Faculté de physique et ingénierie
Université de Strasbourg

RAPPORT DE PROJET

Systèmes Numériques Embarqués

Mini-Projet

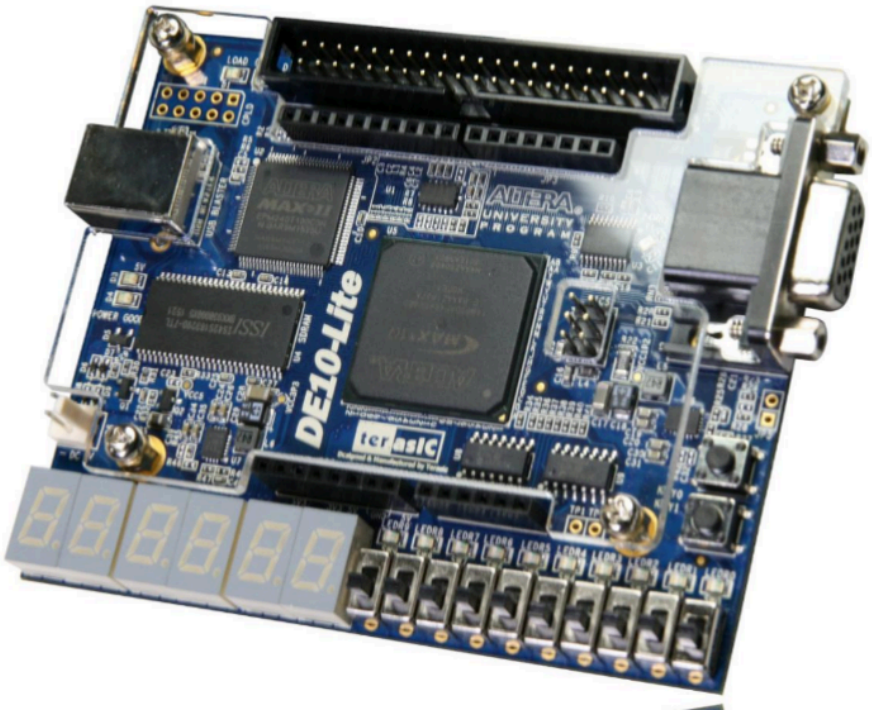


Figure 1 : Kit de développement DE10-Lite

Conception et implantation d'un réveil électronique sur cible FPGA

Documents joints : Manuel d'utilisation

Encadrant : M. Foudil DADOUCHE

Tables des matières

I. Introduction

1. Contexte du projet

Ce mini-projet s'inscrit dans le cadre de ma formation en Master 1- PAIP MESI à la faculté de Physique & Ingénierie de l'université de Strasbourg. Il a pour but la conception conjointe donc le Co-design (matérielle et logicielle) de systèmes numériques embarqués sur cible FPGA (System on a Programmable Chip - SoPC).

2. Objectif

L'objectif principal est de concevoir et d'implémenter un réveil électronique complet sur une cible FPGA DE10-Lite. Le système repose sur un processeur Nios II, programmé en langage C, interfacé avec des périphériques matériels (afficheurs 7 segments, interrupteurs, boutons poussoirs, LEDs, haut-parleur) via des PIOs définis dans Platform Designer (Qsys).

3. Outils utilisés

Le flot de conception s'appuie sur la chaîne d'outils Intel/Altera :

- Quartus Prime : synthèse matérielle, placement-routage et configuration du FPGA.
- Platform Designer (Qsys) : intégration du système matériel (processeur, mémoire, PIOs, timers).
- Nios II SBT for Eclipse : écriture, compilation et déploiement du code C sur le processeur Nios II.
- Kit DE10-Lite : carte de développement équipée d'un FPGA MAX 10 (10M50DAF484C7G).

4. Remerciements

Tout d'abord, je tiens à exprimer ma reconnaissance ainsi que ma profonde gratitude à Monsieur **Foudil DADOUCHE** pour la qualité de son encadrement tout au long de ce projet. Ses conseils avisés et son expertise technique dans le domaine des systèmes numériques embarqués m'ont permis d'appréhender avec succès les enjeux du co-design matériel/logiciel.

II. Analyse du cahier des charges

1. Fonctionnalités de base

a. Affichage

En mode normal, l'heure actuelle doit être affichée sur 6 afficheurs 7 segments (format heures, minutes, secondes) en mode 24 heures.

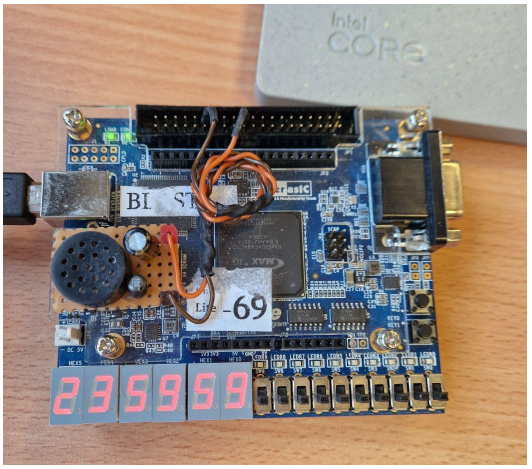


Figure 2 : Affichage du format 24h sur carte FPGA

b. Alarme

Le système génère un signal sonore périodique carré de 500 Hz via le haut-parleur lorsque l'heure actuelle coïncide avec l'heure d'alarme configurée. L'activation et la désactivation de l'alarme sont contrôlées par l'interrupteur SW2.

c. Indicateurs visuels

Une LED témoin (LEDR0) indique visuellement l'état de l'alarme, allumée lorsque SW2 est en position ON, éteinte sinon.

d. Comportement matériel

Les boutons poussoirs KEY0 et KEY1, utilisés respectivement pour ajuster les minutes et les heures, génèrent nativement un niveau logique 1 au repos et 0 à l'appui. Cependant, un inverseur a été placé dans le schéma BDF entre les boutons physiques et les PIOs, ce qui inverse cette logique donc le PIO reçoit 1 quand le bouton est pressé et 0 quand il est relâché.

En conséquence, le PIO est configuré en détection de front montant (Rising Edge) dans Qsys. Les interrupteurs (SW) génèrent un 1 en position ON et un 0 en position OFF. À la mise sous tension, l'heure actuelle et l'heure d'alarme sont initialisées à zéro.

2. Améliorations apportées

Au-delà du cahier des charges de base, j'ai implémenté les améliorations suivantes :

Format d'affichage 12h/24h : j'ai ajouté la possibilité de basculer entre un affichage au format 24 heures et 12 heures, contrôlé par la combinaison des interrupteurs SW1 et SW0.

Signal sonore personnalisé (mélodies) : j'ai remplacé le simple bip de 500 Hz par la génération de véritables mélodies. La sélection de la mélodie souhaitée se fait via les interrupteurs SW7, SW8 et SW9, permettant jusqu'à 8 mélodies différentes.

Modes de réglage : j'ai intégré un choix de mode de réglage via l'interrupteur SW4 : soit par appuis successifs (chaque pression incrémente d'une unité), soit en mode continu par appui maintenu avec un pas d'incrémentation de 0,5 seconde.

Verrouillage de l'affichage : j'ai ajouté un mode de sécurité via l'interrupteur SW3, permettant de basculer entre un mode affichage uniquement (réglage désactivé) et un mode affichage avec réglage.

Initialisation à l'heure de compilation : au lieu de démarrer à 00:00:00, j'ai fait en sorte que le système démarre directement à l'heure de compilation du programme, en exploitant la macro préprocesseur __TIME__. Cela rend le réveil immédiatement fonctionnel dès le déploiement.

Les interrupteurs	Valeur	Opération
SW1, SW0	0, 0	Afficher l'heure actuelle (format 24h)
SW1, SW0	1, 1	Afficher l'heure actuelle (format 12h)
SW1, SW0	0, 1	Afficher l'heure d'alarme (format 24h)
SW1, SW0	1, 0	Afficher l'heure d'alarme (format 12h)
SW2	1	Alarme activée, LED0 allumée
SW2	0	Alarme désactivée, LED0 éteinte
SW3	1	Mode affichage uniquement
SW3	0	Mode affichage avec réglage
SW4	1	Réglage par appuis successifs
SW4	0	Réglage continu (pas de 0,5 s)
SW7, SW8, SW9	—	Sélection de la mélodie

Tableau 1 : Mapping des interrupteurs

III. Conception matérielle (Hardware)

1. Architecture globale

J'ai commencé par mettre en place toute la partie matérielle avant de passer à la programmation. J'ai défini les besoins en processeur, mémoire, PIOs et timers dans Platform Designer (Qsys). Une fois le système généré, je l'ai instancié dans un fichier top-level BDF dans Quartus Prime, puis j'ai lancé la synthèse, le placement-routage et le déploiement sur le FPGA.

