Réalisation partielle d'un réveil avec un Tiny861

Règles de déroulement de l'épreuve

Il n'est pas interdit de communiquer verbalement entre binômes tant que cela ne perturbe pas le déroulement correct de cette séance.
Il vous est absolument interdit, par contre, de vous déplacer (sauf urgence), d'échanger des feuilles de brouillons, des notes ainsi que des fichiers et des clés USB.
Les téléphones portables sont interdits pendant toute la durée de l'épreuve.

Introduction

Nous allons partir d'une version corrigée de l'ATTiny861 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 Final2017).

I) Interfaçage direct de l'écran LCD et du codeur au Tiny861

La partie matérielle de départ vous est complètement donnée dans le répertoire Final2016. Elle correspond à la correction du TP7 (figure ci-dessous) où l'on va faire des modifications au fur et à mesure du déroulement de l'épreuve. Nous allons tout d'abord remplacer l'affichage sur deux digits 7 segments par un écran LCD. Ce travail est montré en figure 1 par la superposition d'un écran LCD sur les afficheurs 7 segments.

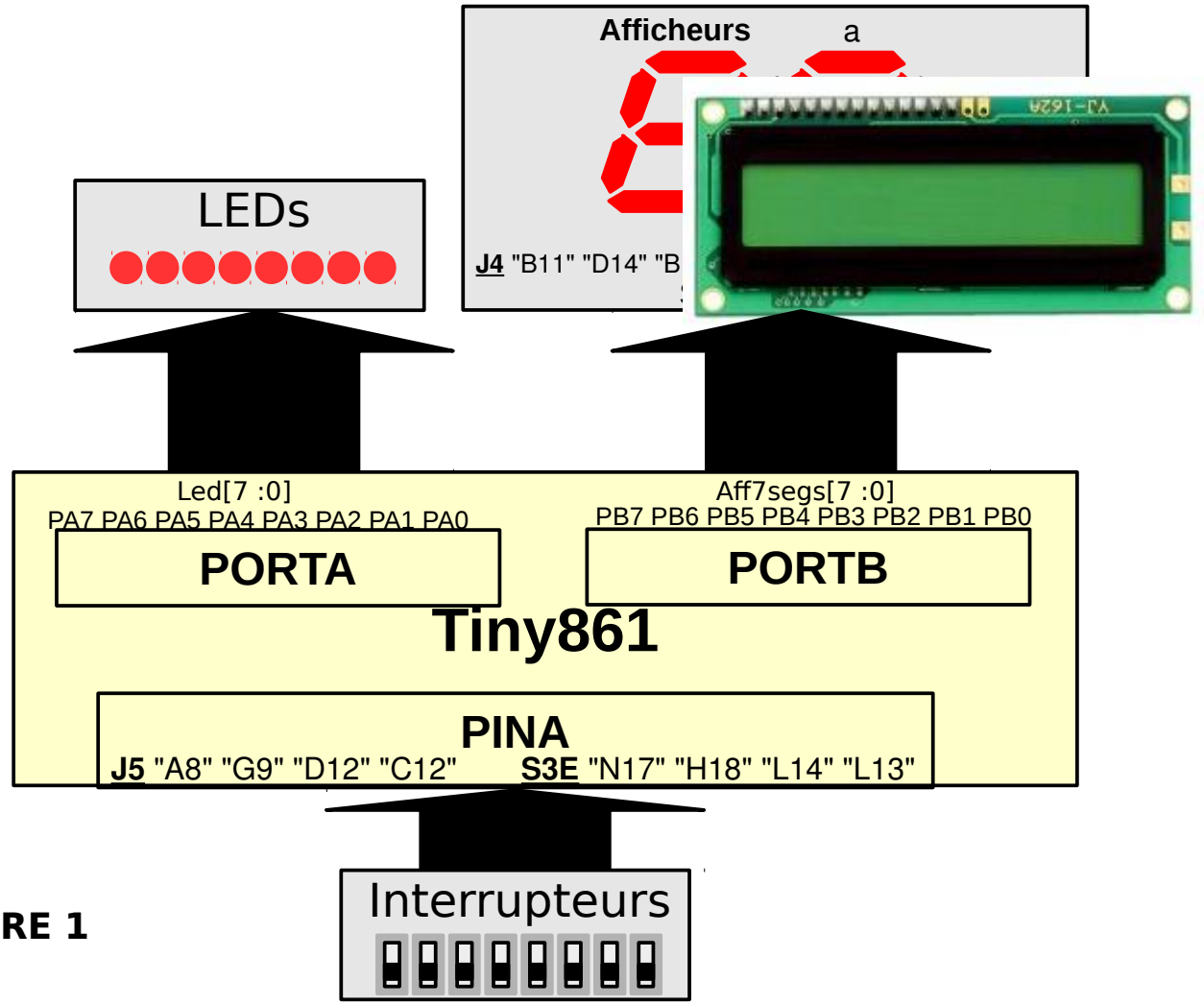

FIGURE 1

Travail à réaliser (exo1)

