeu LogiGame

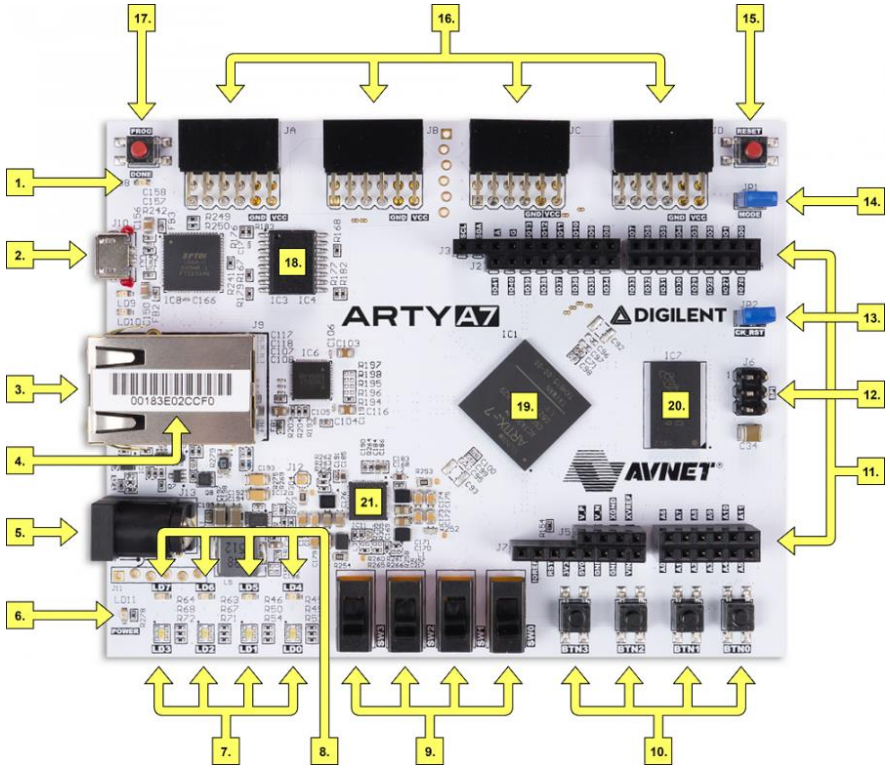
- LogiGame est un jeu de réflexe dans lequel l'utilisateur doit réagir rapidement à l'allumage d'une LED en rouge, en vert ou en bleu.
- En fonction du niveau de difficulté choisi, l'utilisateur dispose d'un temps limité pour appuyer sur le bon bouton correspondant à la couleur de la LED allumée.
- Chaque bonne réponse augmente le score affiché pour tenter d'atteindre une suite de 15 tentatives réussies.
- En cas d'erreur ou si le temps est dépassé pour chaque tentative, le jeu s'arrête et le score final reste affiché.

Fonctionnement général du système

- Un cœur de microcontrôleur :
  - Une unité arithmétique et logique (UAL)
  - Un chemin de données (datapath)
  - Des mémoires internes (buffers, mémoire d'instruction…)
- Des fonctionnalités complémentaires :
  - Un compteur de score (enregistrement des bonnes réponses)
  - Un minuteur ajustable (délais de réponse selon la difficulté)
  - Un contrôleur d'état (interprétation des instructions et pilotage de l'exécution du jeu)
  - Un gestionnaire d'entrées/sorties (gestions des switchs, boutons poussoirs, leds…)