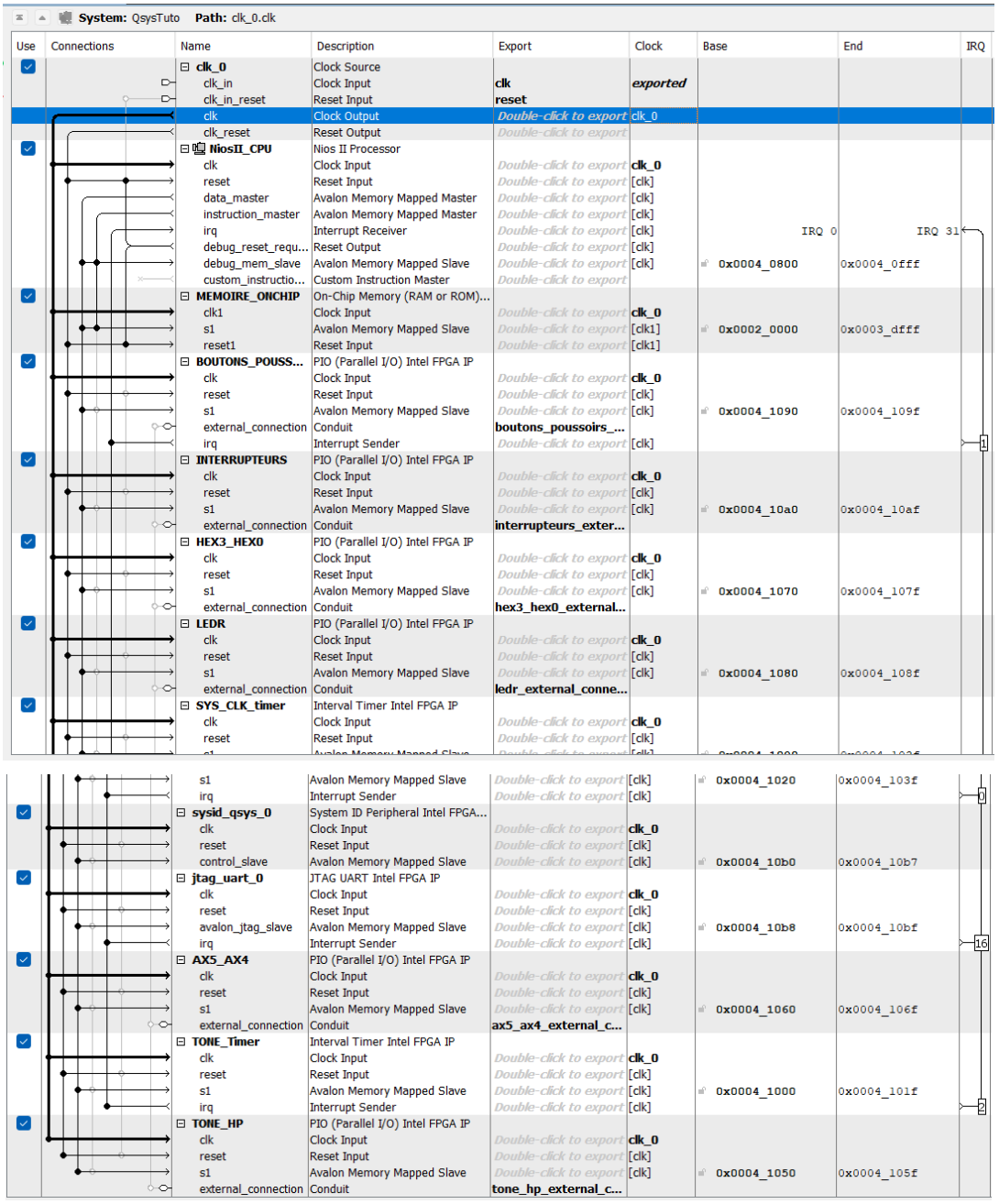


Figure 3 : Vue d'ensemble du système dans Platform Designer (Qsys) - PIO's

2. Cœur du système

Pour le cœur du système, j'ai utilisé un processeur Nios II connecté à une mémoire On-Chip (RAM) qui stocke les données et les instructions du programme C. Le processeur est cadencé par l'horloge à 50 MHz de la DE10-Lite. J'ai fait attention à bien dimensionner la mémoire, surtout que le BSP est configuré avec la Small C Library (-msmallc), ce qui limite les fonctions disponibles pour réduire la taille du binaire ELF.

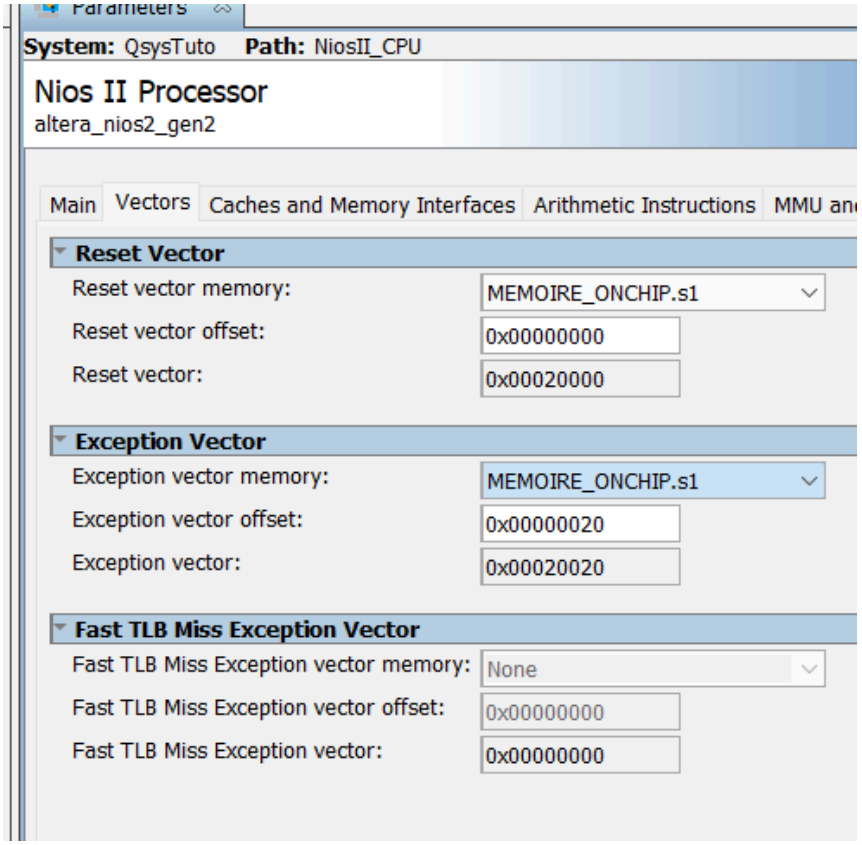


Figure 4 : Configuration du processeur Nios II dans Qsys

L'assignation des vecteurs de démarrage (Reset) et d'interruption (Exception) à la mémoire interne (`MEMOIRE_ONCHIP`) permet une exécution directe du code depuis le FPGA. Ce choix matériel garantit un temps d'accès rapide et déterministe, indispensable à la précision temps réel du réveil.

