Systèmes numériques embarqués (MNE M1S2)

Rapport : *Mini-projet - conception et implantation d'un réveil électronique sur cible FPGA*

Écrit par : Justin Silver

Groupe B (TP)

Fichier électronique : rapport_SNE_M1S2_SILVER.pdf

Résumé

Pour le cours de Systèmes Numériques Embarqués (SNE) de la formation Micro et nano-électronique (MNE) à l'Université de Strasbourg en 2021, voici la présentation du travail de conception d'un réveil électronique réalisé sur un FPGA MAX 10 située sur une carte de développement DE10-Lite. Quartus II version 18.1 a été utilisé comme logiciel de développement et la partie soft (programmation du softcore Nios II) du projet a été conçu à travers l'IDE Eclipse. Le dispositif offrit des fonctionnalités « classiques » de d'autres réveils commerciaux (réglage de l'alarme, réglage de l'heure, affichage du temps, etc.) mais ce « alarm clock jukebox » en proposer quelques-uns de plus. Notamment, un parseur permettant de lire des mélodies adhérentes au format RTTTL de Nokia, une « piano LED » qui permet de visualiser les notes monophoniques sortant du buzzer, et un module PWM qui permet de jouer ces mélodies et de régler leur volume selon une échelle logarithmique. Tous les périphériques fonctionnent en mode interruption, et la modularité du système a été mis au point.

Table des matières

Table des figures

Table des tableaux

GitHub – suivi du développement

Pour les lecteurs intéressés, le développement du projet peut être investigué sur ma page GitHub : www.github.com/sir-drako/alarm-clock-jukebox.

Features

Les features qui ont pu été réalisés :

Feature	Description
Affichage de l'horloge	L'utilisateur voit le temps actuel de l'horloge s'incrémente au fil du temps
Réglage de l'horloge	L'utilisateur peut incrémenter/décrémenter l'heure/les minutes de l'horloge en appuyant sur les boutons poussoirs (les changements s'affichent en temps réel)
Affichage du temps réglé pour le réveil	L'utilisateur peut visualiser quand est-ce que l'alarme va sonner
Réglage du réveil	L'utilisateur peut incrémenter/décrémenter l'heure/les minutes du réveil en appuyant sur les boutons poussoirs (les changements s'affichent en temps réel)
Cascade ou non-cascade des secondes selon le mode du système	En mode configuration de l'horloge (heure/minute) les secondes ne cascadent pas aux minutes (par exemple, 00:11:59 restera à 00:11:00 au lieu de passer à 00:12:00)
Mélodies du réveil	Un module PWM travaille en conjonction avec des mélodies (en format RTTTL) pour pouvoir jouer des chansons lors que l'alarme sonne
Sélection de la mélodie du réveil et possibilité de « preview » des mélodies	L'utilisateur peut à run-time (hors de programmation) sélectionner la mélodie à jouer pendant que l'alarme sonne. En plus, l'utilisateur peut « preview » chaque mélodie (chargée au préalable dans la mémoire on-chip) pour en choisir une
Alarme armée via indicateur LED	L'alarme est armée quand la LED associée est allumée (**remarque : l'alarme ne peut être allumée qu'en mode « affichage de l'horloge »**)
Piano à LED qui permet de visualiser les notes de la mélodie et leurs durations	Avec 7 LEDs associées aux notes musiques A, B, C, D, E, F, et G respectivement, l'utilisateur peut visuellement apprécier l'évolution de la mélodie
Réglage du volume du réveil (de la mélodie)	Le rapport cyclique du module PWM peut être modifié. À l'utilisation d'un look-up table, un mapping logarithmique est employé avec un environ 2 dB incrémentation de puissance par pas de volume. Volume 0 correspond à un rapport cyclique de 0% rapport

	cyclique (minimum), et volume 12 correspond à un rapport cyclique de 50% (maximum)
Sélection du mode du système via des switches	L'utilisateur peut basculer entre les modes suivants selon l'état des switches : 1) Affichage de l'horloge 2) Réglage de l'horloge (heure) 3) Réglage de l'horloge (minute) 4) Affichage du réveil 5) Réglage du réveil (heure) 6) Réglage du réveil (minute) 7) Réglage du volume du réveil 8) Sélection de la mélodie du réveil 9) Armer le réveil La sélection d'un mode invalide ou non-défini par l'utilisateur ne modifie pas le mode actuel du système