Intégration sur la carte de développement ARTY

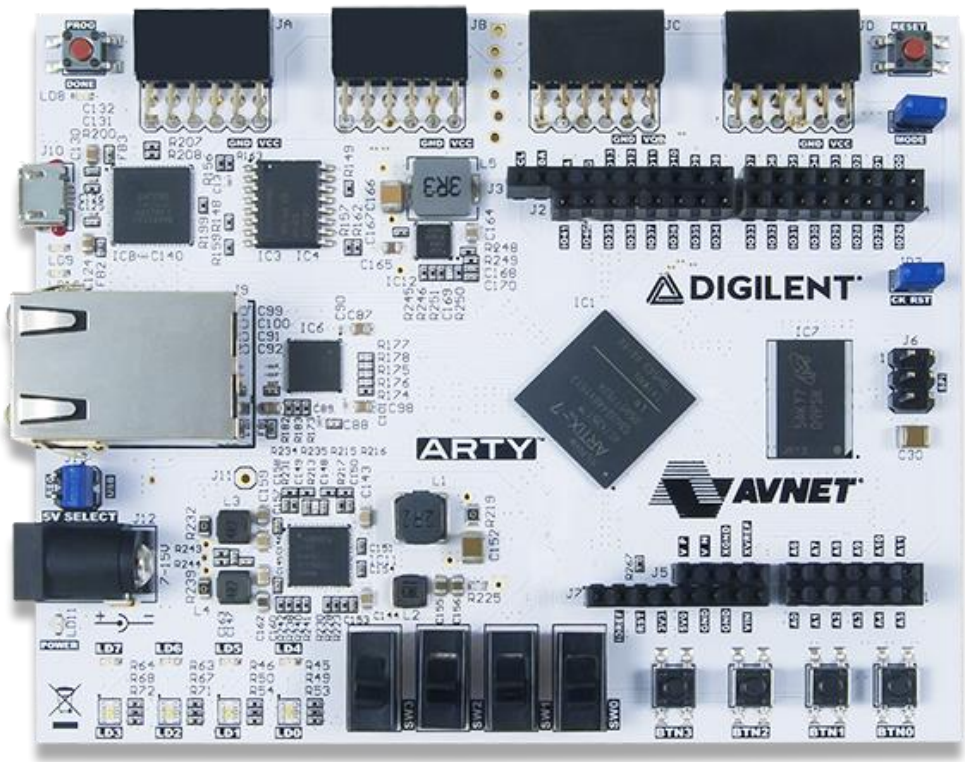
- La carte de développement



Callout	Description	Callout	Description	Callout	Description
1	FPGA programming DONE LED	8	User RGB LEDs	15	chipKIT processor reset
2	Shared USB JTAG / UART port	9	User slide switches	16	Pmod connectors
3	Ethernet connector	10	User push buttons	17	FPGA programming reset button
4	MAC address sticker	11	Arduino/chipKIT shield connectors	18	SPI flash memory
5	Power jack for optional external supply	12	Arduino/chipKIT shield SPI connector	19	Artix FPGA
6	Power good LED	13	chipKIT processor reset jumper	20	Micron DDR3 memory
7	User LEDs	14	FPGA programming mode	21	Dialog Semiconductor DA9062 power supply

Interface avec la carte de développement FPGA

- Led LD3 :
  - Utilisée comme LED de jeu principale (clignotement stimulus)
- Boutons BTN1, BTN2 et BTN3 :
  - Permet d'indiquer la couleur de LD3 :
    - ✓ BTN1 : couleur BLEUE
    - ✓ BTN2 : couleur VERTE
    - ✓ BTN3 : couleur ROUGE
- Switches SW3 et SW2 :
  - Utilisés pour choisir le niveau de difficulté
- Bouton BTN0 :
  - Permet de réinitialiser le jeu et de le lancer
- Led LD0 :
  - Indique le niveau de réussite final (vert, orange, rouge) :
    - ✓ Vert : score de 15
    - ✓ Orange : score entre 7 et 14
    - ✓ Rouge : score entre 0 et 6

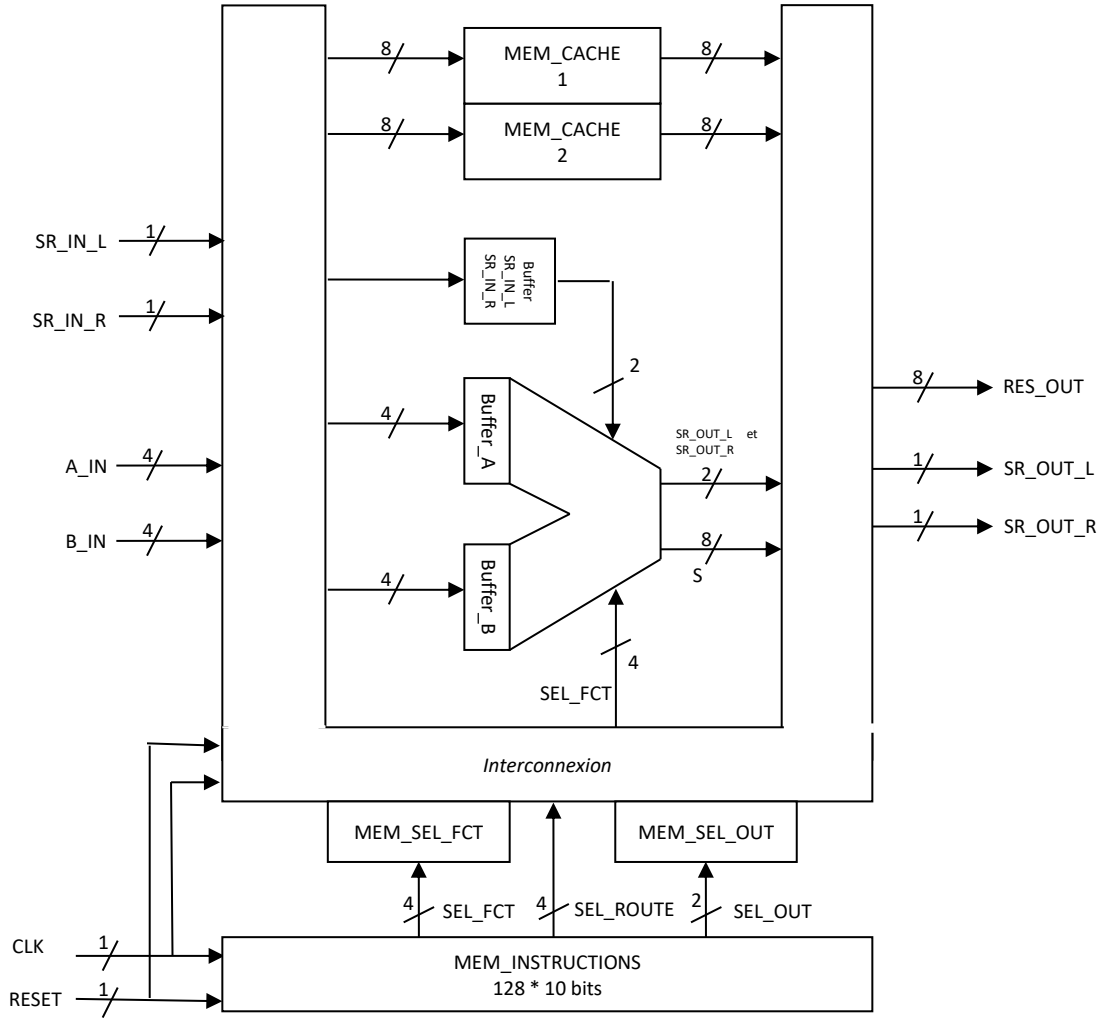


Projet

Conception, simulation et synthèse d'un microcontrôleur à l'aide du VHDL dans un composant à architecture programmable

