Compte Rendu FPGA1

Name: Hongbo Jiang StudentID: 3602103

Name: Songlin Li StudentID: 3770906

Partie 1 : Synthèse VHDL sur FPGA

I) Prise en Main de la carte et des outils

1) Présentation de la carte Digilent Nexys4

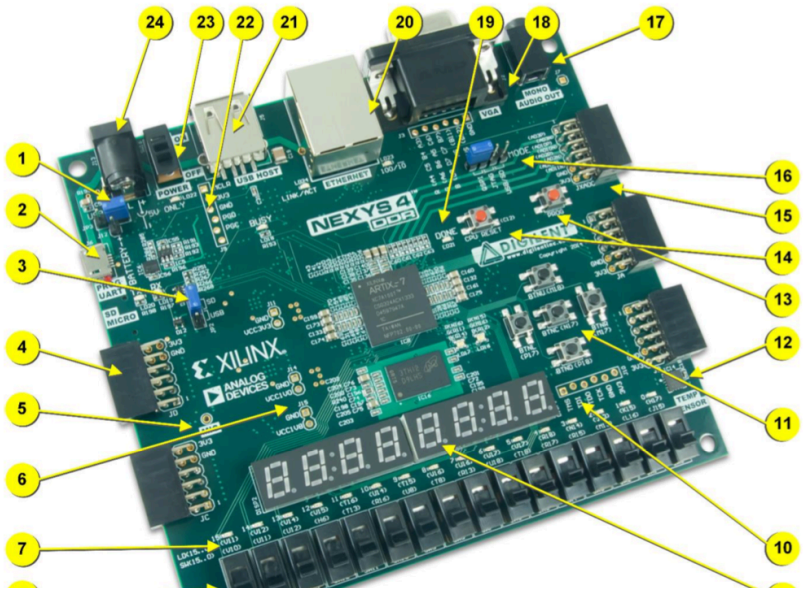


Figure 1. Nexys4 DDR board features.

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod port (XADC)
4	Pmod port(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

Nexys4DDRTM FPGA Board Reference Manual(https://reference.digilentinc.com/_media/nexys4-ddr:nexys4ddr_rm.pdf)

2) Création d'un projet avec le logiciel Vivado

Dans la première partie, après avoir étudié la présentation de la carte Digilent Nexys4, on a créé un projet avec Vivado pour mieux connaître l'utilisation de ce logiciel.

3) Création d'un module VHDL

Pour créer un module VHDL , on ajoute la source en choisissant **Add or Create Design Sources** et crée un fichier Test. Et puis on indique 3 ports d'entrée appelés SW2 , SW1, SW0 et un port de sortie sur 3 bits LED(2 :0). L'architecture du module Test est que la LED(0) prend la valeur de SW0, LED(1) prend la valeur de SW1 et LED(2) prend la sortie d'un ET logique entre les 3 interrupteurs.

• Lorsque SW0 est actif, LED0 l'est aussi.
• Lorsque SW1 est actif, LED1 l'est aussi.
• Lorsque SW0 et SW1 sont actifs, LED0, LED1 et LED2 le sont aussi.

L'entité décrite par le code VHDL est bien le système attendu. Afin de l'implémenter sur la carte, il faut ajouter un fichier de contrainte afin d'associer ses différentes entrées, aux interrupteurs de la carte et sa sortie, aux leds de la carte. Une fois le chier ajouté, il faut lancer la synthèse, l'implémentation sur la carte et générer le flux de données pour configurer la carte.

4) Testbench et simulation avec Modelsim

Et puis on crée un fichier Testbench en cliquant Add or Create Simulation Sources et vérifie on a coché l'option Copy sources into project.

```
entity Test is
    Port ( SW2 : in STD_LOGIC;
           SW1 : in STD_LOGIC;
           SW0 : in STD_LOGIC;
           LED : out STD_LOGIC_VECTOR (2 downto 0));
end Test;

architecture Behavioral of Test is

begin
    LED(0) <= SW0;
    LED(1) <= SW1;
    LED(2) <= SW2 and SW1 and SW0;

end Behavioral;
```