Table 1. Features réalisés

À cause des contraintes du temps, les features suivants initialement prévus n'ont pas été implementés :

Feature	Description
Mode débogage et un basculement aisé de celui-ci	Pour debugger les différents modules du système en stade de développement, les « printf statements » ont été beaucoup employés. Or, ces appels de fonctions ont introduit une large « overhead » au processeur qui a notamment ralenti le déroulement des mélodies. Un simple « #ifdef/#endif » syntaxe de preprocessor pourrait être employé pour les appels printf au lieu de les commenter directement dans le code source
Plus de modularité dans le code source et de plus clairs APIs (fonctions d'interface au module main.c)	Alors que le code soit divisé dans les fichiers .c pour les différents périphériques et pour les fonctionnalités diverses du système, les modules exhibent toujours de la codépendance qui peut être réduite. De plus, certains des modules peut encore être divisés dans de « sub-modules ». Un exemple : la conversion décimal-BCD n'est pas indépendante du module « display.c »
Préemption des interruptions (des ISRs)	Le système dans son état actuel ne permet pas la préemption d'un ISR (interrupt service routine) par une autre interruption. Ce fait peut devenir inacceptable si une haute précision de l'heure actuelle est nécessaire. Si par exemple une mélodie RTTTL est en train d'être parsé dans son ISR associé alors que le timer second se déclenche son interruption, le système doit d'abord retourner de l'ISR « RTTTL » avant d'incrémenter les secondes de l'horloge. Un meilleur système serait de permettre la préemption des interruptions, éventuellement avec une espèce de RTOS (Real-time operating système) minimaliste.

Feedback visuel à l'utilisateur si un mode invalide est sélectionné	L'utilisateur n'a pas de feedback (visuel ou auditoire) au cas où il sélectionne un mode non-défini via des switches. Le clignotement d'une LED et/ou l'apparition d'un message défilant sur l'afficheur pourrait indiquer à l'utilisateur que le mode demandé n'est pas valide
Feedback visuel à l'utilisateur si le volume maximale/minimale est atteint	Un message défilant affichant « MAX VOLUME » sur l'afficheur peut indiquer à l'utilisateur que le volume ne peut pas être augmenté au-delà de ce qui est affiché
Remise manuelle des secondes à XX:XX:00 en mode configuration	Pendant que l'utilisateur configure le temps de l'horloge, s'il souhaite le régler en comparant avec une horloge extérieure, une remise à 0 des secondes lui permettra de mieux synchroniser les deux horloges. Celle-ci peut être configurée si par exemple les 2 boutons poussoirs sont appuyés en même temps par l'utilisateur

Table 2. Features non-réalisés

Pour faciliter les tests de l'appareil sur la carte DE10-Lite, l'interface suivant a été construite. Nous remarquons bien la sélection du mode à partir des switches, l'affichage, et l'incrémentation/la décrémentation à partir des boutons poussoirs.

Figure 1. Interface de l'utilisateur

Introduction et vue « haut-niveau » du système

Le système fonctionne en mode interruption, où les actions à prendre sont déterminées selon l'état du système (son mode actuel) et sont exécutées dans les ISRs. Alors que les différents modes d'utilisation sont sélectionnables par l'utilisateur, il en existe d'autres à l'intérieur du système qui gèrent l'évolution du programme. Pour garantir le bon déroulement du système, le mode « courant » est toujours mis à jour si besoin et sauvegardé dans une *struct* (espace de mémoire).

```
struct mode{
  uint8_t invalid;
  uint8_t display;
  uint8_t alarm;
  struct config config;
};

// values possible for the mode struct members

// mode.invalid = {FALSE, TRUE}
// mode.config.on = {FALSE, TRUE}
// mode.config.hour = {FALSE, TRUE}
// mode.config.minute = {FALSE, TRUE}
#define FALSE 0
#define TRUE 1

// mode.alarm = {OFF, ON}
#define OFF 0
#define ON 1

// mode.display = {DISP_CLOCK, DISP_ALARM, DISP_VOLUME, DISP_SONG}
#define DISP_CLOCK 0
#define DISP_ALARM 1
#define DISP_VOLUME 2
#define DISP_SONG 3
```