3. Interface avec les composants externes (PIOs)

J'ai interfacé chaque périphérique externe de la DE10-Lite avec le processeur Nios II via un PIO (Parallel I/O) configuré dans Qsys.
Le tableau ci-dessous montre comment j'ai dimensionné chaque PIO :

Périphérique (Nom Qsys)	Largeur	Direction	Rôle
HEX3_HEX0_BASE	32 bits	OUT	Afficheurs HEX0 à HEX3
AX5_AX4_BASE	16 bits	OUT	Afficheurs HEX4 et HEX5
INTERRUPTEURS_BASE	10 bits	IN	Switches SW0 à SW9
BOUTONS_POUSSOIRS_BASE	2 bits	IN	KEY0 et KEY1 (avec IRQ Rising Edge)
LEDR_BASE	10 bits	OUT	LEDs rouges (dont LED0 témoin alarme)
TONE_HP_BASE	1 bit	OUT	Haut-parleur (signal carré)

Tableau 2 : Dimensionnement des PIOs

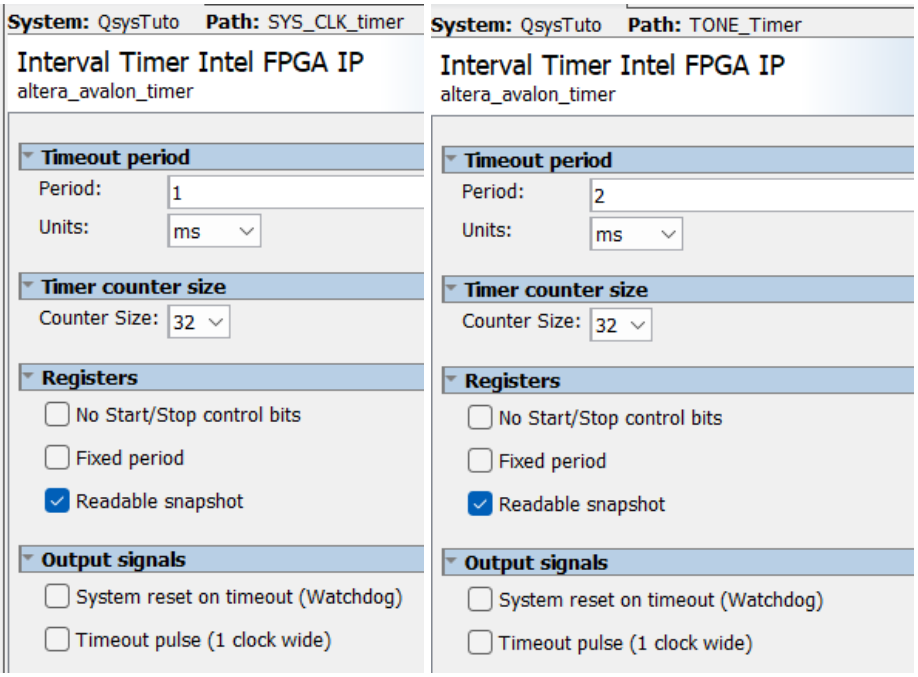
Pour justifier le dimensionnement des PIOs, j'ai raisonné de la manière suivante. Pour les afficheurs 7 segments HEX0 à HEX3, chaque afficheur nécessite 7 bits (un par segment, active-low), donc 4 afficheurs donnent 4 × 8 = 32 bits (j'ai arrondi à 8 bits par afficheur pour simplifier l'adressage, le bit de poids fort de chaque octet n'est pas utilisé). Pour HEX4 et HEX5, c'est le même raisonnement avec 2 × 8 = 16 bits. Les switches SW0 à SW9 sont au nombre de 10, donc un PIO de 10 bits en entrée. Les boutons poussoirs KEY0 et KEY1 font 2 bits en entrée, configurés en Rising Edge avec IRQ pour détecter les appuis. Les LEDs rouges LEDR0 à LEDR9 nécessitent 10 bits en sortie, même si dans ce projet je n'utilise que LED0 comme témoin d'alarme. Enfin, le haut-parleur ne nécessite qu'un seul bit en sortie pour générer le signal carré.

4. Gestion du temps

Pour la gestion du temps, j’ai configuré deux timers matériels dans Qsys :

SYS_CLK_TIMER (IRQ 0) : timer périodique de 1 ms qui sert de base de temps pour le comptage des secondes. L’ISR (Interrupt Service Routine) associée incrémente un compteur ; lorsqu’il atteint 1000, une seconde s’est écoulée et les variables heures/minutes/secondes sont mises à jour.

TONE_TIMER (IRQ 2) : timer dédié à la génération du signal audio. Sa période est ajustée dynamiquement pour produire la fréquence souhaitée (500 Hz pour le bip de base, ou les fréquences des notes pour les mélodies).



Figures 5 : Configurations des timers dans Qsys

Une fois les périphériques configurés et le système Qsys généré, je l'ai instancié dans un fichier top-level BDF dans Quartus Prime. C'est aussi dans ce schéma que j'ai placé les inverseurs entre les boutons poussoirs physiques et les PIOs pour avoir une logique active-high (le PIO reçoit 1 quand le bouton est pressé).

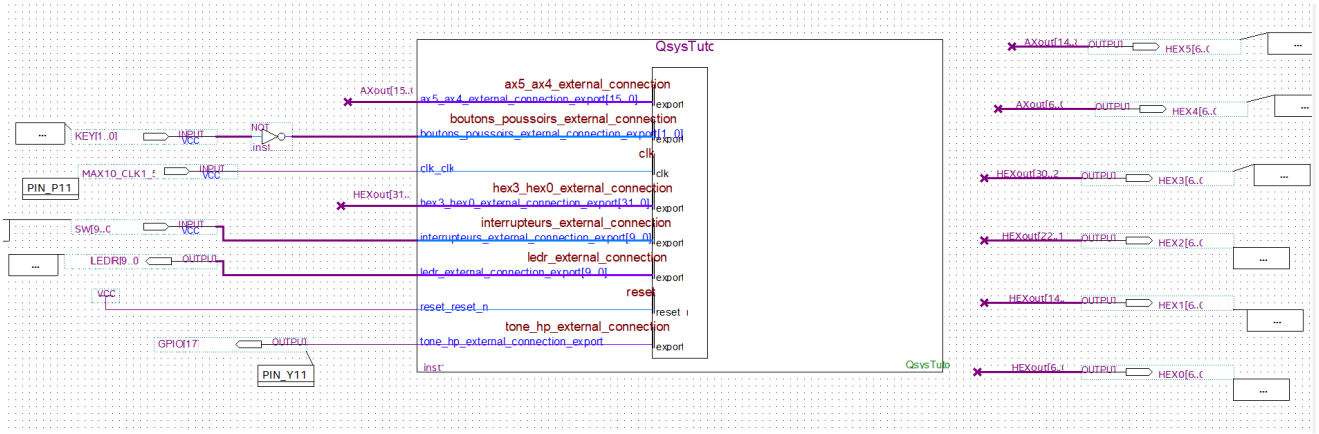


Figure 6 : Schéma BDF top-level avec inverseurs sur les boutons poussoirs

Remarque : À chaque modification, il faut que je régénère un nouveau BDF.

IV. Conception Logicielle (Software)

1. Architecture du code C

Pour la partie logicielle, j'ai structuré mon code C autour d'une boucle principale infinie (while(1)) et de deux routines d'interruption (ISR). J'ai créé un type Heure_t qui contient trois champs (heure, minute, seconde) de type alt_u8, et j'ai déclaré deux variables de ce type : heure_courante pour l'heure actuelle et heure_alarme pour l'heure d'alarme.

Concrètement, la boucle principale s'occupe de lire l'état des interrupteurs et de mettre à jour l'affichage sur les 7 segments, alors que les ISR gèrent la base de temps et la génération audio. J'ai trouvé que cette séparation rend le code beaucoup plus lisible et facile à déboguer.

```
/* Structure Heure_t et déclaration des variables d'état */
typedef struct {
    alt_u8 heure;    /* de 0 à 23 */
    alt_u8 minute;   /* de 0 à 59  */
    alt_u8 seconde;  /* de 0 à 59 aussi */
} Heure_t;
Heure_t heure_courante = {0, 0, 0};
Heure_t heure_alarme   = {0, 0, 0};
```

Figure 7 : Structure de données Heure_t et variables d'état

2. Gestion du temps réel

Pour gérer le temps, j'ai utilisé l'interruption du timer SYS_CLK_TIMER qui se déclenche toutes les millisecondes. Dans l'ISR, j'incrémente un compteur de millisecondes et quand il atteint 1000, je sais qu'une seconde s'est écoulée. À ce moment-là, j'incrémente les secondes, puis je gère les débordements en cascade : 60 secondes donne une minute, 60 minutes donne une heure, et 24 heures remet tout à zéro.
C'est aussi dans cette ISR que je vérifie si l'heure actuelle coïncide avec l'heure d'alarme. Si c'est le cas et que l'alarme est activée (SW2 = 1), le signal sonore se déclenche.