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

entity TB_Test is
end TB_Test;

architecture Behavioral of TB_Test is

-- Signaux pour le port map du module à tester
signal SW0 : STD_LOGIC;
signal SW1 : STD_LOGIC;
signal SW2 : STD_LOGIC;
signal LED : STD_LOGIC_VECTOR (2 downto 0);

begin

-- Instanciation du Module Test
10: entity work.Test
port map(SW0,SW1,SW2,LED);

-- Evolution des Entrees
SW0 <= '0', '1' after 200 ns, '0' after 800 ns;
SW1 <= '0', '1' after 400 ns;
SW2 <= '0', '1' after 600 ns;

end Behavioral;
```

Ensuite on lance la simulation comportementale et corrige les erreurs à l'aide des informations de la Console.

5) Implémentation sur la carte FPGA

Après avoir ajouté le fichier Test_Nexys4DDR.xdc (Add or Creat Constraints) dans le projet et avoir modifié la correspondance des noms, on lance l'implémentation : **Run Synthesis -> Run Implementation -> Generate Bitstream -> Open Hardware Manager**. Et donc le FPGA est programmé après toutes cettes démarches.

II) Cas d'Etudes – Synthèse VHDL

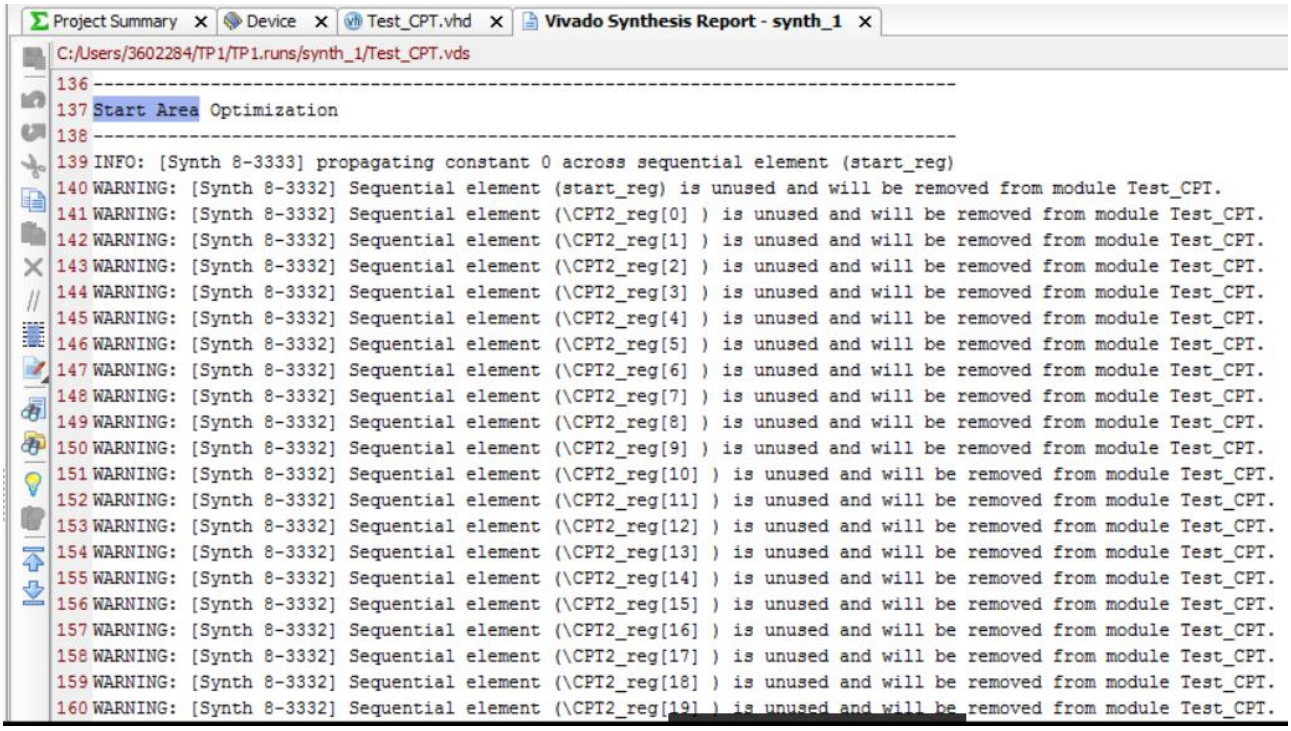
Le but de cette partie est de voir en quoi les architectures d'applications sont bien ou mal interprétées par l'outil de synthèse d'ISE et comment on modifie le code pour qu'il puisse être compréhensible et synthétisable par l'outil en cas échéant.

1) Compteur Imbriqués

On ajoute au projet les fichiers Test_CPT.vhd (Add or Create Design Sources) qui propose deux compteurs imbriqués dont la valeur des poids forts est affichée sur les LEDs, et Test_CPT_Nexys4DDR.xdc (Add or Creat Constraints) , puis implémente le design sur Generate Bitstream.