Figure 2. Modes variés du système

En sachant les différents « modes » du système, le flux du programme peut être décrit par le schéma suivant : ** *en tenant compte de larges dimensions du schéma, une haute résolution d'image n'est pas garantie. Pour une meilleure lisibilité, se référer directement au pdf du schéma inclus dans ce rapport et zoomer sur celui-ci* ***

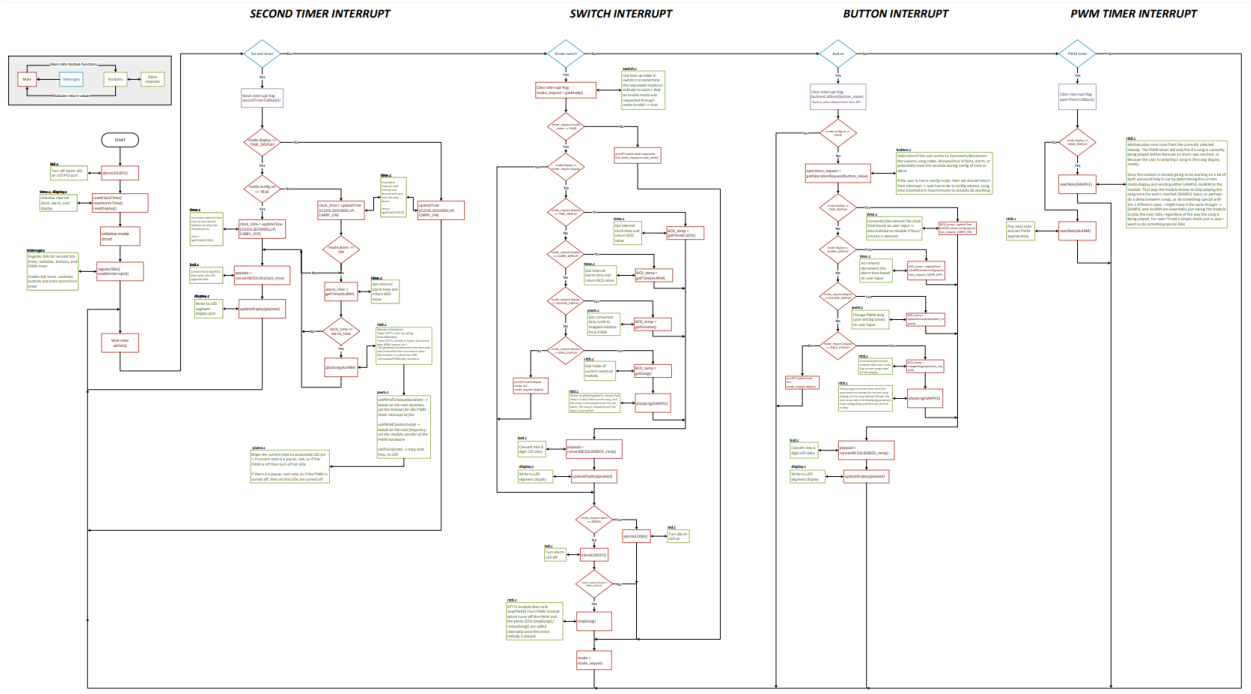

Figure 3. Flux schématique du système

En premier vue, ce schéma peut apparaitre assez complexe. Pour donner plus de clarté sur le fonctionnement du système, le schéma global peut être découpé en quelques morceaux et individuellement élaborés. Un bref résumé de chaque morceau (apparaissant dans la Figure 3 de gauche à droite) est donné par la table suivante :

Section du schéma	Description
Configuration initiale du système	Permettre le « reset » du système pour établir son état initial. Les interruptions sont enregistrées et mise en route, l'heure initiale est affichée, etc.
Interruption « second timer »	Mettre à jour l'horloge interne du module time_keeper.c. Selon le mode du système, l'affichage ne pourrait pas être mis à jour (par exemple, si l'utilisateur est en train de configurer le réveil, l'affichage n'est pas mis à jour). Ce module détermine aussi si le réveil devrait sonner
Interruption « switch »	Le changement de l'état des switches engendre une requête de « mode » par l'utilisateur. Le système détermine les actions nécessaires à prendre pour mettre à jour l'affichage, si par exemple l'utilisateur bascule du mode réglage du réveil à l'affichage de l'horloge.
Interruption « button »	Si le système est dans un mode valide, la détermination d'une opération « + » ou « -« est faite, pour ensuite mettre à jour le temps interne et/ou la

	sélection de mélodie/le rapport cyclique du PWM. Selon le mode, l'affichage est mis à jour.
Interruption « PWM timer »	Cette branche permet le bon déroulement d'une mélodie. La durée du timer est toujours mis pour être la durée d'une note, donc chaque fois que cette interruption a lieu, une nouvelle note est chargée du module rtttl.c, puis envoyée au module pwm.c pour changer le bruit venant du buzzer

