I) Introduction

Nous allons partir d'une version corrigée de l'ATTiny861 avec rs232 et en faire un certain nombre de modifications. Dans cette épreuve , on ne vous demande pas de vous poser de questions sur l'utilité ou non des modifications mais seulement de les exécuter et de montrer que le travail résultant fonctionne toujours. Le but de l'épreuve est ainsi d'évaluer comment vous appréhendez des modifications à partir d'un schéma complexe décrit en VHDL.
Vous disposez d'une feuille réponse sur laquelle vous répondez et sur laquelle l'enseignant notera ce qu'il a vu fonctionner et sinon, quelques indications sur l'état d'avancement de votre travail.

Ce qui vous intéresse est dans "Resources2015.zip"
(dans le répertoire Final2016).

II) ATTiny861, compteur BCD et RS232

La partie matérielle vous est complètement donnée dans le répertoire Final2016. Elle correspond à la correction du TP9 (figure ci-dessous). En voici un schéma fonctionnel.

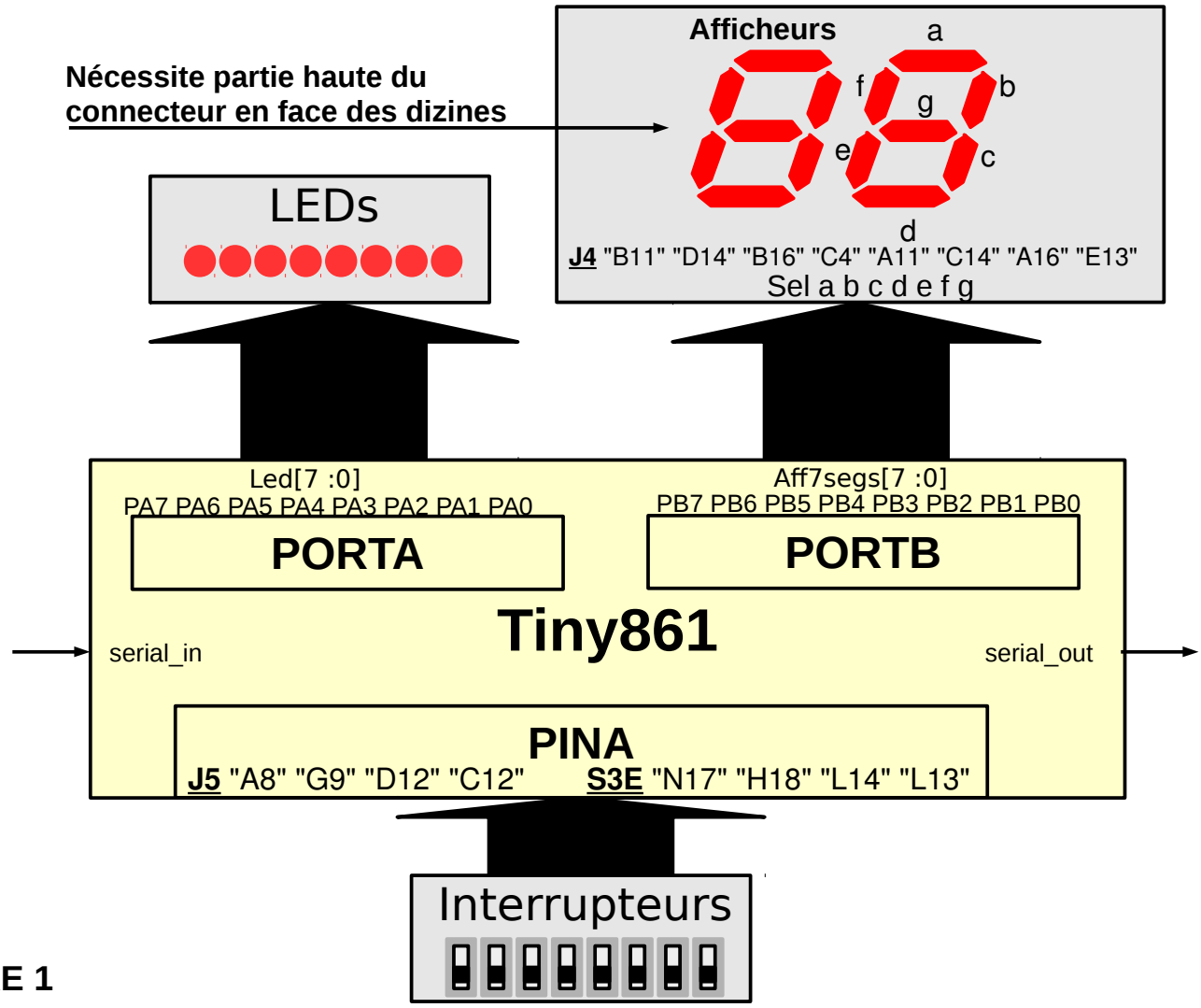

FIGURE 1

Travail à réaliser (exo1)

2-1) On donne un programme qui réalise un compteur de passages sur deux digits complètement en langage C. Le faire fonctionner. Comment modifier le programme pour qu'il affiche correctement ?

Ce travail consiste donc à :
- faire un projet (Xilinx IDE)
- décompacter la ressource dans le répertoire du projet Xilinx !!!!!!
- ajouter tous les fichiers (sauf cmpPassageUTT.vhd) et compiler le projet (Xilinx IDE)
- compiler le programme C « CmptPassage.c » (script dans « soft » en modifiant peut-être le nom du programme à compiler)
- télécharger l'ensemble dans le FPGA (Impact)

Travail à réaliser (exo2)

On vous demande de déporter la partie logicielle de transcodage dans le matériel.

2-2) Réaliser cette modification matérielle conformément au schéma partiel ci-dessous :