Il s'agit d'un système comportant deux compteurs qui s'incrémentent à chaque front d'horloge : un qui s'active sur un signal start et l'autre qui met le signal start à 0 lorsqu'il atteint sa valeur limite. En l'exécutant sur la carte et en associant les états logiques des bits du signal LED à des LEDs de la carte, on devrait les voir a cher la valeur des 4 derniers bits du compteur 2 : on utilise les 4 derniers bits pour diviser la fréquence de l'horloge principale.
En le testant sur la carte, on obtient bien ce qui était attendu à la différence que le compteur repart de 0 lorsqu'il a atteint sa valeur maximale alors qu'il devrait s'arrêter.
Lors de l'implémentation sur la carte du système décrit par le code VHDL, les lignes qui ne sont pas utilisées sont supprimées par Vivado pour optimiser l'implémentation.
D'après le message d'information, le signal start est toujours à 0 ce qui est normal car la valeur limite du compteur 1 est trop grande et n'est donc jamais atteinte : en e et, dans le code VHDL initial, elle est de 70 000 000 alors que le compteur ne peut compter que jusqu'à 20 000 000.

Mais on a failli à finir l'implémentation. Dans la console on ouvre le Vivado Synthesis Report pour analyser les messages INFO et WARNING.

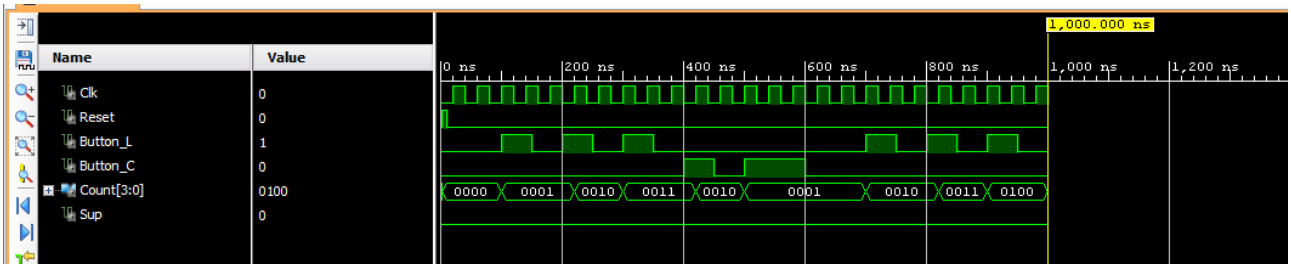


Aussi on analyse le schéma RTL pour trouver les problems. Et puis on modifie le code et diminue la valeur limite de cpt, comme ci-dessous:

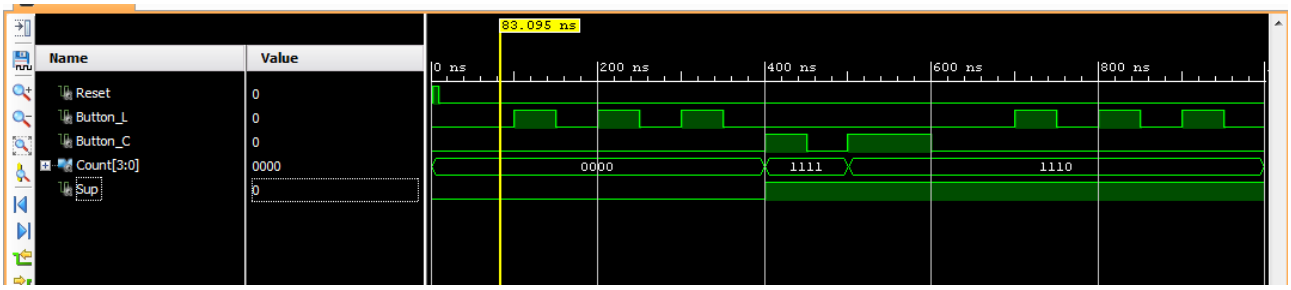
```
    -------------------------
    process(Clk,Reset)

    begin

        -- Reset Asynchrone
        if Reset = '1' then Cpt <= 0; start <='0';