3. Génération du son

Le son est produit par l'ISR du TONE_TIMER. À chaque interruption, je toggle le bit de sortie TONE_HP_BASE, ce qui crée un signal carré sur le haut-parleur. La fréquence du son dépend de la période que j'ai configurée pour le timer. Pour le bip basique à 500 Hz, c'est assez simple.
Pour les mélodies, j'ai défini des tableaux de notes avec les fréquences correspondantes et les durées de chaque note. Le système parcourt ces tableaux séquentiellement en ajustant la période du timer pour chaque note. La sélection de la mélodie se fait par la combinaison des interrupteurs SW7, SW8 et SW9, ce qui donne accès à 8 mélodies différentes.

```
/*Tableau de notes (extrait : bip simple et Frère Jacques)
                et sélection de mélodie par SW9:SW7*/

typedef struct {
    alt_u16 freq_hz;
    alt_u16 duree_ms;
} Note_t;

/*Mélodie 0 - Bip simple 500 Hz (cahier des charges de base)*/
static const Note_t MEL_BIP[] = {
    {500, 400}, {0, 200}, {500, 400}, {0, 200}, {500, 800}
};

/* Mélodie 1 - Frère Jacques */
static const Note_t MEL_FRERE[] = {
    {261,300},{293,300},{329,300},{261,300},
    {329,300},{349,300},{392,600},
    {392,150},{349,150},{329,300},{261,300}
};

/* ... (mélodies 2 à 7 définies de la même façon) ... */

#define NB_NOTES(t)  ((alt_u8)(sizeof(t) / sizeof((t)[0])))

static const Melodie_t MELODIES[8] = {
    {MEL_BIP,     NB_NOTES(MEL_BIP)    },
    {MEL_FRERE,   NB_NOTES(MEL_FRERE)  },
    {MEL_GAMME,   NB_NOTES(MEL_GAMME)  },
    {MEL_LOUP,    NB_NOTES(MEL_LOUP)   },
    {MEL_DOUBLE,  NB_NOTES(MEL_DOUBLE) },
    {MEL_HEUREUX,NB_NOTES(MEL_HEUREUX) },
    {MEL_YUSUF,   NB_NOTES(MEL_YUSUF)  },
```

```
    {MEL_UMIYE,  NB_NOTES(MEL_UMIYE) },
};

/* --- Sélection dans la boucle principale --- */
alt_u8 mel_sel = (alt_u8)((sw >> 7) & 0x07u);   /* SW9:SW7 → 0..7 */
```

Figure 8 : Définition des mélodies et sélection par switches

4. Interface utilisateur

Dans la boucle principale, je lis en permanence l'état des interrupteurs et des boutons poussoirs pour savoir ce que l'utilisateur veut faire. Pour l'affichage, j'ai créé une lookup table qui convertit chaque chiffre décimal (0 à 9) en son équivalent 7 segments, puis j'écris les valeurs sur les PIOs HEX3_HEX0_BASE et AX5_AX4_BASE.

Pour le réglage de l'heure, j'ai implémenté deux modes selon SW4. En mode appuis successifs (SW4 = 1), chaque appui sur KEY0 ou KEY1 incrémente les minutes ou les heures d'une unité.

En mode continu (SW4 = 0), quand on maintient le bouton enfoncé, ça incrémente automatiquement toutes les 0,5 secondes. Pour détecter si le bouton est maintenu, je fais du polling sur le niveau logique du bouton, qui vaut 1 quand il est pressé grâce à l'inverseur BDF.

Le mode verrouillage (SW3 = 1) désactive le réglage complètement. Comme ça, on ne risque pas de modifier l'heure par accident quand on ne le veut pas.

V. Tests, Difficultés et Validation

1. Méthodologie de test

J'ai testé le système de manière progressive. J'ai commencé par vérifier que l'affichage sur les 7 segments fonctionnait correctement avec des valeurs statiques, puis j'ai activé le timer pour voir si le comptage des secondes était bon. Ensuite, j'ai testé le réglage de l'heure avec les boutons, et enfin le déclenchement de l'alarme et le son.

Pour les améliorations (format 12h, mélodies, réglage continu, verrouillage), j'ai ajouté et validé chaque fonctionnalité une par une pour m'assurer que rien ne cassait le fonctionnement de base.

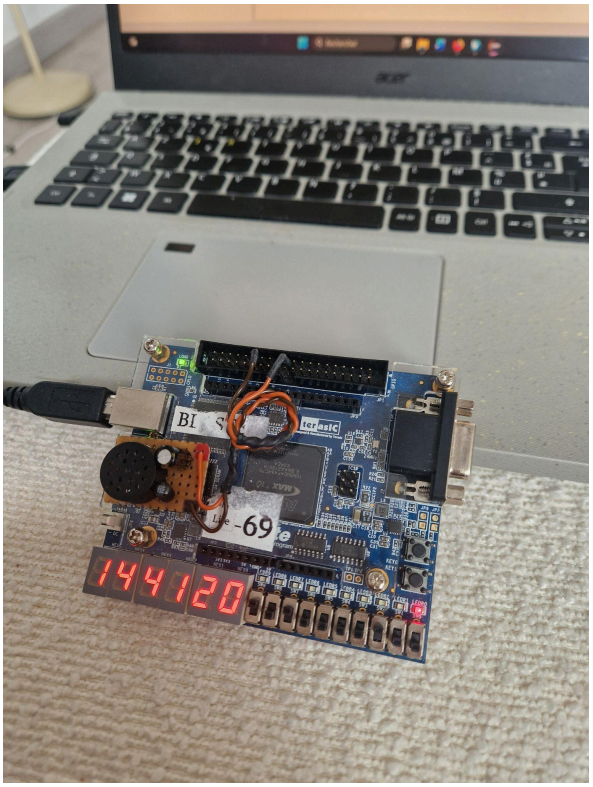


Figure 9 : Réveil en fonctionnement avec alarme active

2. Problèmes rencontrés et solutions

a. Problème avec sscanf et la Small C Library

J'ai rencontré pas mal de problèmes lors de la conception matérielle, ainsi que lors de la conception logicielle, j'ai voulu ajouter une amélioration, pour qu'à chaque fois que je compile, l'heure de la compilation sera prise par l'horloge afin d'avoir une alarme plus pro, j'ai donc utilisé sscanf, pour extraire l'heure de la compilation et la mettre dans mes variables de base qui débute à 0.

J'avais fait le choix d'utiliser sscanf ci-dessous en *figure 10*, mais malheureusement cet outil est inclus dans la bibliothèque stdio.h qui est considérablement lourd en kilo-octet. J'ai donc eu des problèmes de undefined reference to sscanf, qui veut dire puis j'ai compris qu'en fait cela est dû à la configuration du processeur Nios II, car le BSP est configuré avec l'option Small C Library (`-msmallc`) donc les fonctions complexes et gourmandes en mémoire sont supprimés pour réduire la taille du fichier binaire elf.

```c
// __TIME__ contient par exemple "14:30:05"
/*Heure_t heure_courante;*/
int h = 0, m = 0, s = 0;

Heure_t heure_courante = {0, 0, 0};
Heure_t heure_alarme   = {0, 0, 0};

// sscanf extrait les nombres de la chaîne "14:30:05"
if (sscanf(__TIME__, "%d:%d:%d", &h, &m, &s) == 3) {
    heure_courante.heure   = (alt_u8)h;
    heure_courante.minute  = (alt_u8)m;
    heure_courante.seconde = (alt_u8)s;
}
/*-------------------------------------------------------
```

Figure 10 : Première approche avec sscanf (non fonctionnelle)

b. Solution retenue : décodage ASCII manuel

En systèmes embarqués, chaque octet de mémoire programme compte donc j'ai dû partir sur une autre voie, ci-après en *figure 11*. J'ai donc fait le code pour qu'il décode manuellement les chaînes de caractères de l'heure de compilation, je préfère faire travailler le processeur plutôt que de sacrifier de la mémoire programme.

```
 * Variables d'état
 * ------------------------------------------------------------ */
Heure_t heure_courante = {0, 0, 0};
Heure_t heure_alarme   = {0, 0, 0};

/* Extraction manuelle de l'heure du PC (format "HH:MM:SS")
 * Plus léger que sscanf pour la Small C Library du Nios II */
const char *t = __TIME__;
heure_courante.heure   = (alt_u8)((t[0] - '0') * 10 + (t[1] - '0'));
heure_courante.minute  = (alt_u8)((t[3] - '0') * 10 + (t[4] - '0'));
heure_courante.seconde = (alt_u8)((t[6] - '0') * 10 + (t[7] - '0'));
```

Figure 12 : Décodage ASCII manuel de __TIME__ (solution retenue)

Finalement, cette solution est bien meilleure. Elle ne repose que sur des opérations arithmétiques simples (soustraction et multiplication par 10), elle ne dépend pas de stdio.h, et elle est plus performante. J'ai préféré faire travailler le processeur plutôt que de gaspiller de la mémoire programme.

c. Transition vers un développement incrémental

Au début du projet, mon manque de familiarité avec le langage C m'a d'abord poussé à écrire le code de l'alarme d'un seul bloc, en m'appuyant massivement sur des exemples en ligne et des outils d'IA. Cette approche monolithique s'est avérée contre-productive, générant un grand nombre d'erreurs et de bugs très difficiles à isoler.
Suite aux conseils de mon professeur et d'un camarade, j'ai radicalement repensé ma méthode de travail pour adopter un développement incrémental. J'ai appris à concevoir d'abord la logique sur papier, puis à coder et tester chaque sous-fonctionnalité individuellement sur la carte matérielle. Ce n'est qu'une fois un bloc validé qu'il est intégré au reste du programme.
L'assemblage progressif de ces parties m'a finalement permis d'obtenir un code robuste. Cette démarche de validation pas-à-pas a grandement facilité le débogage et constitue sans doute l'apprentissage méthodologique le plus important de ce projet.