Tableau 1. Explication des parties du schéma global

Le code du projet est séparé dans de différents « modules » qui se communiquent via le module principal : « main.c ». Un effort de découper les codépendances entre ces modules a conduit à un développement plus simplifié et à une testabilité plus aisée. Par exemple, le module time_keeper.c ne gère que l'état de l'horloge et du réveil. Il ne se concerne pas de l'affichage aux LEDs, de la conversion décimal-BCD, de la lecture des switches, etc. Pour pouvoir échanger l'état du temps à travers le système, ce module communique à travers main.c avec des fonctions « API ». Pour pouvoir obtenir l'heure actuelle de l'horloge pour son affichage (par exemple) main.c appelle : getClockTime() qui retourne la *struct* contenant les valeurs du temps.

Modules

La réalisation du projet a été faite étape par étape, module par module. Ci-dessous, chaque module (fichier .c) du système est individuellement investigué. Pour encore plus de détails sur leurs fonctionnements, le lecteur est invité à directement explorer le code source et les commentaires qui s'y trouvent.

Module main.c

« main.c » est le module principal du système. Après quelques instructions d'initialisation (enregistrement des ISRs, reset du display, etc.) le programme se met dans une boucle infinie : *while(1)*. À ce point, le module attend une des interruptions qui ont été préalablement enregistrées et « enabled ». Il gère la communication entre les différents modules et se comporte comme « arbitre » entre eux. Il est le seul module qui change explicitement le mode du système. En plus, pour que le système fonctionne comme prévu, il partage le mode actuel avec de différents modules.

Le module tout seul ne fait presque rien, vu qu'il n'implémente pas de vraies fonctionnalités. Les ISRs qui sont appelés sont situés dans ce module, mais dans ces fonctions l'importance est de déterminer les actions à prendre selon le mode actuel et ensuite d'appeler les fonctions « API » de différents modules.

Module display.c

Ce module est responsable pour la conversion décimal-BCD en utilisant un algorithme de division par 10. Il est également chargé d'écrire aux registres associés aux afficheurs 7 segments LED, après la conversion du résultat BCD. Voici un exemple de conversion qui se fait à partir des arrays et de la division successive:

```
/* Conversion decimal to BCD example (number 512):
dec = 512
bcd[2] = 512/100 = 5
bcd[1] = (512-5*100)/10 = 12/10 = 1
bcd[0] = (512 - (5*100 + 1*10))/1 = 2
*/
```

Figure 4. Conversion du décimal à BCD

Module altera_avalon_pwm_routines.c

Pour pouvoir facilement jouer des mélodies, un PWM (Pulse-width modulator) a été employé. Un design de référence diffusée par Altera et nommé *Design Example: Pulse-Width Modulator Slave* a été adopté [1]. La documentation pour ce design date et les instructions pour configurer le PWM se référaient à une vieille version du logiciel Quartus et du Platform Designer. Par conséquent, il fallait de la patience et de la reconfiguration pour pouvoir correctement adapter le code verilog du PWM et le rendre accessible comme « esclave du bus » à partir du Platform Designer.

Le code dans ce module est le « glue » entre le Hardware Description code (verilog) et l'interface software (Nios II) au PWM. Le lecteur peut se rendre à la documentation du PWM fournie par Altera pour plus de détails, mais un bref résumé sur le fonctionnement du système est donné ci-dessous (image prise de la documentation) :