Présentation générale de l'architecture cible

- L'objectif de ce projet consiste à réaliser un cœur de microcontrôleur intégrant :
  - Une unité de traitements arithmétiques et logiques
  - Des mémoires internes de calcul
  - Une mémoire d'instructions
- Ce cœur de microcontrôleur fonctionnera sur 4 bits signés pour les entrées et sur 8 bits pour les sorties et les mémoires internes.
- A partir d'opérations de base, il est possible de réaliser l'ensemble des fonctions logiques et des opérations arithmétiques à l'aide d'automates (successions d'instructions).
  - Les fonctions plus « complexes » n'étant qu'une succession de fonctions simples avec parfois la mémorisation de résultats intermédiaires.
  - Il est donc nécessaire d'adjoindre à la structure de base
    - ✓ Une mémoire interne permettant de stocker des résultats intermédiaires
    - ✓ Une mémoire interne permettant de stocker les instructions successives à réaliser
- Les derniers travaux consisteront à réaliser trois automates et de les implémenter dans la mémoire d'instructions.
  - Pour cette dernière partie, vous serez amenés à implémenter les trois automates permettant de piloter l'UAL afin de réaliser les fonctions choisies.



Fonctions réalisées par l'UAL

SEL_FCT[3]	SEL_FCT[2]	SEL_FCT[1]	SEL_FCT[0]	Significations
0	0	0	0	nop (no operation) S = 0 | SR_OUT_L = 0 et SR_OUT_R = 0
0	0	0	1	S = A | SR_OUT_L = 0 et SR_OUT_R = 0
0	0	1	0	S = B | SR_OUT_L = 0 et SR_OUT_R = 0
0	0	1	1	S = not A | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	0	0	S = not B | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	0	1	S = A and B | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	1	0	S = A or B | SR_OUT_L = 0 et SR_OUT_R = 0
0	1	1	1	S = A xor B | SR_OUT_L = 0 et SR_OUT_R = 0
1	0	0	0	S = Déc. droite A sur 4 bits(avec SR_IN_L) | SR_IN_L pour le bit entrant et SR_OUT_R pour le bit sortant
1	0	0	1	S = Déc. gauche A sur 4 bits (avec SR_IN_R) | SR_IN_R pour le bit entrant et SR_OUT_L pour le bit sortant
1	0	1	0	S = Déc. droite B sur 4 bits (avec SR_IN_L) | SR_IN_L pour le bit entrant et SR_OUT_R pour le bit sortant
1	0	1	1	S = Déc. gauche B sur 4 bits (avec SR_IN_R) | SR_IN_R pour le bit entrant et SR_OUT_L pour le bit sortant
1	1	0	0	S = A + B addition binaire avec SR_IN_R comme retenue d'entrée
1	1	0	1	S = A + B addition binaire sans retenue d'entrée
1	1	1	0	S = A – B soustraction binaire
1	1	1	1	S = A * B multiplication binaire

Les mémoires

- Les mémoires Buffer_A et Buffer_B
  - Les mémoires Buffer_A, Buffer_B permettent de stocker les données directement liées au cœur de l'UAL, c'est-à-dire à la sous-fonction arithmétique et logique.
  - Elles seront chargées (activées sur front montant de l'entrée clk) suivant les valeurs de l'entrée SEL_ROUTE.

- Les mémoires MEM_SR_IN_L et MEM_SR_IN_R permettent de stocker les valeurs des retenues d'entrées.
  - Elles sont systématiquement mémorisées à chaque front montant de l'horloge CLK.

- La mémoire MEM_SEL_FCT permet de mémoriser la fonction arithmétique ou logique à réaliser.
  - Elle est systématiquement chargée à chaque front montant d'horloge.

- Les mémoires MEM_CACHE_1 et MEM_CACHE_2
  - Les mémoires MEM_CACHE_1 et MEM_CACHE_2 permettent de stocker les données des entrées A_IN et B_IN ou les résultats intermédiaires de la sortie S, c'est-à-dire à la sortie de la sous-fonction arithmétique et logique.
    - ✓ Il est à noter que les sorties SR_OUT_L et SR_OUT_R ne peuvent pas être mémorisées.
    - ✓ Ces deux mémoires seront chargées (activées sur front montant de l'entrée CLK) suivant les valeurs de l'entrée SEL_ROUTE.

- La mémoire MEM_SEL_OUT
  - La mémoire MEM_SEL_OUT permet de stocker les données de l'entrée SEL_OUT.
  - Elle est systématiquement chargée à chaque front montant de l'horloge CLK.