        -- Au Front d'Horloge...
        elsif rising_edge(Clk) then

            Cpt <= Cpt + 1; -- Incrémentation CPT

            -- Si on Arrive à la Valeur Limite
            if Cpt = 7000000 then
                start <= not start;      -- On Inverse le Niveau
                Cpt <= 0;                -- RAZ Synchrone de CPT

            end if;

        end if;
    end process;
```

2) Compteur d'Implusions

On veut maintenant réaliser un compteur qui va incrémenter si on appuie le bouton left et décrémenter si on appuie le bouton center. On a ajouté le code fourni dans la source et lancé la simulation comportementale :



On peut obtenir le résultat idéal. Et puis on lance la simulation après synthèse, il y a des erreurs :

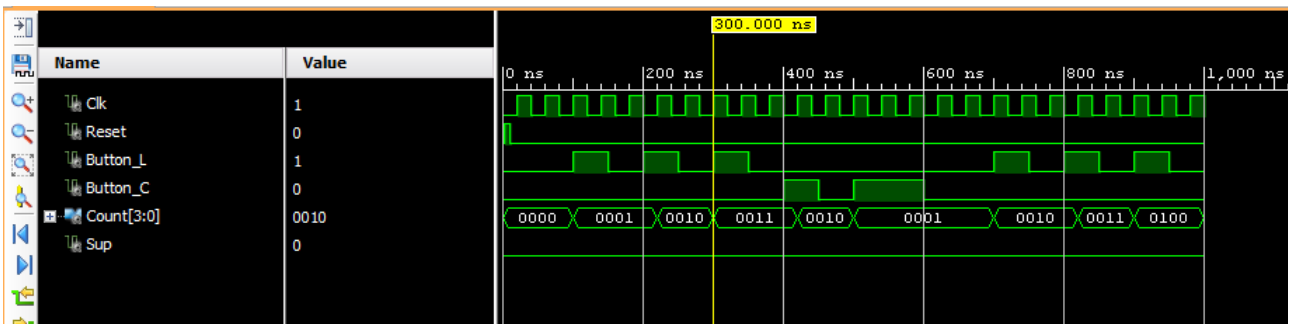


Pour contrôler les deux actions dans le même horloge on modifie le code de la gestion de compteur dans le fichier Impulse_count_.vhd en ajoutant deux signaux : signal bl: std_logic:= '0';

signal bc: std_logic:= '0';

```
-------------------------
-- Gestion du Compteur --
-------------------------
process(reset,Clk)

begin

        -- Reset Asynchrone
        --if reset='1' then cpt<="0000"; end if;

        -- Incrémentation Si on Appuie sur le Bouton Left
        if reset = '1' then
              cpt<="0000";
        elsif rising_edge(Clk) then
                --cpt<=cpt+1;
                if Button_L = '1' and bl = '0' then
                        cpt<=cpt+1;
                elsif Button_C = '1' and bc = '0' then
                        cpt<=cpt-1;
                end if;
                bl <= Button_L;
    bc <= Button_C;
        end if;

        -- Décrémentation Si on Appuie sur le Bouton Center
        --if rising_edge(Button_C) then
--      cpt<=cpt-1;
--end if;

end process;
```

Et puis on refait la simulation après synthèse et on peut bien obtenir le bon résultat:



3) Décodeur

Maintenant on ajoute une nouvelle module Selector.vhd pour générer en sortie une valeur limite qui dépend de la sortie de impulse_count. Et la valeur limite sera remit à jour à chaque appui sur le bouton right. Pour

bien fonctionner, on modifier le code en ajoutant when others => NULL ; else "00" ; dans la position nécessaire :