Figure 6–3. PWM Task Logic Structure

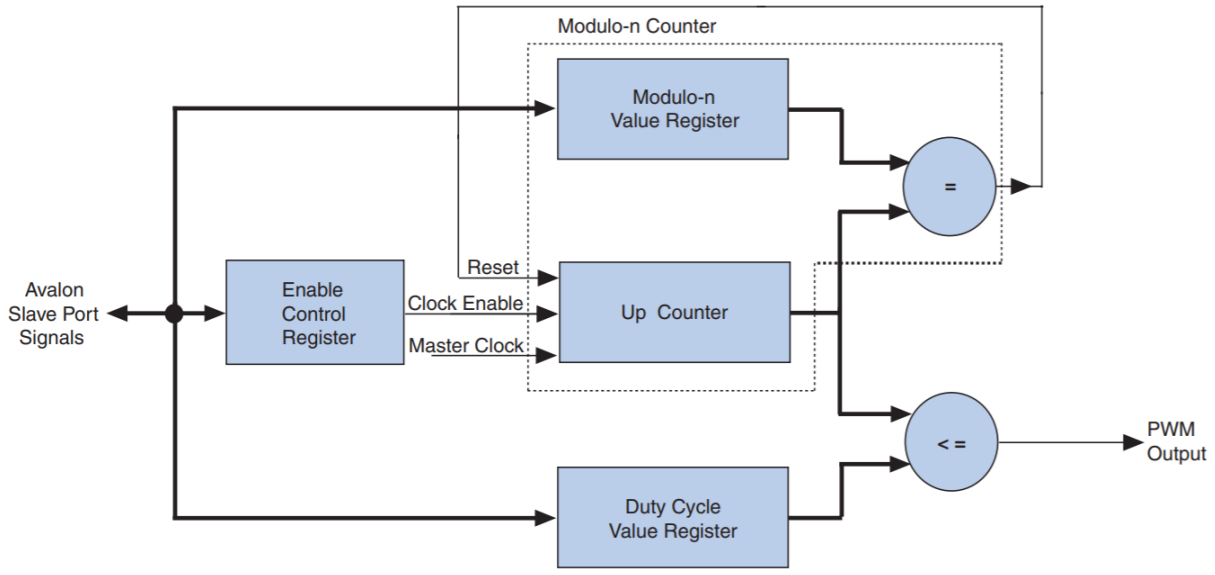

Figure 5. Bloc hardware du PWM Altera

Un bloc de contrôle permet d'éteindre ou d'allumer l'output du PWM. Ensuite, si le PWM est enabled, un compteur commence à incrémenter à chaque front d'horloge. La valeur du compteur est comparée à un registre nommé Modulo-n Value Register qui contient des données préalablement écrites par l'utilisateur et qui déterminera la fréquence du signal PWM.

Une fois l'opération égal passe à 1, le compteur est remis à zéro, et le cycle se répète. Nous remarquons également la possibilité de varier le rapport cyclique du PWM. En effet, la sortie du Duty Cycle Value Register est comparée (moins ou égal) au compte actuel du compteur. Si par exemple le registre du rapport cyclique contenant une valeur qui est la moitié du registre du modulo-n, le signal PWM serait carré avec un rapport cyclique de 50%.

Module pwm.c

Ce module peut être considéré comme un « wrapper » pour le module PWM fourni par Altera. Il est là pour consommer les APIs du PWM (écriture, reset, changement du rapport cyclique, etc.). C'est aussi dans ce module que l'information de la note (sa durée en millisecondes et sa fréquence en Hz) et le volume réglé par l'utilisateur est convertis aux valeurs à mettre dans le registre Modulo-N et dans le registre Duty Cycle du PWM Altera.

Le module est également censé de régler le volume (et ainsi le rapport cyclique) du PWM en faisant une conversion du volume demandé par l'utilisateur (une valeur entre 0 et 12) dans une valeur de rapport cyclique. La conversion se fait à l'aide d'un lookup table qui emploi une relation logarithmique entre les 2 grandeurs.

