Session S4 GEGI

APP1_COMBI

Annexe du guide étudiant
Logique combinatoire

Département de génie électrique et de génie informatique

Faculté de génie

Université de Sherbrooke

Hiver 2024

Note : En vue d'alléger le texte, le masculin est utilisé pour désigner les femmes et les hommes.

Document S4_APP_Annexe_H2024.docx
Rédigé par Gérard Lachiver, ing., décembre 2001
Révisé par Réjean Fontaine, ing., novembre 2004
Révisé par Frédéric Mailhot, janvier 2006
Révisé par Réjean Fontaine, ing., janvier 2010
Révisé par Réjean Fontaine, Daniel Dalle, janvier 2014
Révisé par Réjean Fontaine, Daniel Dalle, 15 janvier 2015
Révisé par Réjean Fontaine, Amer Al-Canaan, 21 décembre 2015
Révisé par Réjean Fontaine, Daniel Dalle, décembre 2018
Révisé par Réjean Fontaine, Marc-André Tétrault, décembre 2020
Révisé par Réjean Fontaine, Marc-André Tétrault, décembre 2021
Révisé par Réjean Fontaine, Marc-André Tétrault, décembre 2022,
Distinction entre liste de tests et plan de vérification

Tables des matières

Contenu

Table des illustrations et tables

1. Architecture générale de la section numérique du projet de détecteur de proximité et spécifications haut niveau des modules

1.1. Les cartes électroniques

La résolution de la problématique de l'APP1 nécessite 4 cartes électroniques : la carte Zybo-Z7 de la compagnie Digilent, la carte *ADC thermométrique*, un PmodSSD (afficheur 7 segments) et un Pmod8LD (8 DELs) (Figure 1.1 et Figure 1.2). D'une part, la carte Zybo effectue les opérations logiques dans une matrice de portes programmables (FPGA) et d'autre part, agit comme interface d'entrée/sortie grâce à ses boutons, interrupteurs et diodes électroluminescentes (DEL). La carte *ADC thermométrique* effectue la conversion d'un signal analogique (0 - 3,3 V) sur l'entrée V_{in} en un signal numérique encodé sous forme thermométrique sur 12 niveaux (si on exclut le niveau 0 où rien n'est présent). Cette carte contient également 2 DELs, 2 interrupteurs, 1 potentiomètre ainsi que des circuits auxiliaires assurant une alimentation adéquate aux différents circuits électroniques. Dans le cadre de la problématique de l'APP, un capteur capacitif pourra être branché à la carte *ADC thermométrique* ; un cavalier doit être positionné alors entre les pattes JP1-2 et JP1-3. Même sans capteur, il vous est possible de vérifier le comportement de votre électronique grâce à un potentiomètre sur la carte *ADC thermométrique*. À ce moment, un cavalier doit être positionné entre les pattes JP1-1 et JP1-2. La rotation de potentiomètre permet de générer manuellement une tension entre 0 et 3,3 V. Les potentiomètres sont un peu oxidés. Il est possible que leur fonctionnement soit erratique. Nous y sommes habitués et il n'y aura pas d'enjeux de points perdus à cause de ce composant. Finalement, il est possible d'utiliser un générateur de fonctions si un cavalier est positionné entre les pattes JP1-2 et JP1-3 et que l'on fait entrer le signal du générateur entre J1-1 et J1-3 (où J1-1 est la masse) ou encore sur TP1 et TP2 (voir schéma électrique de la carte *ADC thermométrique* disponible sur le site WEB) et ainsi imiter le comportement du capteur. Si vous faites cela, il faut faire attention à ce que le signal provenant du générateur de fonction soit entre 0 et 3,3 V.

Dans le cadre plus spécifique de cet APP, il vous sera demandé de réaliser les simulations complètes de ce circuit. Vous pourrez simuler le comportement des interfaces dans Vivado. Il ne restera que la partie de l'implémentation sur la carte FPGA à réaliser. Une importance plus grande sera apportée aux tests de fonctionnalité des circuits cette session. En industrie, ces tests occuperont plus de la moitié de votre temps. Il est important de bien maitriser les notions vis à vis les différents types de tests développés tout au long de la session. Un document est disponible sur le site de la session pour lancer votre réflexion sur la différence subtile entre la vérification et les tests. Pour cet APP, nous amorçons cette réflexion à très bas niveau, c'est-à-dire avec une simple liste de tests.