- La mémoire MEM_INSTRUCTIONS
  - La mémoire MEM_INSTRUCTIONS permet de stocker les instructions successives à réaliser
  - Son pointeur est systématiquement indenté à chaque front montant de l'horloge CLK

- SEL_ROUTE permet de définir le transfert de données qui sera effectué lors du prochain cycle horloge (prochain front montant de l'horloge).
  - Elle ne doit donc pas être mémorisée comme MEM_SEL_FCT et MEM_SEL_OUT

Les interconnexions et le routage des données

SEL_ROUTE[3]	SEL_ROUTE[2]	SEL_ROUTE[1]	SEL_ROUTE[0]	Significations
0	0	0	0	Stockage de l'entrée A_IN dans Buffer_A
0	0	0	1	Stockage de MEM_CACHE_1 dans Buffer_A (4 bits de poids faibles)
0	0	1	0	Stockage de MEM_CACHE_1 dans Buffer_A (4 bits de poids forts)
0	0	1	1	Stockage de MEM_CACHE_2 dans Buffer_A (4 bits de poids faibles)
0	1	0	0	Stockage de MEM_CACHE_2 dans Buffer_A (4 bits de poids forts)
0	1	0	1	Stockage de S dans Buffer_A (4 bits de poids faibles)
0	1	1	0	Stockage de S dans Buffer_A (4 bits de poids forts)
0	1	1	1	Stockage de l'entrée B_IN dans Buffer_B
1	0	0	0	Stockage de MEM_CACHE_1 dans Buffer_B (4 bits de poids faibles)
1	0	0	1	Stockage de MEM_CACHE_1 dans Buffer_B (4 bits de poids forts)
1	0	1	0	Stockage de MEM_CACHE_2 dans Buffer_B (4 bits de poids faibles)
1	0	1	1	Stockage de MEM_CACHE_2 dans Buffer_B (4 bits de poids forts)
1	1	0	0	Stockage de S dans Buffer_B (4 bits de poids faibles)
1	1	0	1	Stockage de S dans Buffer_B (4 bits de poids forts)
1	1	1	0	Stockage de S dans MEM_CACHE_1
1	1	1	1	Stockage de S dans MEM_CACHE_2

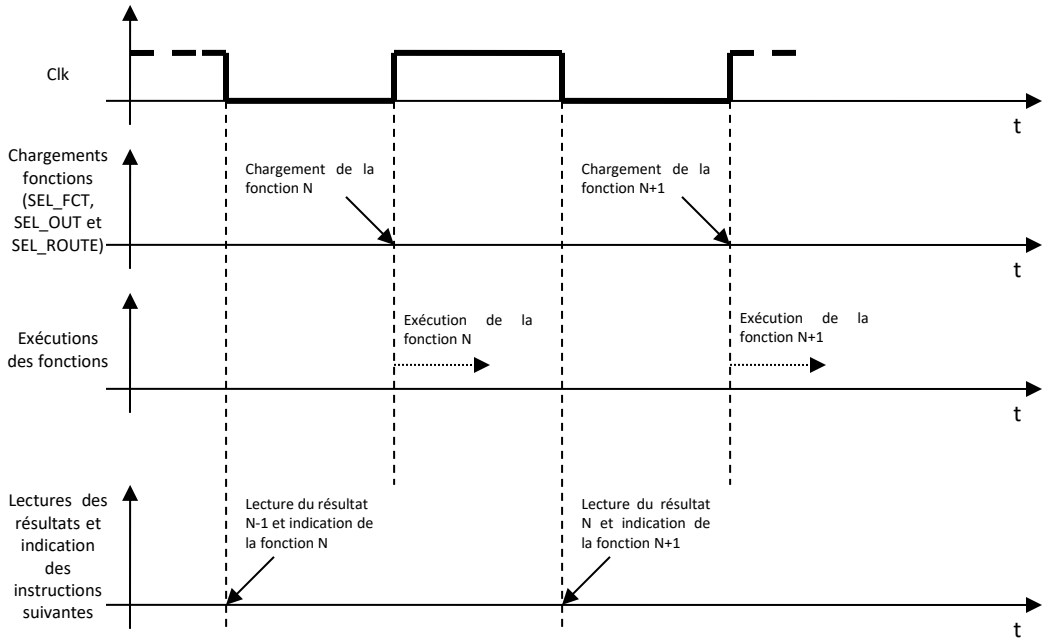
SEL_OUT[1]	SEL_OUT[0]	Significations
0	0	Aucune sortie : RES_OUT = 0
0	1	RES_OUT = MEM_CACHE_1
1	0	RES_OUT = MEM_CACHE_2
1	1	RES_OUT = S

Synchronisation temporelle

- Le diagramme ci-contre présente le fonctionnement temporel que devra respecter le montage
  - En cas de pilotage externe (sans mémoire d'instructions) pour tests
  - En interne avec la mémoire d'instructions

- Les composants externes utilisant cette UAL modifient les valeurs des entrées sur les fronts descendants de l'horloge CLK.
  - Au prochain font montant, l'UAL exécute les instructions.
  - Les résultats sont lus lors du front descendant suivant (tout en indiquant la nouvelle fonction à réaliser).