```
            case (Decode) is

                when "00" => Limit <= (others => '0');
                when "01" => Limit <= X"96E3600";       -- 24 000 000 en Décimal
                when "10" => Limit <= X"87A1200";       -- 8 000 000 en Décomal
                when "11" => Limit <= (others => '1');

                when others => NULL; -- RETIRER LE COMMENTAIRE POUR CORRIGER L'ERREUR

            end case;
        end if;
    end if;

end process;

--------------------------
-- Commande du Decodeur --
--------------------------
Decode <=       "11" when Sup='1'       -- Si Count > 9          --> Decode = 11
           else   "10" when Count > 5 -- Si Count = 6,7,8,9 --> Decode = 10
           else "01" when Count > 2   -- Si Count = 3,4,5    --> Decode = 01
           else "00";                    -- Si Count = 0,1,2    --> Decode = 00

end Behavioral;
```

Partie 2 : Codesign matériel/logiciel

1) Présentation

Ce TP est d'implémenter puis de programmer un système mixte matériel /logiciel sur le FPGA. Ce système sera basé sur le processeur embarqué Microblaze de Xilinx.

2) Spécification de la plate-forme matérielle

D'abord on voudrait générer la plate-forme matérielle par le module IP Integrator de Vivado.

On copie les répertoires Nexys4_DDR qui contient les propriétés des cartes de développement à l'endroit correspondant et on crée un nouveau projet. On clique Create Block Design et instancier Microblaze graphiquement dans le fenêtre Diagram. Et puis on clique sur Run Block Automation et modifie les paramètres d'IPs selon la consigne.

Après on choisit Add IP et modifie les paramètres selon le besoin dans l'onglet IP Configuration, pour ajouter des interfaces GPIO pour piloter les LED, interrupteurs, boutons de la carte. Il faut aussi renommer les connecteurs d'E /S reliés au port GPIO selon les fonctions et tracer manuellement un fil entre les ports ip2intc_irpt et les deux entrées du module Concat.

Sauvegarder le Block Design et puis valider le design en choisissant Validate Design.

3) Implémentation de la plate-forme matérielle

Maintenant on affecte les broches du FPGA aux différents connecteurs d'E/S du Block Diagram : - En ajoutant le fichier constraint (.xdc) ;
- En modifiant les lignes correspondant aux pins utilisés ;
- Et en renommant les noms de ports utilisés.

```
##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_i[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_i[1] }];
#IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_i[2] }];
#IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports { sw_tri_i[3] }];
#IO L13N T2 MRCC 14 Sch=sw[3]
```

```
## LEDs

set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { led_tri_o[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { led_tri_o[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { led_tri_o[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { led_tri_o[3] }];
#IO L8P T1 D11 14 Sch=led[3]
```

(Pas complète ici, on a modifié tous les 16 LEDs en TP.)

```
##Buttons

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }];
#IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { boutons_tri_i[1] }];
#IO_L9P_T1_DQS_14 Sch=btnc
#set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { BTNU }];
#IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { boutons_tri_i[2] }];
#IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { boutons_tri_i[0] }];
#IO_L10N_T1_D15_14 Sch=btnr
#set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { BTND }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd
```

Après on peut exécuter l'implémentation en cliquant **Create HDL Wrapper -> Generate Bitstream -> Open Implemented Design -> Export Hardware -> Launch SDK**.

4) Création des projets pour le développement logiciel Remarque sur SDK (Développement logiciel) :
- Création d'un projet logiciel pour la plateforme matérielle ; - Ecriture du code C du MicroBlaze (Drivers) ;
- Chargement Bitstream FPGA + Téléchargement du code.

On crée deux nouveaux projets : Projet logiciel + Projet BSP (Board Support Package) . Ouvre le fichier xparameter.h dans le projet BSP , on peut trouver les définitions associées aux contrôleurs d'interruptions, en plus d'un alias pour le numéro d'ID , les canaux d'interruptions associés à chaque périphérique GPIO .

Et le fichier **xgpio.h** déclare toutes les données et fonctions du driver des contrôleurs GPIO.