1-1-a) Modifier le processeur du TP 8 (de la figure ci-dessus = microcontroleur.vhd) pour interfacer directement un afficheur LCD en respectant le tableau ci-dessous : (NC=Not Connected)

PORTB	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	D3	D2	D1	D0	RW	RS	E	NC

INFO

Ce travail consiste donc à :
- modifier le **PORTB** pour qu'il commande les signaux de l'afficheur LCD au lieu des deux afficheurs sept segments
- la présence des PORTB |= PORTB ... dans le code lcd.c nécessitent la gestion du PORTB à la fois en écriture et en lecture. Profitez de ces modifications pour changer le nom des sorties « Aff7segs »
- ajouter tous les fichiers nécessaires et compiler le projet (Xilinx IDE)
- compiler le programme C « lcd.c » (script dans « soft » en modifiant

peut-être le nom du programme à compiler)

- télécharger l'ensemble dans le FPGA (Impact)

1-1-b) Réaliser un compteur en C ainsi que son affichage correct sur l'écran LCD.

Travail à réaliser (exo2)

On vous demande d'utiliser **PINB** (déjà présent) pour interfacer directement le codeur incrémental et les boutons associés (ceux qui sont disponibles).

1-2-a) Réaliser cette modification matérielle conformément aux contraintes ci-dessous :

PINB	b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
	NC	BTN_W	BTN_S NC	BTN_N	BTN_E	rot_center	rot_B	rot_A

1-2-b) Réaliser un programme capable de gérer le décodeur incrémental avec le compteur de l'exercice 1-1-b. Ce travail nécessite la détection d'un front descendant sur rot_A.

Nous rappelons que lcd.c contient du code important pour cet exercice.

La détection de front se fait avec un code du genre :

```
rot_A_p = PINB & 0x01;
if ((rot_A_p) == 0 && (rot_A == 1)) // front descendant
   if (PINB & 0x02) // rot_B == 1
        tmp++;
   else
        tmp--;
setCursor(5,0);
if ((tmp&0xF0)<0xA0)
        writecar(((tmp&0xF0)>>4)+'0');
else
        writecar((((tmp&0xF0)-0xA0)>>4)+'A');
if ((tmp&0x0F)<0x0A)
        writecar((tmp&0x0F)+'0');
else
        writecar(((tmp&0x0F)-0x0A)+'A');
delai(10000);
rot_A = rot_A_p;
```

II) 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 « Final2017/LCD » de la ressource la partie matérielle de chez Opencores.org, celle que l'on a utilisée aussi pour le médian.

Travail à réaliser (exo3)

2-1) On vous demande de réaliser l'interface entre « lcd16x2_ctrl.vhd » et notre Tiny861 pour au moins gérer un affichage de l'heure courante sur la première ligne comme pour le médian :

Ce travail consiste donc à :
- interfacer « lcd16x2_ctrl.vhd » à l'heure courante : ceci était réalisé dans « lcd16x2_ctrl_demo.vhd » pour le médian mais sera fait dans « microcontroleur.vhd » maintenant.
- ajouter **ReveilHHMMUp.vhd** comme dans le schéma et câbler
- compléter la première ligne par des espaces
- câbler la deuxième ligne avec un texte de votre choix(1)
- entrer des données au choix pour la deuxième ligne
- ajouter tous les fichiers nécessaires et compiler le projet (Xilinx IDE)
- l'affichage de l'heure courante fonctionne quelque soit le programme C
- télécharger l'ensemble dans le FPGA (Impact)

(1) Il est possible de câbler rapidement du texte fixe sur une ligne en faisant directement dans un port map :

```
line2_buffer => X"2020202048656C6C6F20202020202020",
```

qui affiche « Hello ». Pourquoi ?

La partie jaune du dessin n'est pas à réaliser. Elle a été réalisée pendant le médian et est donnée complètement corrigée dans une entité appelée « heure_courante » dans le fichier **ReveilHHMMUp.vhd**.