- L'entrée RESET permet d'initialiser de manière asynchrone toutes les mémoires à 0



Les instructions à coder pour tests et validations

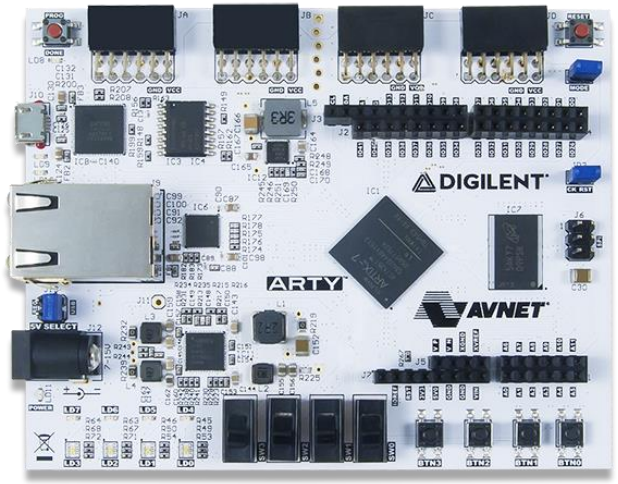
- Vous aurez à implémenter 3 fonctions spécifiques dans la mémoire à instructions :
  1) RES_OUT_1 = (A mult. B) *(RES_OUT_1 sur 8 bits)*
  2) RES_OUT_2 = (A add. B) xnor A *(xnor sur les 4 bits de poids faibles – RES_OUT_2 sur 4 bits)*
  3) RES_OUT_3 = (A_0 and B_1) or (A_1 and B_0) *(RES_OUT_3 sur le bit de poids faible)*

- On considère les entrées A, B, SR_IN_L et SR_IN_R stables toute la durée du calcul :
  - Ce qui, en réalité, ne sera pas toujours le cas…

- La sortie RES_OUT restera à 0 tant que le résultat final du calcul ne sera pas disponible (pas de résultats intermédiaires sur RES_OUT)
  - Une fois le résultat calculé, il sera maintenu sur la sortie RES_OUT jusqu'au prochain lancement du calcul
    - ✓ Pour les tests sur la carte FPGA, un signal spécifique sera également passé de 0 (calcul en cours) à 1 (calcul terminé) pour indiquer que le résultat RES_OUT est disponible

Définition de l'environnement de travail

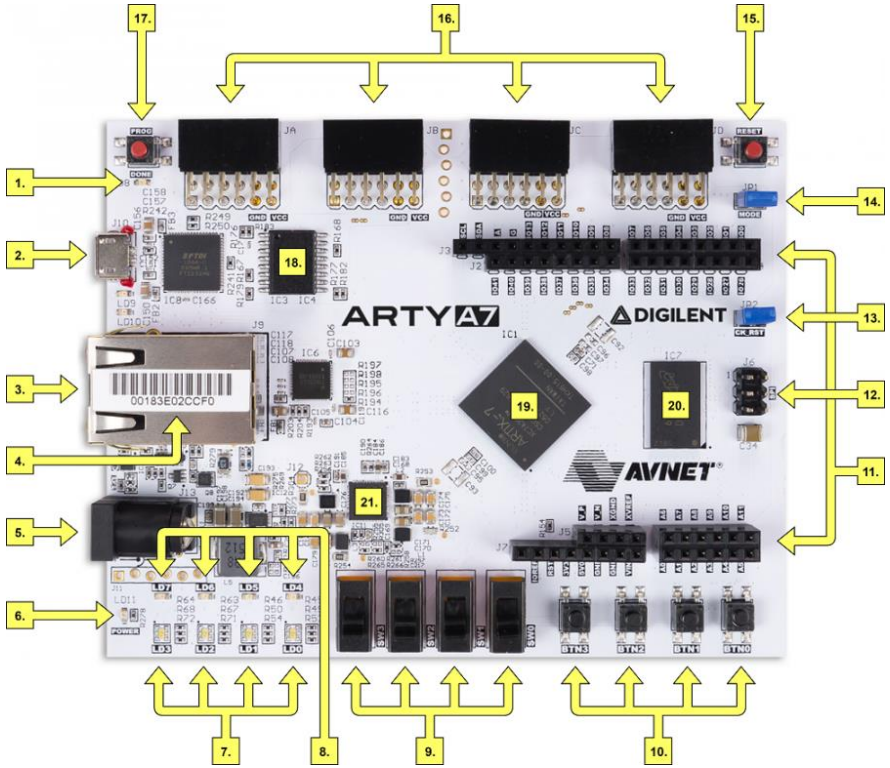
- Environnement de développement et de simulation :
  - Environnements GHDL & GTKWave
  - EDA Playground
  - Xilinx Vivado 2018.2 (ou version ultérieure)



- Maquette de développement :
  - Carte ARTY de Digilent
    - ✓ Xilinx Artix-35T FPGA
  - La maquette de développement n'est pas nécessaire pour la réalisation du projet
    - ✓ Néanmoins, elle sera utilisée pour réaliser les implémentations lors de la dernière séance de projet