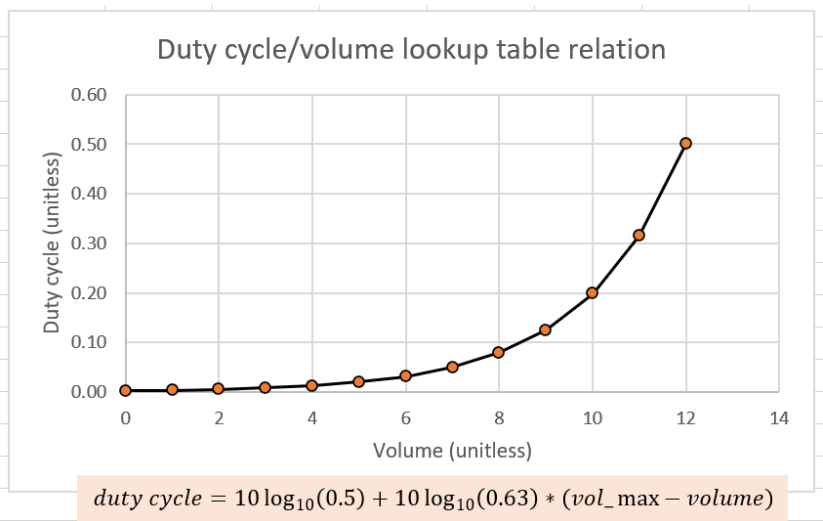

Figure 6. Rapport cyclique/volume relation

La formule exacte a été choisie pour offrir un 2 dB gain de puissance par pas de volume. La formule permet aussi d'obtenir un rapport cyclique maximum de 50% si le volume choisi par l'utilisateur est = au volume max permissible.

Module rtttl.c

La décision de directement parser les chansons RTTTL a été pour la facilité de lecture et de téléchargement sur la mémoire on-chip du FPGA. D'autres choix ont été brièvement investigués, par exemple parser les mélodies RTTTL à partir du langage JavaScript et puis exporter les données en format fréquence/durée.

En tenant compte que le système passe la majorité de son temps dans la boucle while(1) et par conséquent a suffisamment du temps disponible pour parser les données, l'algorithme RTTTL player du Michael Ringgaard a été adapté et reconfiguré pour cette application [2].

Module switches.c

Tout ce qui concerne la détermination du mode à partir de l'état des switches se passe dans ce module. L'architecture choisie pour l'appareil fait que plusieurs switches mis dans l'état ON engendrai une demande d'un mode invalide. C'est-à-dire que les modes demandés où seulement un switch est allumé (si ce mode est défini) ceux-ci sont valides. L'exception est quand le système est en mode affichage du temps de l'horloge. Dans ce cas, aucun switch n'est allumé, alors que le système est dans un état valide. Par conséquent, la présence d'un mode valide ou pas est déterminée en regardant si seulement un switch (ou zéros switches) sont ON. En mathématique binaire, cela se traduit en déterminant si le nombre binaire qui représente l'état des switches est une puissance de 2 ou de 0. Un exemple de ce calcul est donné ci-dessous :

```
/***** Determining whether or not a single switch is active (valid mode) ****

if the switch state is a power of 2 or 0, then
it is a valid mode. Otherwise more than one switch is activated, which
means that the mode requested is invalid....
--------------------------------------------
example with 3 bits: 000

valid modes (power of 2 or 0)       : 0, 1, 2, 4
invalid modes (not power of 2 or 0) : 3, 5, 6, 7

0 & -1  = 0 -> valid!
1 & 0   = 0 -> valid!
2 & 1   = 0 -> valid!
4 & 3   = 0 -> valid!

3 & 2  != 0 -> invalid!
5 & 4  != 0 -> invalid!
6 & 5  != 0 -> invalid!
7 & 6  != 0 -> invalid!
--------------------------------------------
*/

uint8_t isPowerOfTwoOrZero(uint16_t value) {
  if ((value & (value-1)) == 0) {
    return TRUE;
  }
  else {
    return FALSE;
  }
}
```

Figure 7. Algorithme pour déterminer un mode valide

Module time_keeper.c

Comme son nom indique, ce module gère tout ce qui concerne les valeurs de temps pour l'horloge et pour le réveil. C'est le seul module où ces valeurs sont directement modifiées (incrémentées ou décrémentés). La décision de séparer toutes les opérations dans de petites fonctions au lieu de regrouper des opérations similaires a été explicitement faite. Un exemple : une fonction « upClockMinute() » existe ainsi qu'une fonction « upAlarmMinute() » malgré le fait que ces deux font exactement la même chose sur 2 *struct* similaires. Même si le code dans ce module est notamment répétitif, les fonctions sont très explicites et testables. De plus, le temps de l'horloge et le temps du réveil peuvent être indépendamment changés sans trop de difficulté.