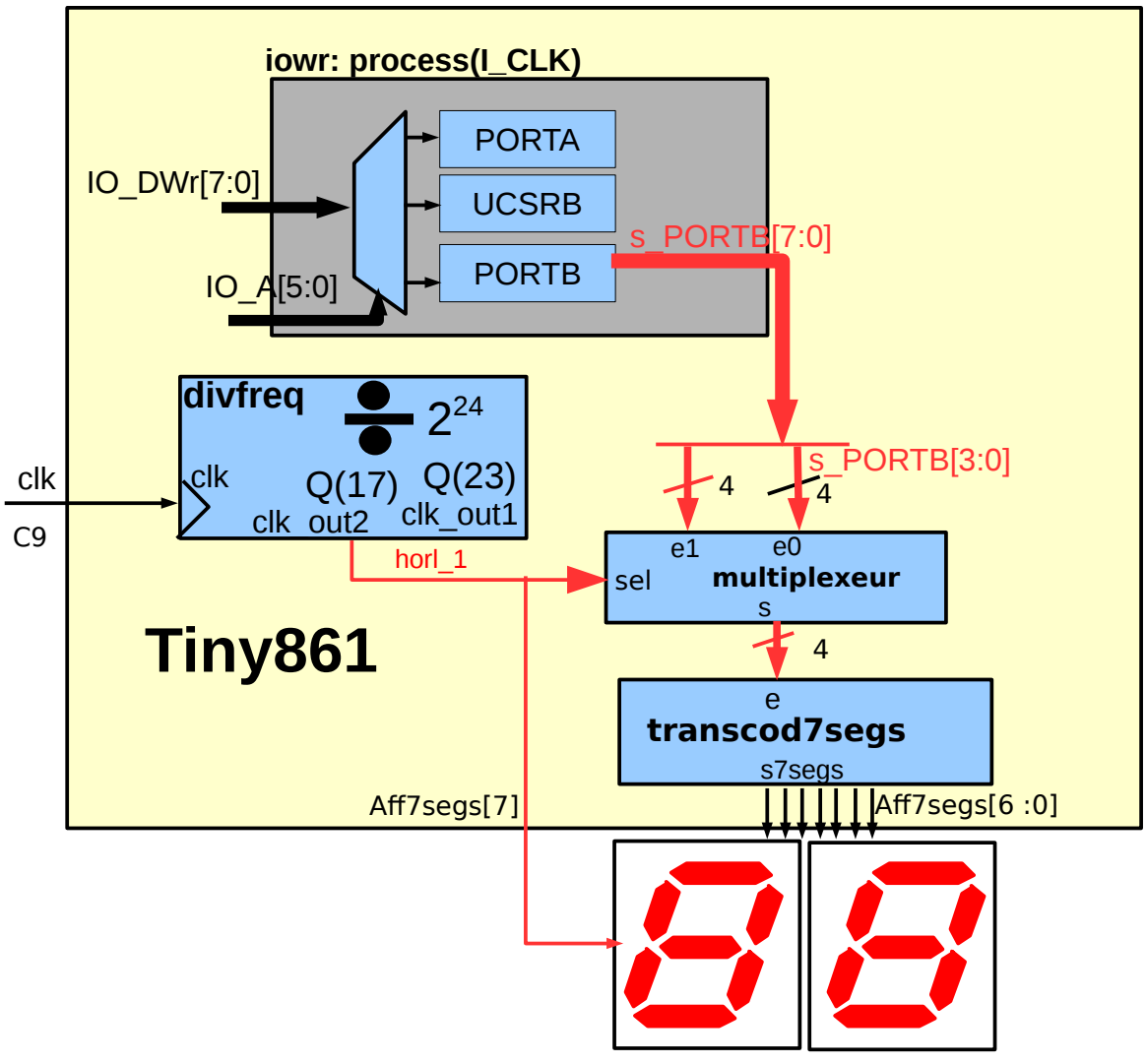

FIGURE 2

Les rectangles bleus de la figure peuvent être trouvé dans le fichier « cmpPassageUTT.vhd » **PORTB** ne sort plus vers l'extérieur mais est relié au multiplexeur. L'écriture de 0x97 dans le PORTB affichera « 97 »

Ce qui est à faire est en rouge. (Tout ceci a été fait en TP dirigé)

2-3) Modifier le programme correspondant pour que l'ensemble continue à fonctionner comme un compteur de passages. Vous devrez retirer complètement le transcodage en C et la gestion des deux afficheurs.

Envoi du compteur de passages dans la liaison série

2-4) On vous demande d'ajouter dans le programme précédent, un envoi par la liaison série du compteur cmpt mais uniquement quand ce dernier a changé.

La difficulté de cet exercice est de bien comprendre que :

- la liaison série fonctionne en mode caractères
- le compteur BCD fonctionne en chiffres
- le chiffre 0 est différent du caractère '0'

On désire maintenant déporter cet affichage sur un écran LCD. Là encore nous avons affaire à un périphérique en mode caractères.

III) Interfaçage direct de l'écran LCD au Tiny861