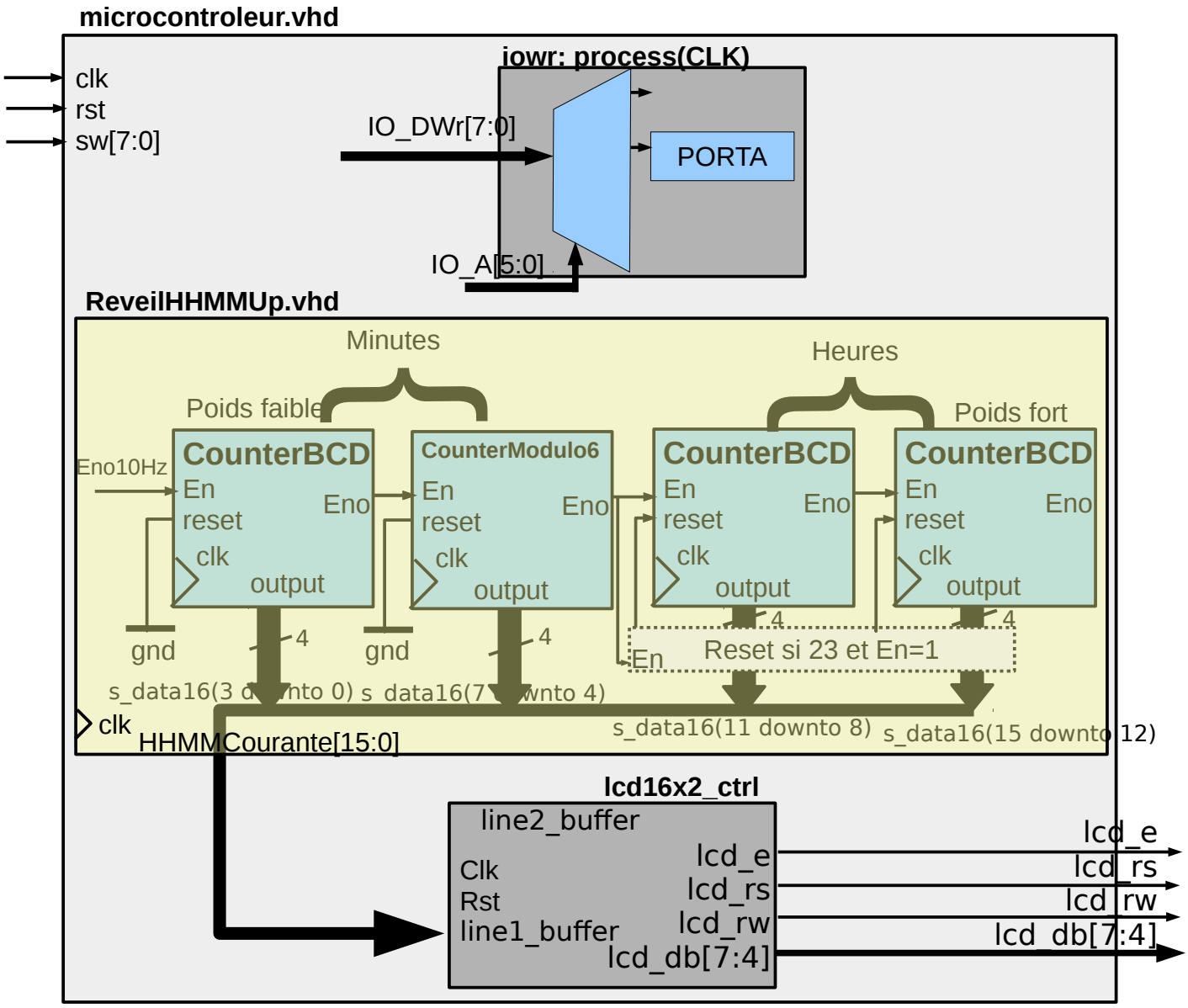

FIGURE 2

III) Gestion d'une deuxième ligne complète

La première ligne de l'afficheur LCD est laissée à l'heure courante. Il s'agit d'un affichage spécialisé, capable de n'afficher que HH:MM. Nous désirons réaliser maintenant un affichage complètement généraliste sur la deuxième ligne.

Il nous faut donc gérer 16 caractères (de 8 bits). Pour se faire, il est facile de réaliser une solution avec 16 PORTs mais cette solution consomme beaucoup trop de registres. Si nous avons besoin d'autres périphériques, ce serait difficile...

Nous allons donc utiliser la technique du FIFO. Il est alors possible de n'utiliser qu'un seul PORT pour écrire les 16 caractères à afficher. Comme nous allons le voir, réaliser une architecture qui écrit dans un FIFO est très simple... c'est le vider automatiquement qui nécessite un peu d'attention.