```
#define XPAR_BOUTONS_IP2INTC_IRPT_MASK 0X000001
#define XPAR_MICROBLAZE_0_AXI_INTC_BOUTONS_IP2INTC_IRPT_INTR 0
#define XPAR_LED_SWITCH_IP2INTC_IRPT_MASK 0X000002
#define XPAR_MICROBLAZE_0_AXI_INTC_LED_SWITCH_IP2INTC_IRPT_INTR 1
```

```
/* Definitions for driver GPIO */
#define XPAR_XGPIO_NUM_INSTANCES 2

/* Definitions for peripheral BOUTONS */
#define XPAR_BOUTONS_BASEADDR 0x40010000
#define XPAR_BOUTONS_HIGHADDR 0x4001FFFF
#define XPAR_BOUTONS_DEVICE_ID 0
#define XPAR_BOUTONS_INTERRUPT_PRESENT 1
#define XPAR_BOUTONS_IS_DUAL 0


/* Definitions for peripheral LED_SWITCH */
#define XPAR_LED_SWITCH_BASEADDR 0x40000000
#define XPAR_LED_SWITCH_HIGHADDR 0x4000FFFF
#define XPAR_LED_SWITCH_DEVICE_ID 1
#define XPAR_LED_SWITCH_INTERRUPT_PRESENT 1
#define XPAR_LED_SWITCH_IS_DUAL 1
```

5) Développement de l'application logicielle

Dans cette partie on dédie à écrire le code dans le projet logiciel pour piloter les LED en fonction de l'état des boutons et des interrupteurs de la carte en utilisant les fonctions du driver de GPIO.

Après avoir étudié les fonctions du driver de GPIO, on a écrit un premier programme gpio.c (Veuillez regarder le code ci-joint) qui permet d'allumer une des LED si on commute un interrupteur vers le haut et de l'éteindre si on tire l'interrupteur vers le bas.

Et puis on programme le FPGA pour qu'il contienne le système Microblaze sur lequel sera exécuté le code. On télécharge le code logiciel dans la mémoire du Microblaze et on fait le Debug.

Après avoir testé le programme et avoir corrigé le code, on peut bien exécuter le programme et réaliser les actions qu'on veut. Mais on a observé que l'allumage et l'éteint de LED a une réaction lente, c'est-à-dire la sensibilité n'est pas assez idéale. (parce qu'il est ergodique comparé à l'utilisation du contrôleur d'interruptions)

6) Utilisation du contrôleur d'interruptions

Pour augmenter la sensibilité du programme, on veut maintenant utiliser les boutons poussoirs en mode interruption.

- Il faut configurer la sortie interruption pour le contrôleur GPIO des boutons dans Vivado.

- Et on doit modifier le programme précédent pour que le compteur modulo 16 soit géré par une interruption.

Et puis on a écrit un nouveau programme interrupt.c (Veuillez regarder le code ci-joint) en utilisant le contrôleur d'interruptions. Cette fois-ci les LEDs ont une réaction vite et on a bien augmenté la sensibilité du programme.

Partie3 : Conception d'IP pour le Microblaze

Le but de ce TP est de réaliser une IP destinée à être connectée au Microblaze , en remplacement du contrôleur de LED. Le développement sera réalisé grâce aux outils Xilinx : Vivado pour le développement de l'IP et la génération de la plate-forme matérielle ; SDK pour le développement et l'exécution de l'application logicielle.

1) Création d'une IP Contrôleur de LED

Dans Vivado on clique sur Create and Package IP pour créer un nouveau IP, et puis on crée aussi un

nouveau fichier my_led.vhd qui permet d'implémenter la fonctionnalité de l'IP dans le menu File.

A part le fichier my_led.vhd, on ajoute aussi deux autres fichiers aux sources du projet :
- my_led_v1_0.vhd : Le fichier top de l'IP ;
- my_led_v1_0_S00_AXI.vhd : Le fichier qui implémente le protocole AXI pour relier au bus.

Dans le fichier my_led_v1_0.vhd : 1) on ajoute le port de LEDs :

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

entity led is
port(   sw_state : in STD_LOGIC_VECTOR ( 3 downto 0 );
        leds : out STD_LOGIC_VECTOR ( 15 downto 0 )
);
end led;

architecture fonction of led is
begin
    leds(3 downto 0) <= "1111" when sw_state(0) = '1' else
                        "0000";
    leds(7 downto 4) <= "1111" when sw_state(1) = '1' else
                        "0000";
    leds(11 downto 8) <= "1111" when sw_state(2) = '1' else
                        "0000";
    leds(15 downto 12) <= "1111" when sw_state(3) = '1' else
                        "0000";
end fonction;
```

```
port (
        -- Users to add ports here
leds : out STD_LOGIC_VECTOR ( 15 downto 0 );
        -- User ports ends
        -- Do not modify the ports beyond this line
```