Intégration sur la carte de développement ARTY

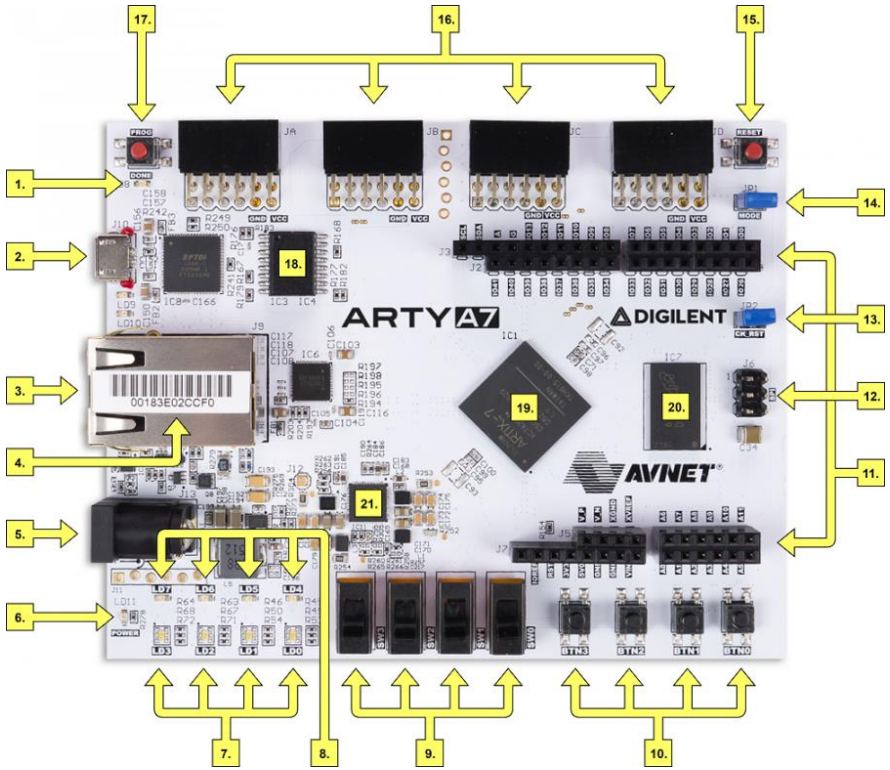
- La carte de développement



Callout	Description	Callout	Description	Callout	Description
1	FPGA programming DONE LED	8	User RGB LEDs	15	chipKIT processor reset
2	Shared USB JTAG / UART port	9	User slide switches	16	Pmod connectors
3	Ethernet connector	10	User push buttons	17	FPGA programming reset button
4	MAC address sticker	11	Arduino/chipKIT shield connectors	18	SPI flash memory
5	Power jack for optional external supply	12	Arduino/chipKIT shield SPI connector	19	Artix FPGA
6	Power good LED	13	chipKIT processor reset jumper	20	Micron DDR3 memory
7	User LEDs	14	FPGA programming mode	21	Dialog Semiconductor DA9062 power supply

Intégration sur la carte de développement ARTY

- Réalisation de tests sur la carte de développement :
  - Fonctionnement à 100 MHZ (clk)
  - A = B
    - ✓ Swt(3-0) : User slide switches
  - SR_IN_L et SR_IN_R = 0 (non utilisés pour le test)
  - Reset global
    - ✓ btn(0) : User push buttons
  - Lancement des calculs :
    - ✓ btn(1) : RES_OUT_1
    - ✓ btn(2) : RES_OUT_2
    - ✓ btn(3) : RES_OUT_3
  - Résultats de calculs :
    - ✓ 8 leds (couleur = rouge)
    - ✓ Résultat disponible : 8ème led avec du vert (LSB)
    - ✓ SR_OUT_L : 5ème led avec du bleu
    - ✓ SR_OUT_R : 6ème led avec du bleu



Description de l'entité globale et fichier de contraintes

- Pour l'implémentation sur FPGA, vous devrez respecter la description de l'entité globale afin que les ports d'entrées et de sorties puissent correspondre avec la carte de développement :
  - La description de l'entité est disponible en téléchargement sur votre espace Moodle

- Afin de permettre de réaliser les tests sur carte FPGA, vous devrez utiliser le fichier de contraintes spécifique à la carte de développement :
  - Le fichier de contraintes est disponible en téléchargement sur votre espace Moodle

Projet

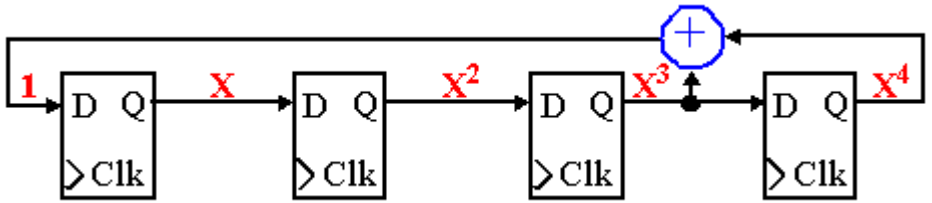
Conception, simulation et synthèse d'un jeu LogiGame à l'aide du microcontrôleur et de codes spécifiques en VHDL dans un composant à architecture programmable

Génération d'un générateur de séquence pseudo aléatoire

- Objectif :
  - Synthétiser puis simuler un registre à décalage à rétroaction linéaire (LFSR) sur 4 bits basé sur le polynôme :

$$X^4 + X^3 + 1$$

  - Ce LFSR génère une valeur pseudo-aléatoire à chaque exécution, qui pourra être utilisée comme stimulus couleur (ex : modulo 3 pour obtenir rouge, vert ou bleu).