VI. Conclusion et Perspectives

1. Bilan

Ce mini-projet m’a permis de découvrir et de pratiquer le flot complet de conception conjointe matérielle/logicielle sur cible FPGA. J’ai réussi à concevoir un réveil électronique fonctionnel qui intègre toutes les fonctionnalités demandées dans le cahier des charges de base, ainsi que plusieurs améliorations telles que le format 12h/24h, le réglage continu à 500 ms et la gestion de mélodies complexes.

1.1 - Apprentissage technique et contraintes

Ce que j'ai le plus apprécié, c'est la confrontation aux contraintes réelles de l'embarqué. La problématique liée à la *Small C Library* (impossibilité d'utiliser `sscanf` par manque de ressources mémoire) a été un moment clé : cela m'a fait comprendre qu'en système embarqué, les ressources sont limitées et qu'on ne peut pas programmer comme sur un PC. Le décodage ASCII manuel de la macro `__TIME__` pour initialiser l'horloge à la compilation illustre bien ce type de compromis nécessaire entre richesse fonctionnelle et optimisation.

1.2 - Maîtrise du Co-design

J'ai également beaucoup appris sur la gestion des interruptions et sur l'importance de la cohérence entre le schéma matériel et le logiciel. L'utilisation d'inverseurs sur les boutons poussoirs dans le schéma BDF m'a montré qu'un détail matériel change radicalement la logique du code. Pour garantir la fiabilité du système, j'ai dû mettre en œuvre des **sections critiques** (désactivation temporaire des IRQ) afin de sécuriser le partage de données entre les routines d'interruption et la boucle principale, évitant ainsi tout risque de corruption de données.

1.3 - Méthodologie de travail

Au-delà de la technique, l'enseignement le plus précieux reste méthodologique. L'abandon d'une approche de codage "monolithique" au profit d'un **développement incrémental**, donc coder, tester et valider chaque bloc individuellement avant intégration, ça été le facteur déterminant de la réussite du projet. Cette approche pas-à-pas m'a permis de livrer un système stable et facile à déboguer, tout en me faisant gagner une compétence triviale.

1.4 - Perspectives

Pour aller plus loin, le système pourrait être amélioré par l'ajout d'un composant matériel de type RTC (Real Time Clock) externe pour maintenir l'heure en cas de coupure d'alimentation, ou encore par l'implémentation d'une fonction "Snooze"

utilisant les boutons poussoirs pour stopper temporairement l'alarme sans la désactiver totalement.

Une première piste d'évolution majeure concerne la robustesse et la fiabilité du système face aux interruptions d'alimentation. L'intégration d'un composant matériel de type RTC (Real Time Clock) externe permettrait de maintenir le décompte de l'heure exacte même en cas de coupure de courant. En complément, l'ajout d'une fonction « Snooze » (rappel d'alarme) via les boutons poussoirs offrirait une expérience utilisateur plus proche des standards industriels, permettant de suspendre temporairement l'alerte sonore sans pour autant désactiver totalement le cycle de réveil.

L'interface homme-machine pourrait également être enrichie par le remplacement ou l'adjonction d'écrans LCD ou OLED aux afficheurs 7 segments actuels. Ce changement technologique permettrait un affichage beaucoup plus riche, autorisant non seulement une lecture plus aisée de l'heure, mais aussi l'intégration d'informations complémentaires telles que la date complète, le jour de la semaine ou encore la température ambiante via un capteur dédié.

Sur le plan de la restitution sonore, bien que la génération de mélodies par modulation de fréquence soit fonctionnelle, l'utilisation d'un DAC (Convertisseur Numérique-Analogique) constituerait une amélioration significative. Ce composant permettrait de s'affranchir des signaux carrés pour reproduire des formes d'ondes plus complexes et plus harmonieuses, augmentant ainsi considérablement la fidélité et la qualité auditive des alarmes proposées.

Enfin, d'un point de vue purement technique et académique, il serait pertinent de porter ce projet vers une architecture entièrement matérielle en langage VHDL. Une telle migration permettrait de réaliser une analyse comparative rigoureuse entre une approche SoPC basée sur un processeur Nios II et une solution « full hardware ». Cela permettrait d'évaluer précisément les gains en termes de performances temporelles et d'optimisation des ressources logiques au sein du FPGA.

VII. Bibliographie et références techniques

Ressources

- Intel FPGA — Nios II Software Developer Handbook.
- Terasic — DE10-Lite User Manual.
- Intel FPGA — Quartus Prime Handbook.
- Support de cours et tutoriaux de TD (M. Dadouche).

Références techniques du système

J'ai repris le tableau de mon manuel d'utilisation (ci-joint) que j'ai fait pour mon alarme en section 9.

SW	Bas (0)	Haut (1)	Détail
SW0	—	—	Combiné avec SW1 → sélection affichage/format (voir §2.2)
SW1	—	—	Combiné avec SW0 → sélection affichage/format (voir §2.2)
SW2	Alarme OFF	Alarme ON	Contrôle l'activation de l'alarme. LED0 allumée si = 1. Baisser pour arrêter la sonnerie.
SW3	Réglage	Affichage seul	À 0 : KEY0/KEY1 modifient l'heure. À 1 : les boutons sont inertes.
SW4	Continu 500ms	Successif	Mode de réglage. À 0 : maintien = répétition. À 1 : 1 appui = 1 incrément.
SW5	—	—	Non utilisé dans cette version
SW6	—	—	Non utilisé dans cette version
SW7	Bit 0	Bit 0	LSB de sélection mélodie. Contribue au numéro mélodie (0 à 7).
SW8	Bit 1	Bit 1	Bit intermédiaire de sélection mélodie.
SW9	Bit 2	Bit 2	MSB de sélection mélodie. SW9=1,SW8=1,SW7=1 → mélodie 7.