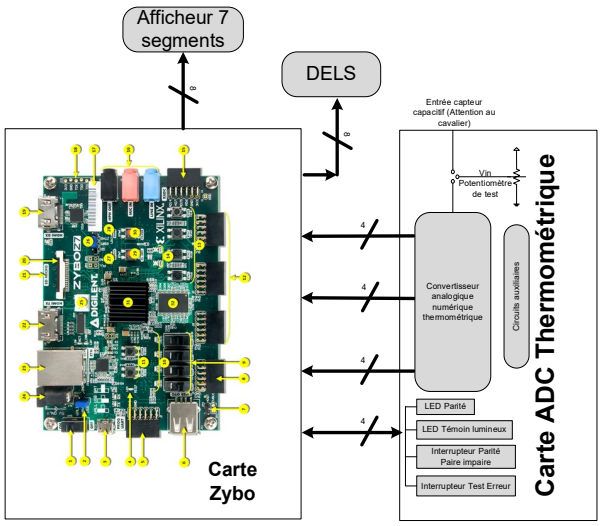

Figure 1.1 – Branchement simplifié des cartes électroniques

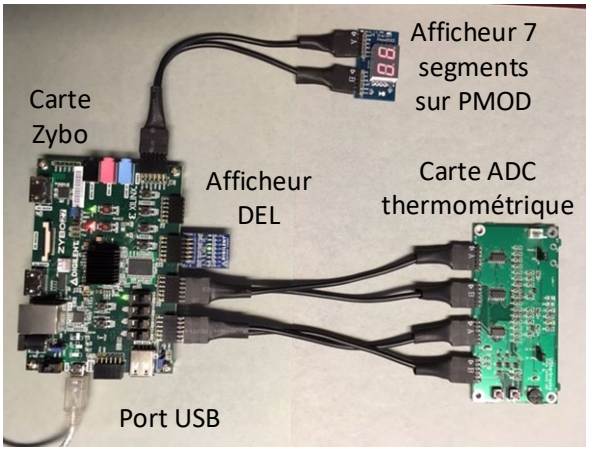

Figure 1.2 – Photographie du branchement à réaliser

Les deux cartes principales contiennent plusieurs boutons poussoirs qui serviront à interagir avec différentes fonctions résumées ci-après.

Sur la carte Zybo, on retrouve 4 boutons poussoirs nommés BTN0 à BTN3 et sur la carte *ADC Thermométrique*, deux boutons poussoirs nommés S1 et S2. La liste suivante résume les fonctions attachées aux boutons poussoir du système :

- BTN0 et BTN1 non pressés : affiche la valeur BCD non signée sur l'afficheur à 7 segments ;
- BTN0 pressé : affiche la valeur hexadécimale sur l'afficheur à 7 segments ;
- BTN1 pressé : affiche la valeur BCD soustraite de 5 sur l'afficheur à 7 segments ;
- BTN0 et BTN1 pressés simultanément : affiche le code « Er » sur l'afficheur à 7 segments ;
- BTN2, BTN3 : non utilisés ;
- S1 (pressé) : parité paire sur *LD0* de la carte Zybo et sur la DEL2 de la carte ADC thermométrique ;
- S1 (non pressé) : parité impaire sur *LD0* de la carte Zybo et sur la DEL2 de la carte ADC thermométrique ;
- S2 : (pressé) : afficher le code « Er » sur l'afficheur à 7 segments.

Les explications détaillées pour ces fonctions seront précisées dans les paragraphes qui suivent.