Travail à réaliser (exo4)

3-1) Nous allons aussi utiliser un seul PORT qui écrira dans le FIFO (le **PORTB** en l'occurrence). C'est donc le signal FIFO plein qui fera partir le séquenceur.

Voici donc comment on interface un FIFO de manière fonctionnelle, même si son contenu n'est utilisé que de manière peu commune à ce point.

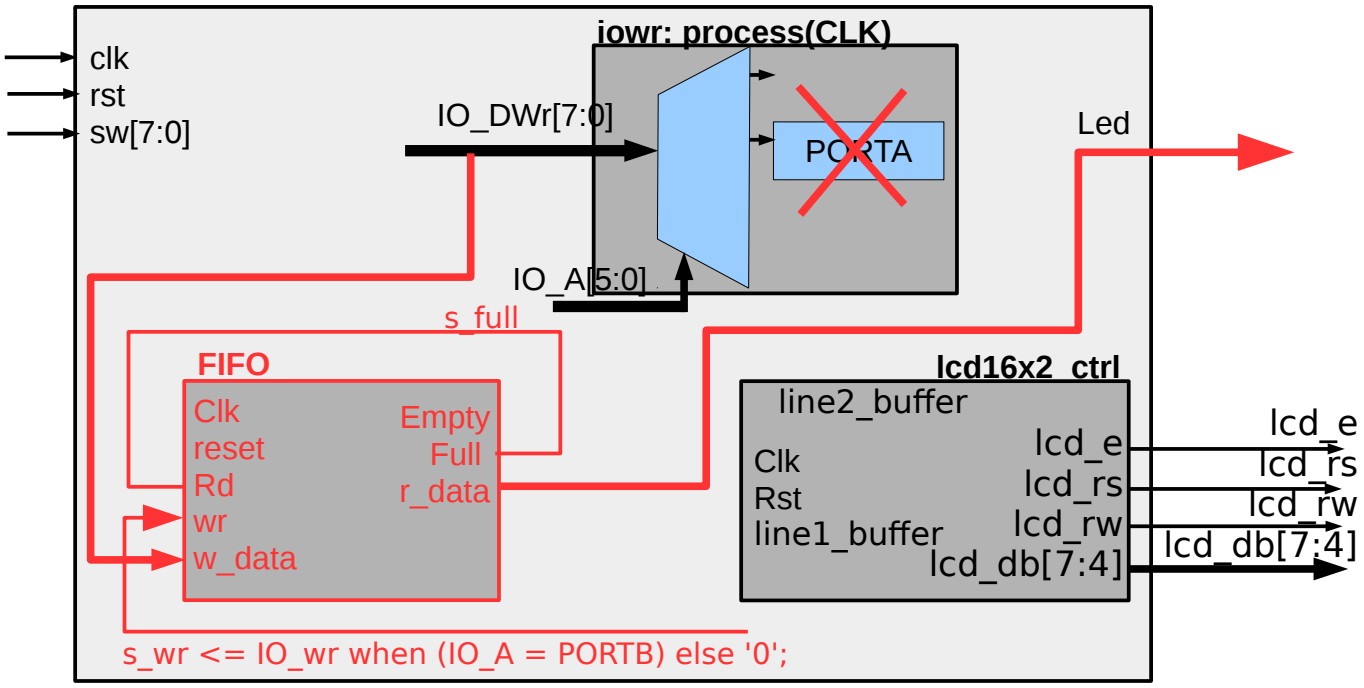

FIGURE 3

Le fil « s_full » qui relie « Full » à « Rd » est là pour vider au fur et à mesure le FIFO une fois qu'il est plein. Autrement seule la première valeur entrée est disponible dans « r_data » et donc dans « led ».

Réaliser la partie matérielle de la figure 3 puis écrire un programme qui réalise un chenillard avec celle-ci pour montrer que vous avez bien compris la partie matérielle.

A partir de maintenant, la connexion aux Leds se fera de nouveau avec le **PORTA**. Cela évitera des erreurs dans le fichier ucf. Par contre « r_data » servira à autre chose (voir figure 4)

Vient la partie la plus difficile : le séquenceur responsable du vidage du FIFO.

Ce séquenceur est réalisé en trois parties comme suggéré dans la figure ci-dessous :

- le séquenceur proprement dit reconnaissable à son schéma avec états

- un compteur responsable de la détection de la fin du vidage du FIFO : s_termine passe à '1' lorsque le compteur est à 15

- un multiplexeur (séquentiel responsable de l'aiguillage des données dans le buffer de ligne.