2) On déclare le component correspondant :

```
-- component declaration
component myled2_v1_0_S00_AXI is
        generic (
        C_S_AXI_DATA_WIDTH      : integer       := 32;
        C_S_AXI_ADDR_WIDTH      : integer       := 4
        );
        port (
        leds : out STD_LOGIC_VECTOR ( 15 downto 0 );
        S_AXI_ACLK      : in std_logic;
```

3) On fait l'instanciation de LED :

```
-- Instantiation of Axi Bus Interface S00_AXI
myled2_v1_0_S00_AXI_inst : myled2_v1_0_S00_AXI
        generic map (
                C_S_AXI_DATA_WIDTH      => C_S00_AXI_DATA_WIDTH,
                C_S_AXI_ADDR_WIDTH      => C_S00_AXI_ADDR_WIDTH
        )
        port map (
          leds => leds,
```

Dans le fichier my_led_v1_0_S00_AXI.vhd :

- on ajoute le port de LEDs dans l'entité:

- On ajoute un signal :

```
port (
        -- Users to add ports here
leds : out STD_LOGIC_VECTOR ( 15 downto 0 );
        -- User ports ends
        -- Do not modify the ports beyond this line
```

```
    ------------------------------------------------
    ---- Signals for user logic register space example
    ------------------------------------------------
    ---- Number of Slave Registers 4
    signal slv_reg0 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    signal slv_reg1 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    signal slv_reg2 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    signal slv_reg3 :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    signal slv_reg_rden     : std_logic;
    signal slv_reg_wren     : std_logic;
    signal reg_data_out     :std_logic_vector(C_S_AXI_DATA_WIDTH-1 downto 0);
    signal byte_index       : integer;

signal sw_state: std_logic_vector(3 downto 0);
```

3) On ajoute aussi les lignes suivantes le dans l'architecture pour fixer le comportement consigné pour l'IP :

Après on fait le **Package IP -> Merge Changes** from IP File Groups Wizard.

2) Intégration de l'IP au système Microblaze

De retour das le projet Vivado, on ouvre le Block Design et fait les travaux suivants :

- On retire l'IP GPIO led_switch ;

- On modifie le nom de l'IP GPIO boutons en sw ;

- On ajoute un nouveau IP my_led ;

- On supprime le bloc Concat et connecte manuellement la ligne d'interruption du bloc sw à l'entrée intr du contrôleur d'interruptions ;

- On mettre en commentaireles assignations de broches correspondant aux boutons ;
Remarque : Il faut faire attention à la correspondance du nom de port et lequel dans le ficher .xdc.

Après avoir fini les travaux et avoir fait les modifications correspondantes, on fait **Validate Design -> Generate Bitstream -> Launch SDK**.

3) Développement logiciel

Et puis on peut remarquer qu'il génère un nouveau projet matériel dans SDK qui s'appelle TP2_wrapper_hw_platform1. C'est le projet correspond à l'IP my_led. Garde le et crée un nouveau projet logiciel TP3 avec un projet BSP associé.

Ouvre le fichier my_led.h au répertoire microblaze_0/include. C'est un fichier contient une série de définitions qui permettent d'utiliser des fonctions de lecture ou d'écriture des registres de l'IP.

Ensuite on crée un programme C my_ip.c (Veuillez regarder le code ci-joint) dans le projet TP3 qui lit l'état des 4 interrupteurs et qui allume les LED par blocs de 4 si les interrupteurs sont actifs.

```
        -- Add user logic here

    10: entity work.led port map(sw_state,leds);
    sw_state(0) <= slv_reg0(0);
    sw_state(1) <= slv_reg0(1);
    sw_state(2) <= slv_reg1(0);
    sw_state(3) <= slv_reg1(1);

        -- User logic ends
```