Génération d'un générateur de séquence pseudo aléatoire

- 1ère partie :
  - Synthétiser le générateur de la séquence pseudo-aléatoire directement en VHDL avant de le tester :
    - ✓ Fréquence de génération de la séquence :
      - F_{LFSR} = 1 kHz
  - Spécifications :
    - ✓ Entrée clk : horloge principale (100 MHz)
    - ✓ Entrée reset : réinitialisation du registre à une valeur initiale non nulle « 1011 »
    - ✓ Entrée enable : active l'évolution du LFSR à chaque front montant
    - ✓ Sortie rnd : vecteur de 4 bits représentant la valeur pseudo-aléatoire courante
  - Fonctionnement attendu :
    - ✓ Le LFSR implémente le polynôme $X^4 + X^3 + 1$
    - ✓ Feedback : XOR entre les bits 3 et 2
    - ✓ La séquence doit parcourir 15 états avant de boucler
  - Résultat attendu
    - ✓ À chaque cycle d'horloge avec enable = '1', une nouvelle valeur pseudo-aléatoire est produite.

Génération d'un générateur de séquence pseudo aléatoire

- 2ème partie (à la fin du projet – remplacement du LFSR en VHDL) :
  - Utiliser le cœur de microcontrôleur comme générateur de la séquence pseudo-aléatoire avant de le tester :
    - ✓ Codage de la mémoire d'instruction :
      - Codage des instructions successives gérant la génération d'une valeur pseudo-aléatoire
    - ✓ Codage de l'automate de pilotage du générateur :
      - Boucle des instructions de la mémoire d'instruction permettant de générer toutes les valeurs pseudo-aléatoires successives
    - ✓ Valeur initiale du vecteur interne :
      - « 1011 »
    - ✓ Fréquence du cœur du microcontrôleur
      - FMCU = 1 kHz
  - Spécifications
    - ✓ Le registre pseudo-aléatoire est simulé via instructions VHDL
    - ✓ Les données sont stockées dans MEM_CACHE_1, Buffer_A, Buffer_B, etc.
    - ✓ Le feedback est obtenu via une opération XOR entre les bits 3 et 2
    - ✓ La séquence est construite via : masquage, décalage, XOR, recomposition
  - Résultat attendu :
    - ✓ La valeur pseudo-aléatoire évolue à chaque exécution du bloc d'instructions LFSR.
    - ✓ La séquence obtenue est identique à celle d'un LFSR matériel de type $X^4 + X^3 + 1$.

Génération d'un minuteur programmable permettant de gérer la difficulté du jeu

- Objectif :
  - Créer un minuteur programmable, contrôlé par 2 bits (SW[3:2]) pour ajuster le temps de réponse autorisé :
    - ✓ 00 → Très facile (temps long)
    - ✓ 01 → Facile
    - ✓ 10 → Moyen
    - ✓ 11 → Difficile (temps court)
  - Ce module déclenche un timeout lorsqu'un nombre prédéfini de cycles est écoulé.

- Principe de fonctionnement :

SW[3:2]	Niveau	Nombre de cycles (ex.)	Durée approx. à 100 MHz
00	Très facile	100 000 000	1 seconde
01	Facile	50 000 000	0,5 seconde
10	Moyen	25 000 000	0,25 seconde
11	Difficile	12 500 000	0,125 seconde

Génération d'un minuteur programmable permettant de gérer la difficulté du jeu

- Spécifications
  - Entrée clk : 100 MHz
  - Entrée reset : remise à zéro du compteur et du signal timeout
  - Entrée start : lancement du décompte à partir du moment déclenché
  - Entrée sw_level : niveau de difficulté (2 bits)
  - Sortie timeout : passe à '1' lorsque le délai programmé est atteint

- Paramètres de délais à implémenter :
  - Niveau 00 : 1s (100 000 000 cycles)
  - Niveau 01 : 0,5s (50 000 000 cycles)
  - Niveau 10 : 0,25s (25 000 000 cycles)
  - Niveau 11 : 0,125s (12 500 000 cycles)

- Résultat attendu :
  - Le signal timeout passe à '1' une seule fois à la fin du délai défini, puis reste à '0' tant qu'un nouveau start n'est pas activé.

Génération d'un compteur de score pour le jeu

- Objectif :
  - Concevoir un compteur de score en VHDL pour le jeu LogiGame, capable de :
  - Incrémenter le score à chaque bonne réponse (valid_hit = '1')
  - Conserver la valeur atteinte en cas de mauvaise réponse (valid_hit = '0')
  - Générer un signal game_over lorsque le score atteint 15 ou en cas d'erreur

- Spécifications :
  - Entrée clk : horloge système
  - Entrée reset : remise à zéro du score
  - Entrée valid_hit : indiquant la réussite (1) ou l'échec (0)
  - Sortie score : score courant codé sur 4 bits
  - Sortie game_over : signal indiquant la fin du jeu

Génération d'un compteur de score pour le jeu

- Comportement attendu :
  - Le score commence à 0 après reset
  - Chaque valid_hit = '1' incrémente le score jusqu'à 15
  - Une seule mauvaise réponse (valid_hit = '0') met fin à la partie
  - Une fois game_over = '1', le score reste figé

- Astuces de modélisation :
  - Utiliser un registre score_reg (type unsigned)
  - Gérer next_score en logique combinatoire
  - Ne mettre à jour score_reg que si game_over = '0'
  - game_over est à '1' si valid_hit = '0' ou score = 15

Génération d'un vérificateur de résultat pour le jeu

- Objectif :
  - Créer un composant VHDL qui valide si le joueur appuie sur le bon bouton dans le temps imparti.