Tableau 3 : Tableau des références à tester

Annexes

Code complet hello_world.c

```c
/* ============================================================
 * Daniel BAL - M1 MESI - hello_world.c
 * Mini-Projet : Réveil Électronique - Nios II / DE10-Lite
 * M1-PAIP - F. Dadouche - Université de Strasbourg
 *
 * Nios II/fast @ 50 MHz | Quartus Prime 18.1 | HAL
 * ============================================================*/


 /*SECTION 1 : Les importations des différentes bibliothèques*/

#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_avalon_timer_regs.h"
#include "sys/alt_irq.h"
#include "alt_types.h"


/* SECTION 2 : Table de décodage 7 segments */

static const alt_u8 SEG7[10] = {
    0xC0,   /* 0 */
    0xF9,   /* 1 */
    0xA4,   /* 2 */
    0xB0,   /* 3 */
    0x99,   /* 4 */
    0x92,   /* 5 */
    0x82,   /* 6 */
    0xF8,   /* 7 */
    0x80,   /* 8 */
    0x90    /* 9 */
};

#define SEG7_ETEINT  0xFF   /* tous segments éteints */

/* ============================================================
 * SECTION 3 : STRUCTURE HEURE
 * ============================================================ */

typedef struct {
    alt_u8 heure;    /* de 0 à 23 */
    alt_u8 minute;   /* de 0 à 59  */
    alt_u8 seconde;  /* de 0 à 59 aussi */
} Heure_t;

/* ============================================================
 * SECTION 4 : MÉLODIES D'ALARME
 * 8 mélodies sélectionnables via SW9:SW7.
 * ============================================================ */

typedef struct {
    alt_u16 freq_hz;
    alt_u16 duree_ms;
} Note_t;

/* Mélodie 0 - Bip simple 500 Hz (cahier des charges de base) */
static const Note_t MEL_BIP[] = {
    {500, 400}, {0, 200}, {500, 400}, {0, 200}, {500, 800}
};

/* Mélodie 1 - Frère Jacques */
static const Note_t MEL_FRERE[] = {
    {261,300},{293,300},{329,300},{261,300},
    {329,300},{349,300},{392,600},
    {392,150},{349,150},{329,300},{261,300}
};

/* Mélodie 2 - Gamme montante */
static const Note_t MEL_GAMME[] = {
    {261,150},{293,150},{329,150},{349,150},
    {392,150},{440,150},{493,150},{523,400},{0,200}
};

/* Mélodie 3 - Vallée des loups */
```

```
static const Note_t MEL_LOUP[] = {
    {293, 400}, {0, 10}, {293, 400}, {0, 10}, {331, 400}, {0, 10}, {350, 400}, {0, 10}, {350, 400},
    {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {293, 400}, {0, 10}, {293, 400}, {0, 10},
    {331, 400}, {0, 10}, {331, 400}, {0, 10}, {261, 400}, {0, 10}, {261, 400}, {0, 10}, {293, 400},
    {0, 10}, {293, 400}, {0, 10}, {293, 400}, {0, 10}, {293, 400}, {0, 10}, {331, 400}, {0, 10},
    {331, 400}, {0, 10}, {331, 400}, {0, 10}, {350, 400}, {0, 10}, {350, 400}, {0, 10}, {350, 400},
    {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {293, 400}, {0, 10},
    {293, 400}, {0, 10}, {293, 400}, {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {331, 400},
    {0, 10}, {261, 400}, {0, 10}, {261, 400}, {0, 10}, {261, 400}, {0, 10}, {293, 400}, {0, 10},
    {293, 400}, {0, 10}, {293, 400}, {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {350, 400},
    {0, 10}, {350, 400}, {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {293, 400}, {0, 10},
    {293, 400}, {0, 10}, {331, 400}, {0, 10}, {331, 400}, {0, 10}, {261, 400}, {0, 10}, {261, 400},
    {0, 10}, {293, 400}, {0, 1500}
};

/* Mélodie 4 - Double bip */
static const Note_t MEL_DOUBLE[] = {
    {880,100},{0,60},{880,100},{0,400}
};

/* Mélodie 5 - Alarme Heureux */
static const Note_t MEL_HEUREUX[] = {
    {330, 1000}, {0, 140}, {330, 233}, {0, 140}, {392, 248}, {0, 140}, {494, 270}, {0, 120}, {392, 250},
    {0, 180}, {440, 690}, {0, 30}, {494, 200}, {0, 500}, {330, 230}, {0, 190}, {392, 130}, {0, 220},
    {494, 230}, {0, 135}, {392, 238}, {0, 125}, {440, 340}, {0, 10}, {494, 396}, {0, 10}, {392, 144},
    {0, 34}, {349, 146}, {0, 20}, {330, 225}, {0, 145}, {330, 233}, {0, 140}, {392, 248}, {0, 140},
    {494, 270}, {0, 120}, {392, 250}, {0, 180}, {440, 690}, {0, 30}, {494, 200}, {0, 500}, {330, 230},
    {0, 190}, {392, 130}, {0, 220}, {494, 230}, {0, 135}, {392, 238}, {0, 125}, {440, 340}, {0, 10},
    {494, 396}, {0, 10}, {392, 144}, {0, 34}, {349, 146}, {0, 20}, {330, 225}, {0, 420}, {330, 670},
    {0, 15}, {440, 460}, {0, 25}, {392, 636}, {0, 50}, {349, 573}, {0, 158}, {349, 1080}, {0, 20},
    {494, 410}, {0, 60}, {440, 563}, {0, 40}, {392, 385}, {0, 336}, {494, 896}, {0, 50}, {494, 147},
    {0, 55}, {440, 112}, {0, 75}, {440, 934}, {0, 87}, {440, 169}, {0, 66}, {392, 164}, {0, 60},
    {494, 724}, {0, 82}, {494, 156}, {0, 49}, {440, 126}, {0, 60}, {440, 705}, {0, 257}, {440, 142},
    {0, 68}, {392, 106}, {0, 79}, {392, 874}, {0, 25}, {440, 309}, {349, 642}, {0, 25}, {330, 285},
    {0, 431}, {330, 126}, {0, 20}, {349, 100}, {0, 30}, {392, 75}, {0, 20}, {349, 68}, {0, 25},
    {330, 994}, {0, 377}, {294, 317}, {0, 87}, {330, 284}, {0, 49}, {349, 328}, {0, 150}, {392, 1480}
};

/* Mélodie 6 - Jingle court */
static const Note_t MEL_YUSUF[] = { /*Sami Yusuf */
   {587, 200}, {0, 80},   {587, 400}, {659, 214}, {587, 214}, {523, 214}, {494, 214},{440, 214}, {392, 214},
        {440, 214}, {494, 214}, {392, 600},{0, 300}, {392, 214},{440, 214}, {494, 214}, {523, 214}, {587, 428},
        {523, 428}, {523, 214},{494, 214}, {440, 214}, {392, 214}, {440, 428}, {392, 428},{0, 400}, {587, 428},
   {587, 214},{659, 214}, {587, 214}, {523, 214}, {494, 428},{523, 214}, {494, 214},{440, 214},{392, 214},
   {440, 428}, {392, 428}, {0, 1500}
};

/* Mélodie 7 - Salati Ummiyye */
static const Note_t MEL_UMIYE[] = {
    {494, 800}, {440, 1600}, {494, 800}, {440, 800}, {494, 800}, {523, 800}, {494, 800}, {440, 1600}, {0, 300},
    {494, 800}, {523, 800}, {494, 800}, {440, 1600}, {0, 600}, {494, 800}, {494, 800}, {494, 800}, {440, 800},
    {494, 800}, {0, 150}, {523, 800}, {587, 400}, {523, 1200}, {494, 800}, {440, 800}, {494, 400}, {440, 1200},
    {392, 800}, {440, 800}, {494, 800}, {0, 300}, {523, 1600}, {440, 1200}, {494, 2400}, {0, 300}, {523, 800},
    {587, 800}, {523, 800}, {494, 2400}, {0, 300}, {440, 800}, {494, 800}, {440, 3200}, {0, 3000}
};
#define NB_NOTES(t)  ((alt_u8)(sizeof(t) / sizeof((t)[0])))

typedef struct {
    const Note_t *notes;
    alt_u8        nb_notes;
} Melodie_t;

static const Melodie_t MELODIES[8] = {
    {MEL_BIP,    NB_NOTES(MEL_BIP)    },
    {MEL_FRERE,  NB_NOTES(MEL_FRERE)  },
    {MEL_GAMME,  NB_NOTES(MEL_GAMME)  },
    {MEL_LOUP, NB_NOTES(MEL_LOUP)},
    {MEL_DOUBLE, NB_NOTES(MEL_DOUBLE)},
    {MEL_HEUREUX,  NB_NOTES(MEL_HEUREUX)  },
    {MEL_YUSUF, NB_NOTES(MEL_YUSUF)},
    {MEL_UMIYE, NB_NOTES(MEL_UMIYE)},
};

/* ============================================================
 * SECTION 5 : VARIABLES GLOBALES PARTAGÉES (main <-> ISR)
 *
 * "volatile" obligatoire : interdit au compilateur de mettre
 * ces variables en cache registre entre les accès ISR/main.
 * ============================================================ */

/* Compteur ms - incrémenté par l'ISR du SYS_CLK_TIMER toutes les 1ms */
static volatile alt_u32 g_ms = 0;
```

```c
/* Appuis boutons capturés par l'ISR (bit0=KEY0, bit1=KEY1)
 * Mis à 1 dans l'ISR sur front montant (après inverseur BDF),
 * lus et remis à 0 dans le main.                              */
static volatile alt_u8 g_appui_bouton = 0;

/* Etat du générateur sonore */
static volatile struct {
    const Note_t *notes;
    alt_u8        nb_notes;
    alt_u8        idx;
    alt_u32       toggles_restants;