INFO

Ces trois composants seront réalisés par des process pour éviter les longs câblages.

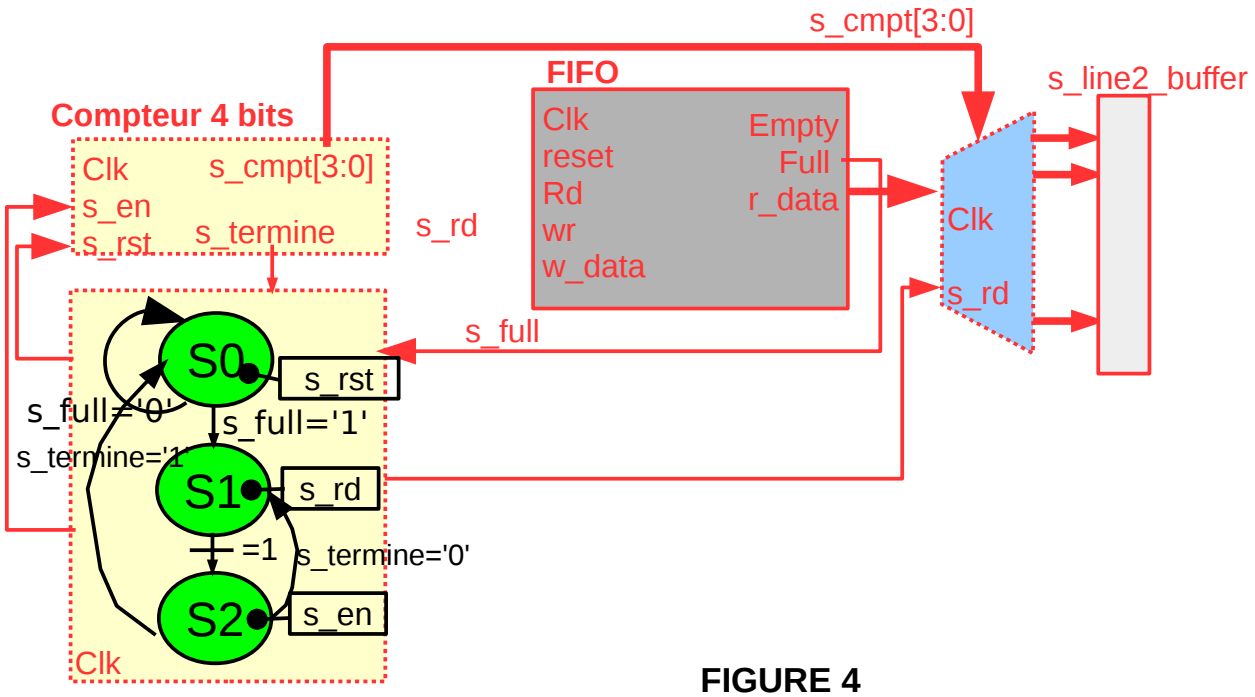

FIGURE 4

3-2-a) Ajouter la partie matérielle ci-dessus dans le microcontrôleur.

INFO

Le multiplexeur est réalisé à l'aide d'un code du genre :

```
-- multiplexeur séquentiel pour les données
buildingLine2: process(clk) begin
   if rising_edge(clk) then
     if s_rd = '1' then
        case s_cmpt is
           when "0000" => s_line2_buffer(127 downto 120) <= s_r_data;
           when "0001" => s_line2_buffer(119 downto 112) <= s_r_data;
```

3-2-b) Réaliser un programme qui affiche « bonjour à tous » sur la deuxième ligne.

Si votre programme fonctionne correctement, vous n'aurez plus à changer la partie matérielle à partir de maintenant.

3-2-c) Réaliser la mise à jour de l'heure de réveil avec le bouton codeur incrémental en programmation.

Pensez à regarder dans RessourcesReveil.c pour les incrémentations et décrémentations des heures minutes.

Utilisez un code comme celui de la question 2-2-b). Nos essais ont montré qu'un delai(1000) ou _delay_ms(2) sont suffisants dans la boucle while(1)

3-2-d) Améliorer l'ensemble comme vous le désirez. Les pistes suggérées sont :

- gestion de l'armement du réveil

- gestion de l'automate de sonnerie

- gestion de la mise à l'heure en séparant les heures et les minutes à l'aide du bouton d'appui du codeur incrémental. Cela était difficile dans le médian mais se résume à de la programmation en C ici.