Module interrupt.c

Ce module gère l'enregistrement des interruptions (l'enregistrement des ISRs passés du main.c) et aussi les fonctions qui les activent. Un cas spécial est réservé pour le timerPWM, qui est parfois désactivé selon l'état du système. Ainsi, une fonction de « disable » pour cette source d'interruption est incluse. L'enregistrement est configuré en suivant les étapes de la documentation fournie par Altera [3]. Grâce au HAL fourni par Altera, la configuration se résume dans un simple appel de fonction « alt_ic_isr_register » en passant des paramètres associés à la source d'interruption (ID de l'interruption, pointeur à la fonction ISR, etc.)

Module led.c

Ce module gère l'état de la LED associée à l'alarme et des LEDs « piano ». En tenant compte du nombre de LEDs disponible sur la carte DE10-Lite, il n'y en avait pas assez par rapport à la gamme de notes dans les mélodies RTTTL. Le piano correspond ainsi aux notes A, B, C, D, E, F, et G (7 LED rouges sur la carte).

L'état du piano dépend de la note parsée en amont par le module rtttl.c. Comme dans les autres cas où plusieurs modules doivent se « communiquer » entre eux, c'est le rôle du module principal (main.c) qui partage l'information entre ces deux modules.

Difficultés rencontrées

- *Modularité et configuration du logiciel*

En tenant de la complexité du système et après avoir pris la décision de construire de petits modules, j'avais une difficulté initiale à configurer le software (Nios II Build Tools) pour supporter plusieurs fichiers en-têtes (pwm.h, rtttl.h, etc.). C'était une aventure amusante pour trouver le bon menu de configuration.

- *Interruption – timer*

La première interruption que j'ai essayée de mettre en place a été pour le timer d'une seconde. Son enregistrement (via le HAL Altera) n'a pas posé trop de problèmes. La vraie difficulté a été de le mettre en route et de gérer l'état de ses flags associés pour qu'il démarre un nouveau cycle de comptage. Une fois le type de timer a été identifié, et après avoir regardé la documentation Altera [4], j'ai pu configurer le timer et obtenir des interruptions périodiques.

- *Configuration du PWM (Altera) dans Platform Designer*

Bien qu'Altera ait fourni une documentation détaillant les étapes nécessaires pour utiliser son design PWM, ces étapes de configuration pour la partie logicielle (Nios II) s'appuyaient sur une très vieille version du Quartus et de Platform Designer (Qsys). Par conséquent, il fallait expérimenter et farfouiller dans la nouvelle interface du Platform Designer pour pouvoir correctement configurer le périphérique comme un « avalon slave ».

- *D'autres bugs…*

D'une manière générale, j'ai employé 2 techniques pour debugger le design. Principalement, j'ai employé l'utilisation des appels « printf » aux moments clés du programme. Par exemple, après les échecs d'enregistrement des ISRs, après que les ISRs sont appelés par le HAL Altera, quand le mode d'affichage change, etc.

Ma deuxième façon de résoudre des problèmes techniques employait le debugger pas-à-pas utilisable dans l'environnement Eclipse. Ceci se prouvait extrêmement utile lors du « parsing » des mélodies RTTTL.

Conclusion

Voici ce qui conclut le projet de réveil électronique sur cible FPGA, avec un softcore Nios II processeur et du logique hard sur la fabrique FPGA (module PWM par exemple). Comme indiqué dans la section Features, il y a toujours beaucoup plus de choses à raffiner, à modifier et à ajouter. Même avec certains inconvénients du système actuel (impossibilité de régler le volume en même temps que le réveil sonne) l'architecture choisie est très flexible pour ajouter plus de fonctionnalités par la suite et pour éventuellement implémenter un RTOS simplifié pour introduire la préemption des interruptions et pour mieux gérer la cohérence des tâches (parser des chansons RTTTLs, mettre à jour l'affichage, pouvoir mettre à jour le temps interne, etc.).

Références

[1] Altera, «Design Example: Pulse Width Modulator Slave,» [En ligne]. Available: http://www.ee.nmt.edu/~erives/554_10/Altera_PWM.pdf.

[2] M. Ringgaard, «Ringing Tones Text Transfer Language (RTTTL) C player,» [En ligne]. Available: http://www.jbox.dk/sanos/source/lib/rtttl.c.html.

[3] Altera, «Exception Handling,» [En ligne]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/nios2/n2sw_nii52006.pdf.

[4] Altera, «Interval Timer Core,» [En ligne]. Available: https://www.intel.co.jp/content/dam/altera-www/global/ja_JP/pdfs/literature/hb/nios2/n2cpu_nii51008.pdf.