    alt_u8        actif;
    alt_u8        etat_pin;
} g_son;

/* ============================================================
 * SECTION 6 : ISR - TIMER BASE DE TEMPS (SYS_CLK_TIMER, IRQ 0)
 *
 * Période = 1 ms (LOAD_VALUE = 49999 à 50 MHz, configuré Qsys).
 * Incrémente g_ms à chaque interruption.
 * g_ms remplace tout usleep() dans le code.
 * ============================================================ */
static void isr_timer_base(void *context)
{
    (void)context;
    /* Acquitter l'IRQ : effacer le bit TO du registre STATUS */
    IOWR_ALTERA_AVALON_TIMER_STATUS(SYS_CLK_TIMER_BASE, 0);
    g_ms++;
}

/* ============================================================
 * SECTION 7 : ISR - TIMER SON (TONE_TIMER, IRQ 2)
 *
 * Génère un signal carré à fréquence variable par toggle du pin.
 *
 * Pour une note de fréquence F Hz :
 *   demi-période (ticks) = 50 000 000 / (2 × F) - 1
 *   nb toggles pour durée D ms = 2 × F × D / 1000
 *
 * Pour un silence (freq_hz = 0) :
 *   période fixe 1 ms, pin à 0, on décompte la durée en ms.
 * ============================================================ */
static void charger_note(alt_u8 idx);  /* déclaration anticipée */

static void isr_timer_son(void *context)
{
    (void)context;

    IOWR_ALTERA_AVALON_TIMER_STATUS(TONE_TIMER_BASE, 0);

    if (!g_son.actif) {
        IOWR_ALTERA_AVALON_PIO_DATA(TONE_HP_BASE, 0);
        return;
    }

    /* Toggle le pin si ce n'est pas un silence */
    if (g_son.notes[g_son.idx].freq_hz != 0) {
        g_son.etat_pin ^= 1u;
        IOWR_ALTERA_AVALON_PIO_DATA(TONE_HP_BASE, g_son.etat_pin);
    }

    /* Décompter et passer à la note suivante si terminée */
    if (g_son.toggles_restants > 0) {
        g_son.toggles_restants--;
    }
    if (g_son.toggles_restants == 0) {
        g_son.idx = (g_son.idx + 1u) % g_son.nb_notes;
        charger_note(g_son.idx);
    }
}

static void charger_note(alt_u8 idx)
{
    const Note_t *n = &g_son.notes[idx];
    alt_u32 period;

    if (n->freq_hz == 0) {
        period = 49999u;                        /* 1 ms */
        g_son.toggles_restants = n->duree_ms;   /* on compte des ms */
        IOWR_ALTERA_AVALON_PIO_DATA(TONE_HP_BASE, 0);
    } else {
```

```
        period = (50000000u / (2u * (alt_u32)n->freq_hz)) - 1u;
        g_son.toggles_restants =
            (alt_u32)2u * n->freq_hz * n->duree_ms / 1000u;
        if (g_son.toggles_restants == 0u) g_son.toggles_restants = 1u;
    }

    IOWR_ALTERA_AVALON_TIMER_CONTROL(TONE_TIMER_BASE, 0);
    IOWR_ALTERA_AVALON_TIMER_STATUS (TONE_TIMER_BASE, 0);
    IOWR_ALTERA_AVALON_TIMER_PERIODL(TONE_TIMER_BASE,  period        & 0xFFFFu);
    IOWR_ALTERA_AVALON_TIMER_PERIODH(TONE_TIMER_BASE, (period >> 16) & 0xFFFFu);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TONE_TIMER_BASE,
        ALTERA_AVALON_TIMER_CONTROL_START_MSK |
        ALTERA_AVALON_TIMER_CONTROL_CONT_MSK  |
        ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);
}

static void son_demarrer(alt_u8 mel_idx)
{
    const Melodie_t *m = &MELODIES[mel_idx & 0x07u];
    g_son.notes    = m->notes;
    g_son.nb_notes = m->nb_notes;
    g_son.idx      = 0;
    g_son.etat_pin = 0;
    g_son.actif    = 1;
    charger_note(0);
}

static void son_arreter(void)
{
    g_son.actif = 0;
    IOWR_ALTERA_AVALON_TIMER_CONTROL(TONE_TIMER_BASE, 0);
    IOWR_ALTERA_AVALON_TIMER_STATUS (TONE_TIMER_BASE, 0);
    IOWR_ALTERA_AVALON_PIO_DATA(TONE_HP_BASE, 0);
}

/* ============================================================
 * SECTION 8 : ISR - BOUTONS POUSSOIRS (IRQ 1)
 *
 * Avec inverseur BDF : bouton appuyé → front MONTANT sur le PIO.
 * → Qsys doit être configuré en "Rising Edge".
 *
 * L'ISR lit l'edge capture (bit=1 si front détecté),
 * sauvegarde dans g_appui_bouton, remet l'edge capture à 0.
 * ============================================================ */
static void isr_boutons(void *context)
{
    (void)context;

    /* Lire le registre Edge Capture du PIO boutons */
    alt_u32 capture = IORD_ALTERA_AVALON_PIO_EDGE_CAP(BOUTONS_POUSSOIRS_BASE)
                      & 0x03u;

    /* Mémoriser les appuis pour le main */
    g_appui_bouton |= (alt_u8)capture;

    /* Remettre le registre Edge Capture à 0 (obligatoire) */
    IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BOUTONS_POUSSOIRS_BASE, 0);
}

/* ============================================================
 * SECTION 9 : AFFICHAGE 7 SEGMENTS
 *
 * Layout :  AX5 AX4 : HEX3 HEX2 : HEX1 HEX0
 *           ← heures → ← minutes→ ←secondes →
 * ============================================================ */
static void afficher_heure(const Heure_t *h)
{
    alt_u32 bas =
        ((alt_u32)SEG7[h->seconde % 10]       ) |  /* HEX0 */
        ((alt_u32)SEG7[h->seconde / 10] <<  8 ) |  /* HEX1 */
        ((alt_u32)SEG7[h->minute  % 10] << 16 ) |  /* HEX2 */
        ((alt_u32)SEG7[h->minute  / 10] << 24 );   /* HEX3 */

    alt_u16 haut =
        ((alt_u16)SEG7[h->heure % 10]       ) |    /* AX4 */
        ((alt_u16)SEG7[h->heure / 10] <<  8 );     /* AX5 */

    IOWR_ALTERA_AVALON_PIO_DATA(HEX3_HEX0_BASE, bas);
    IOWR_ALTERA_AVALON_PIO_DATA(AX5_AX4_BASE,   haut);
}

static void afficheurs_eteindre(void)
```

```
{
    alt_u32 b = ((alt_u32)SEG7_ETEINT      ) |
                ((alt_u32)SEG7_ETEINT <<  8) |
                ((alt_u32)SEG7_ETEINT << 16) |
                ((alt_u32)SEG7_ETEINT << 24);
    IOWR_ALTERA_AVALON_PIO_DATA(HEX3_HEX0_BASE, b);
    IOWR_ALTERA_AVALON_PIO_DATA(AX5_AX4_BASE,
        (alt_u16)SEG7_ETEINT | ((alt_u16)SEG7_ETEINT << 8));
}

/* ============================================================
 * SECTION 10 : HORLOGE – GESTION DU TEMPS
 * ============================================================ */
static void inc_seconde(Heure_t *h)
{
    if (++h->seconde >= 60) { h->seconde = 0;
    if (++h->minute  >= 60) { h->minute  = 0;
    if (++h->heure   >= 24)   h->heure   = 0; }}
}

static void inc_minute(Heure_t *h)
{
    if (++h->minute >= 60) h->minute = 0; /*Dès que 60 min est atteint on recommence de 0*/
}

static void inc_heure(Heure_t *h)
{
    if (++h->heure >= 24) h->heure = 0; /*Lorsque l'heure atteint 24, on recommence à 0*/
}

/* ============================================================
 * SECTION 11 : AMÉLIORATIONS
 *
 *  1. Format 12h / 24h  → SW1
 *  2. Sélection mélodie → SW9:SW7
 *  3. Réglage continu / successif → SW4
 *  4. Mode affichage seul → SW3
 * ============================================================ */
static Heure_t convertir_12h(const Heure_t *h)
{
    Heure_t r = *h;
    if      (r.heure == 0) r.heure = 12;
    else if (r.heure > 12) r.heure -= 12;
    return r;
}

/* ============================================================
 * SECTION 12 : MAIN
 * ============================================================ */
int main(void)
{
    /* ------------------------------------------------------
     * Initialisation des sorties
     * ------------------------------------------------------ */
    afficheurs_eteindre();
    IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE,    0);
    IOWR_ALTERA_AVALON_PIO_DATA(TONE_HP_BASE, 0);

    /* ------------------------------------------------------
     * SYS_CLK_TIMER (IRQ 0) – base de temps 1 ms
     * Timer déjà configuré à 1ms dans Qsys.
     * On active son IRQ et on enregistre l'ISR.
     * ------------------------------------------------------ */