Pour une fois on ne vous donne pas de schéma. Vous allez partir du fichier lcd.c qui vous est donné. Lisez-le. Comme vous pouvez le remarquer, la gestion de l'affichage est complètement réalisée par le **PORTB**. Nous ne pouvons pas garder cette gestion car ce port nous sert à commander les deux afficheurs sept segments et nous tenons à la garder ainsi.
On vous demande donc de choisir une adresse non utilisée de registre parmi les constantes VHDL ci-dessous (dans microcontroleurTP9.vhd) :

```
--Registres et PORTs de l'ATTiny861
constant OCR1A : std_logic_vector(5 downto 0) := "101101";
constant OCR1B : std_logic_vector(5 downto 0) := "101100";
constant PORTA : std_logic_vector(5 downto 0) := "011011";
constant DDRA : std_logic_vector(5 downto 0) := "011010";
constant PINA : std_logic_vector(5 downto 0) := "011001";
constant PORTB : std_logic_vector(5 downto 0) := "011000";
constant DDRB : std_logic_vector(5 downto 0) := "010111";
constant PINB : std_logic_vector(5 downto 0) := "010110";
constant ADCH : std_logic_vector(5 downto 0) := "000101";
constant ADCL : std_logic_vector(5 downto 0) := "000100";
--Registres non présents dans l'ATTiny861
constant UDR : std_logic_vector(5 downto 0) := "000011";
constant UCSRA : std_logic_vector(5 downto 0) := "000010";
constant UCSRB : std_logic_vector(5 downto 0) := "000001";
```

et de définir une constante associée que l'on appellera LCD dans le code VHDL.