- Spécifications :
  - Entrée clk : horloge système
  - Entrée reset : réinitialisation du module
  - Entrée timeout : signal de fin de délai
  - Entrée led_color : couleur affichée sur LD3 (3 bits, R=100, G=010, B=001)
  - Entrées btn_r, btn_g, btn_b : boutons de réponse (BTN1, BTN2, BTN3)
  - Sortie valid_hit : passe à '1' si la bonne réponse a été donnée dans les temps

Génération d'un vérificateur de résultat pour le jeu

- Comportement attendu :
  - Si LD3 = "100" → seul BTN1 est correct
  - Si LD3 = "010" → seul BTN2 est correct
  - Si LD3 = "001" → seul BTN3 est correct
  - valid_hit = '1' uniquement si le bon bouton est pressé avant que timeout = '1'
  - Un seul appui est comptabilisé :
    - ✓ Réponse unique par round

- Astuce :
  - Utiliser un registre user_pressed pour bloquer toute réponse multiple après une réponse détectée.

Synthèse du contrôleur de jeu (automate à états finis)

- **Objectif :**
  - Réaliser un contrôleur de jeu VHDL pilotant la logique globale de LogiGame :
    - ✓ Génération pseudo-aléatoire du stimulus
    - ✓ Lancement du timer en fonction du niveau de difficulté
    - ✓ Vérification de la réponse du joueur
    - ✓ Incrément du score ou fin de partie

- **Composants utilisés :**
  - ✓ LFSR4 : générateur pseudo-aléatoire (stimulus couleur)
  - ✓ DifficultyTimer : minuterie avec niveau SW[3:2]
  - ✓ ScoreCounter : compteur de score avec signal de fin
  - ✓ ResponseChecker : vérification bouton/couleur/temps

Synthèse du contrôleur de jeu (automate à états finis)

- Automate de contrôle (FSM) :
  - IDLE : attente du signal start
  - NEW_ROUND : génère un nouveau stimulus et démarre le timer
  - WAIT_RESPONSE : attend une réponse correcte ou un timeout
  - END_GAME : verrouille le jeu en cas de défaite ou score maximal
- Synchronisation :
  - LFSR activé une seule fois par manche (enable)
  - Timer relancé à chaque round via signal start_timer
  - Une seule réponse valide comptabilisée via valid_hit
- Résultat attendu :
  - Affichage LED cohérent
  - Score mis à jour après chaque bonne réponse
  - Fin de jeu détectée proprement

Intégration globale

- Objectif :
  - Réaliser le fichier top-level complet pour la synthèse du jeu LogiGame sur carte ARTY.
    Ce fichier instancie le contrôleur principal et connecte tous les signaux aux entrées/sorties réelles de la ca

- Signaux principaux :
  - clk : CLK100MHZ (horloge 100 MHz)
  - btn(0–3) : boutons de contrôle et réponse (start, rouge, vert, bleu)
  - sw(3:2) : sélection du niveau de difficulté
  - led : sortie score (4 bits)
  - led3_r/g/b : LED RVB pour affichage du stimulus

Intégration globale

- Simulation (testbench) :
  - Appui simulé sur le bouton de démarrage
  - Réponse correcte à un stimulus simulé
  - Timeout simulé pour vérifier la fin de partie
- Ce que vous devez observer :
  - Affichage correct de la couleur sur LD3
  - Incrémentation du score sur les LED standard
  - Blocage du jeu après une mauvaise réponse ou score maximal
- Conseils :
  - Vérifiez les connexions physiques entre les boutons/LEDs et les modules internes
  - Sur la carte réelle : contraindre les broches via le fichier .xdc pour correspondre aux entrées/sorties utilisées

Livrables attendus et évaluation

- Définition des livrables (rapports et archives) et résultats attendus (validations)
  - A la fin de la période dédiée à la réalisation du projet, vos résultats seront évalués en combinant :
    - ✓ Une validation technique (intégration du montage dans un composant programmable et réalisation d'une séquence automatique de tests).
    - ✓ Une validation théorique dans l'environnement de développement (à l'aide de vos archives de projets et de votre rapport technique) via différentes simulations.
    - ✓ L'évaluation de votre rapport technique du projet.
  - Il est donc primordial de fournir les différents éléments demandés.
    - ✓ Aucun retard ne sera toléré.
    - ✓ Tout élément manquant lors de la remise des archives pourra nuire à l'évaluation de votre projet.
      - Soyez donc vigilent à respecter scrupuleusement les éléments demandés.

- Livraisons de documents et d'archives
  - A la fin de la période impartie à la réalisation du projet, il vous sera demandé de fournir :
    - ✓ 1 archive ZIP au format « noms_des_élèves_du_binôme ».zip contenant
      - L'ensemble de votre(vos) dossier(s) du projet
      - 1 fichier au format « noms_des_élèves ».txt contenant un résumé de chacune de vos entités en VHDL
      - 1 rapport détaillé du mini-projet au format électronique (.doc ou .pdf).

- L'archive devra être déposée sur Moodle avant la fin du délai de réalisation du projet
  - Aucun retard ne sera toléré
  - Le transfert de ces fichiers par courriel n'est pas accepté