    IOWR_ALTERA_AVALON_TIMER_STATUS(SYS_CLK_TIMER_BASE, 0);
    IOWR_ALTERA_AVALON_TIMER_CONTROL(SYS_CLK_TIMER_BASE,
        ALTERA_AVALON_TIMER_CONTROL_START_MSK |
        ALTERA_AVALON_TIMER_CONTROL_CONT_MSK  |
        ALTERA_AVALON_TIMER_CONTROL_ITO_MSK);

    alt_ic_isr_register(
        SYS_CLK_TIMER_IRQ_INTERRUPT_CONTROLLER_ID,
        SYS_CLK_TIMER_IRQ,
        isr_timer_base, NULL, 0x0);

    /* ------------------------------------------------------
     * BOUTONS POUSSOIRS (IRQ 1)
     *
     * Avec inverseur BDF → Qsys en Rising Edge.
     * Étapes du cours (slide 149) :
     *   1. Activer le masque IRQ
     *   2. Reset du registre Edge Capture
     *   3. Enregistrer l'ISR
```

```
 * ------------------------------------------------------- */
IOWR_ALTERA_AVALON_PIO_IRQ_MASK(BOUTONS_POUSSOIRS_BASE, 0x03u);
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(BOUTONS_POUSSOIRS_BASE, 0x00u);

alt_ic_isr_register(
    BOUTONS_POUSSOIRS_IRQ_INTERRUPT_CONTROLLER_ID,
    BOUTONS_POUSSOIRS_IRQ,
    isr_boutons, NULL, 0x0);

/* ------------------------------------------------------
 * TONE_TIMER (IRQ 2) – éteint au démarrage
 * ------------------------------------------------------- */
IOWR_ALTERA_AVALON_TIMER_CONTROL(TONE_TIMER_BASE, 0);
IOWR_ALTERA_AVALON_TIMER_STATUS (TONE_TIMER_BASE, 0);

alt_ic_isr_register(
    TONE_TIMER_IRQ_INTERRUPT_CONTROLLER_ID,
    TONE_TIMER_IRQ,
    isr_timer_son, NULL, 0x0);

/* ------------------------------------------------------
 * Variables d'état
 * ------------------------------------------------------ */

/*TEST HEURE ACTUELLE---------------------------------------- */

/* ------------------------------------------------------
 * Variables d'état
 * ------------------------------------------------------ */
Heure_t heure_courante = {0, 0, 0};
Heure_t heure_alarme   = {0, 0, 0};

/* Extraction manuelle de l'heure du PC (format "HH:MM:SS")
 * Plus léger que sscanf pour la Small C Library du Nios II */
const char *t = __TIME__;
heure_courante.heure   = (alt_u8)((t[0] - '0') * 10 + (t[1] - '0'));
heure_courante.minute  = (alt_u8)((t[3] - '0') * 10 + (t[4] - '0'));
heure_courante.seconde = (alt_u8)((t[6] - '0') * 10 + (t[7] - '0'));

alt_u32 t_dernier_tick    = g_ms;
alt_u32 t_dernier_continu = g_ms;

alt_u8 alarme_sonne  = 0;
alt_u8 sw2_precedent = 0;
alt_u8 mel_precedent = 0xFF;

/* FIX BUG 1 : drapeau "bouton maintenu" initialisé par l'ISR.
 * Garantit que le mode continu ne s'active JAMAIS tant qu'un
 * vrai appui ISR n'a pas été reçu. Résout l'emballement.      */
/*alt_u8 btn_maintenu = 0; je ne l'utilise plus car il me pose des problèmes de défilement*/

alt_u32 t_next_m = 0;
alt_u32 t_next_h = 0;

/* ======================================================
 * BOUCLE PRINCIPALE
 * ====================================================== */
while (1)
{
    alt_u32 maintenant = g_ms;

    /* ------------------------------------------------------
     * Lecture des switches
     * ------------------------------------------------------ */
    alt_u32 sw     = IORD_ALTERA_AVALON_PIO_DATA(INTERRUPTEURS_BASE)
                   & 0x3FFu;

    alt_u8 sw0     = (sw >> 0) & 1u;
    alt_u8 sw1     = (sw >> 1) & 1u;
    alt_u8 sw2     = (sw >> 2) & 1u;
    alt_u8 sw3     = (sw >> 3) & 1u;
    alt_u8 sw4     = (sw >> 4) & 1u;
    alt_u8 mel_sel = (alt_u8)((sw >> 7) & 0x07u);

    alt_u8 afficher_alarme = sw0 ^ sw1;
    alt_u8 format_12h      = sw1;

    /* ------------------------------------------------------
     * LED témoin alarme
     * ------------------------------------------------------ */
    IOWR_ALTERA_AVALON_PIO_DATA(LEDR_BASE, sw2 ? 0x001u : 0x000u);
```

```
/* ---------------------------------------------------------
 * HORLOGE : tick 1 seconde
 * --------------------------------------------------------- */
if ((maintenant - t_dernier_tick) >= 1000u) {
    t_dernier_tick += 1000u;
    inc_seconde(&heure_courante);
}

/* ---------------------------------------------------------
 * ALARME : déclenchement
 * --------------------------------------------------------- */
if (sw2) {
    alt_u8 correspondance =
        (heure_courante.heure   == heure_alarme.heure  ) &&
        (heure_courante.minute  == heure_alarme.minute ) &&
        (heure_courante.seconde == heure_alarme.seconde);

    if (correspondance && (!alarme_sonne || mel_sel != mel_precedent)) {
        alarme_sonne  = 1;
        mel_precedent = mel_sel;
        son_demarrer(mel_sel);
    }
}

/* ALARME : arrêt sur front descendant de SW2 */
if (sw2_precedent && !sw2) {
    alarme_sonne = 0;
    son_arreter();
}
sw2_precedent = sw2;

/* ---------------------------------------------------------
 * Lecture des appuis - SECTION CRITIQUE
 * --------------------------------------------------------- */
alt_irq_context ctx = alt_irq_disable_all();
alt_u8 appuis = g_appui_bouton;
g_appui_bouton = 0;
alt_irq_enable_all(ctx);

/* ---------------------------------------------------------
 * RÉGLAGE DE L'HEURE (SW3=0 pour activer)
 * --------------------------------------------------------- */
if (!sw3) {
    Heure_t *h_reg = afficher_alarme ? &heure_alarme : &heure_courante;

    if (sw4) {
        /* ------------------------------------------------
         * Mode appuis successifs (SW4=1)
         * Un incrément par appui physique, capturé ISR.
         * ------------------------------------------------ */
        if (appuis & 0x01u) { inc_minute(h_reg); h_reg->seconde = 0; }
        if (appuis & 0x02u) { inc_heure  (h_reg); h_reg->seconde = 0; }

    } else {
        /* ------------------------------------------------
         * Mode continu (SW4=0)


        /* Lire l'état physique des boutons (après inverseur) :
         *   bit = 1 → appuyé
         *   bit = 0 → relâché                                  */
        alt_u32 brut = IORD_ALTERA_AVALON_PIO_DATA(BOUTONS_POUSSOIRS_BASE) & 0x03u;

                        // --- GESTION DES MINUTES (KEY0) ---
        if ((brut & 0x01u)) { // SI le bouton est physiquement APPUYÉ (0)
            // On incrémente si c'est le 1er appui (edgecap) OU si le délai de 500ms est passé
            if ((appuis & 0x01u) || (maintenant >= t_next_m)) {
                inc_minute(h_reg);
                h_reg->seconde = 0;         // On fige les secondes pour le réglage
                t_next_m = maintenant + 500; // On programme le prochain saut dans 0.5s
            }
        } else {
            t_next_m = maintenant; // Reset du timer quand on relâche
        }

        // --- GESTION DES HEURES (KEY1) ---
        if ((brut & 0x02u)) { // SI le bouton est physiquement APPUYÉ (0)
            if ((appuis & 0x02u) || (maintenant >= t_next_h)) {
                inc_heure(h_reg);
                h_reg->seconde = 0;
                t_next_h = maintenant + 500;
            }
```

```
                } else {
                    t_next_h = maintenant;
                }
            }
        }

        /* ------------------------------------------------------
         * AFFICHAGE : heure courante ou alarme, 24h ou 12h
         * ------------------------------------------------------ */
        const Heure_t *source     = afficher_alarme ? &heure_alarme : &heure_courante;
        Heure_t        a_afficher = format_12h ? convertir_12h(source) : *source;
        afficher_heure(&a_afficher);

    } /* fin while(1) */

    return 0;
}
```