Travail à réaliser (exo3)

3-1) réalisation de la partie matérielle et les essais correspondants
- Ajouter une sortie correspondante avec le nom que vous voulez et gérez-là correctement dans le process

d'écriture
- Ajouter une entrée correspondante capable de lire à tout moment ce qui a été sorti (demandez aide si besoin)
- adaptez les contraintes ucf à votre sortie :

```
#lcd : registre LCD
NET " ????<0>" LOC = "B4" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<1>" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<2>" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<3>" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<4>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<5>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<6>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
NET " ????<7>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW = SLOW ;
```

- Vous êtes prêt à compiler votre partie matérielle

- Modifiez le programme lcd.c en remplaçant tous les **PORTB** par des **LCD**, pour qu'il fonctionne avec votre registre **LCD**. Il serait donc bien que ce nom apparaisse dans votre programme c.

3-2) Adaptez votre programme du compteur de passages pour qu'il fonctionne maintenant complètement avec l'écran LCD (en gardant éventuellement l'affichage sur les afficheurs 7 segments et par la liaison série).

IV) Déporter la gestion du LCD dans le matériel

On désire déporter la gestion du LCD dans le matériel. Pour cela on vous donne dans le répertoire « Final2016/LCD » de la ressource une partie matérielle que l'on a trouvé sur Internet (chez Opencores.org). Elle a été adaptée pour notre carte FPGA : seul le fichier ucf était à changer.

Travail à réaliser (exo4)

4-1) Réaliser un projet avec les trois fichiers du répertoire et montrer le bon fonctionnement de l'ensemble. Il n'y a pas de processeur ici, c'est juste un test.

4-2) Pouvez-vous imaginer et réaliser une interface (même partielle) entre le fichier « lcd16x2_ctrl.vhd » et notre Tiny861 pour au moins gérer un affichage des deux chiffres du compteur de passages. Évidemment on laissera tomber le fichier « lcd16x2_ctrl_demo.vhd » qui n'est là que pour montrer comment l'ensemble fonctionne.

V) On déporte maintenant les compteurs BCD

Il est possible de déporter un peu plus de logiciel dans le matériel. Après le transcodage, nous allons déplacer aussi les deux compteurs BCD dans le matériel.

Voici en schématique ce qui va être réalisé :

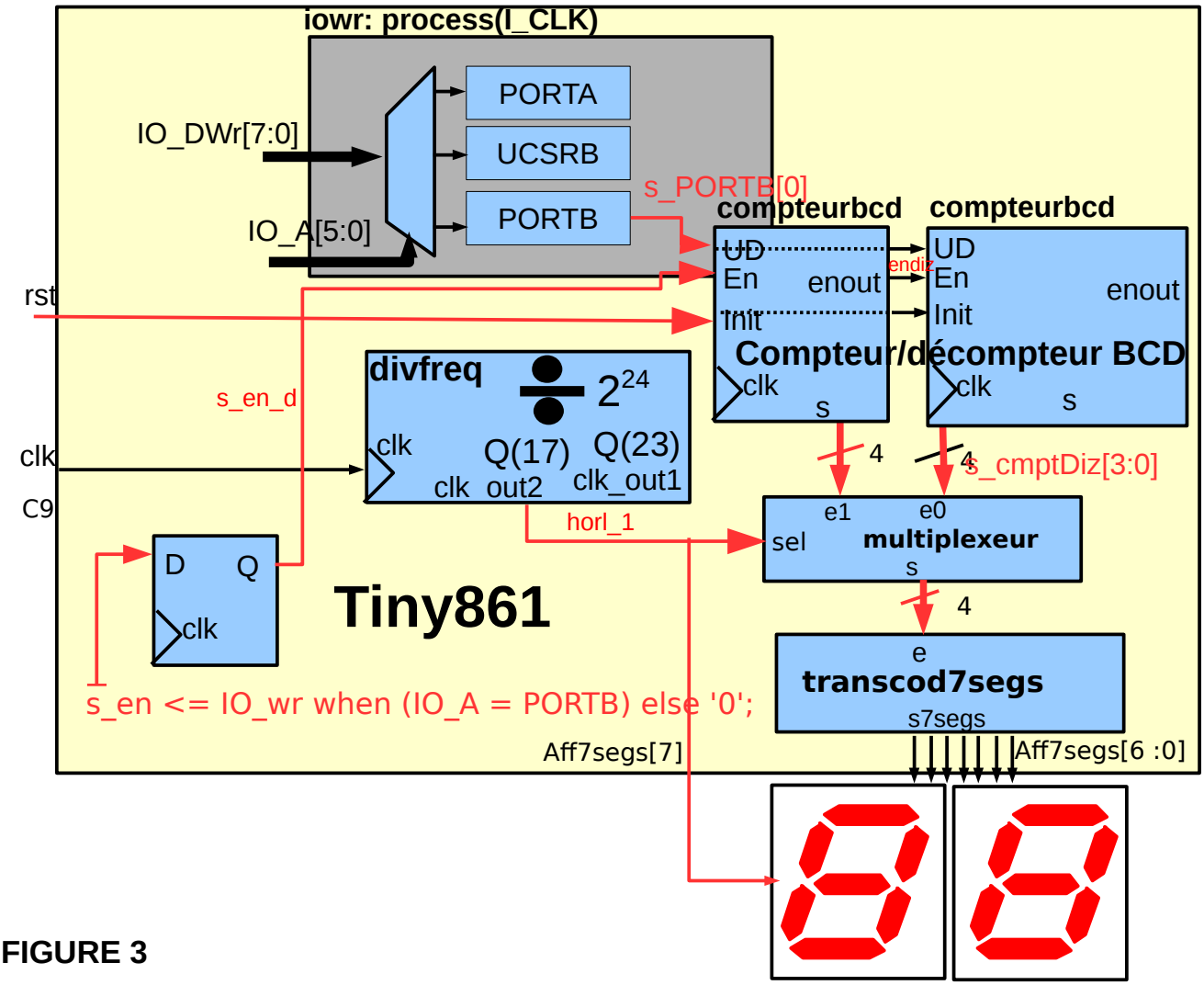

FIGURE 3

La présence de la bascule D n'est pas forcément très facile à appréhender. Elle est liée au fait que le bit de poids faible de PORTB sert à changer l'entrée UD (choix comptage ou décomptage) des deux compteurs. Si en même temps on valide avec 'en' et bien on aura toujours un temps de retard. En retardant, on positionne d'abord 'UD' **puis** on valide le comptage ou le décomptage.

Travail à réaliser (exo4)

5-1) Pouvez-vous prévoir à l'aide du code du compteur BCD ce que fera l'opération :
PORTB = 1 ;

en C ? Réponse sur la feuille réponse.

5-2) Réaliser la partie matérielle.

Le compteur BCD est disponible dans les ressources dans le répertoire Final2016 dans le fichier « cmptPassageUTT.vhd »

5-3) Adaptez le compteur de passages en C à votre nouveau matériel.

5-4) Sauf si vous gardez l'ancien calcul, votre processeur ne sait plus à combien sont les compteurs. Modifiez l'architecture du **Tiny861** pour donner la possibilité au processeur de lire la valeur des deux compteurs BCD. Envoyez ces valeurs sur l'écran LCD et comparez.