1.2. L'encodage thermométrique

L'information peut être encodée numériquement de façons plus ou moins efficaces. Vous n'avez qu'à penser aux algorithmes de compression des images (.jpg, .gif, .tiff, .ps) ou encore à la compression des fichiers (.zip, .tar.gz). Dans notre cas, il ne s'agit pas d'une compression d'images ou de fichiers mais d'une façon de représenter l'information analogique. Plusieurs d'entre vous auront entendu parler des circuits convertisseurs analogiques numériques (CAN) (*Analog to digital converters* (ADC) en anglais). Ces convertisseurs encodent généralement l'information de façon binaire, 0_{10} (00_{16}) correspondant à une tension analogique minimale et 255_{10} (FF_{16}) à une tension analogique maximale dans le cas d'un convertisseur de 8 bits. L'encodage s'effectue de façon linéaire entre 0 et 255 et toutes les valeurs binaires sont possibles. Dans le cadre de la problématique de l'APP, l'encodage est réalisé par une série de comparateurs (TLV2374) (schéma de la carte *ADC thermométrique* disponible sur le site WEB) qui comparent la tension d'entrée V_{in} avec la tension obtenue à travers une série de résistances (R1 à R13) qui créent elles-mêmes un diviseur résistif multiniveau. La série de résistance divise la tension de référence de 3,3 V en 12 niveaux plus ou moins égaux[1] de ~253 mV. La sortie d'un comparateur monte à 3,3 V (ou *1* logique) lorsque la tension V_{in} devient supérieure à la tension issue du diviseur résistif, sinon, elle est de ~0 V (ou *0* logique). Les niveaux de tensions et de courants de la sortie du comparateur sont compatibles avec l'entrée du FPGA et seront interprétés comme un *0* ou un *1* logique selon qu'elles soient ~0 V ou ~3,3 V.
Dans le cas de notre convertisseur thermométrique, l'augmentation de la tension V_{in} amène successivement à 3,3 V les sorties des comparateurs à partir de $ADC_{th}0$ jusqu'à $ADC_{th}11$ à la manière d'un thermomètre au mercure dont le niveau monte en fonction de la température ; d'où la terminologie *thermométrique*. Par exemple, les tensions V_{in} suivantes généreront les codes thermométriques suivants :

Tension V_{in}	$ADC_{th}11. \ . \ .ADC_{th}0$
0 mV - 253 mV	-> 0000 0000 0000_2,
253 mV - 507 mV	-> 0000 0000 0001_2,
507 mV - 760 mV	-> 0000 0000 0011_2.
760 mV - 1013 mV	-> 0000 0000 0111_2,
. . .	
3047 mV - 3300 mV	-> **1111 1111 1111**$_2$.

Comme vous le constatez, au fur et à mesure que la tension V_{in} augmente, un *1* s'insère vers la gauche à partir du bit le moins significatif (LSB).

Malgré que ce convertisseur soit facile à réaliser matériellement, il ne représente pas l'encodage d'information optimal car il contient beaucoup de codes impossibles. Par exemple, le code 0000 00**1**0 00**11**$_2$ est un cas qui ne peut jamais arriver en pratique à moins d'un problème électronique au niveau d'un des comparateurs. Cet encodage *troué* est peu efficace par rapport à la quantité d'électronique requise pour la traiter et l'on préfère généralement le CAN linéaire. Vous utiliserez un CAN linéaire de 8 bits à l'APP2 portant sur la logique séquentielle et comprendrez alors davantage les limitations du convertisseur thermométrique. Cependant, ce convertisseur comporte un grand intérêt scientifique car plusieurs détecteurs ont un tel comportement, mais aussi académique, pour la compréhension des circuits logiques, des comparateurs et des diviseurs résistifs.

[1] On dit plus ou moins car les résistances ont une certaine imprécision (S1-APP1)

1.3. Les modules à réaliser dans le FPGA

Le projet nécessite la conception de plusieurs sous-modules, à l'image d'un programme séquentiel (C/C++, Python, Java) qui divise ses processus en sous-fonctions. Une exception importante doit être bien comprise quant à l'utilisation des variables en VHDL. Ces dernières exigent une compréhension poussée du VHDL et leur utilisation est proscrite pour la synthèse pendant les APP de la session afin de vous éviter des ennuis subtiles et difficiles à déverminer. Elles sont cependant acceptées dans les simulations. D'autre part, plutôt qu'un appel à une fonction « *main* », l'intégration des modules se fait dans un bloc VHDL de niveau cime (*top level*). Tout comme un projet de programmation, la réalisation d'un projet de circuits numériques demande de comprendre les objectifs de haut niveau (voir l'énoncé de la problématique) et un découpage du travail, rôle et/ou les actions des sous-modules sur les signaux de contrôle et les données. La Figure 1.3 est une reformulation graphique du texte de la problématique et présente le découpage des sous-modules en vue de le coder en VHDL. À gauche, on retrouve en entrée les boutons S0, S1, BTN0 et BTN2 ainsi que l'encodeur thermométrique à binaire non-signé. À droite, on voit les trois groupes de sorties (série de 8 DEL, DEL de parité et affichage 7-segments). Similairement, on aperçoit 3 principaux trajets ou chemins de données (traits pointillés en bleu, mauve et rouge) partant du code thermométrique et terminant à une des sorties, avec deux trajets modulés par des boutons de contrôle. Finalement, le bouton S2 permet d'afficher « Er » sur les affichages.

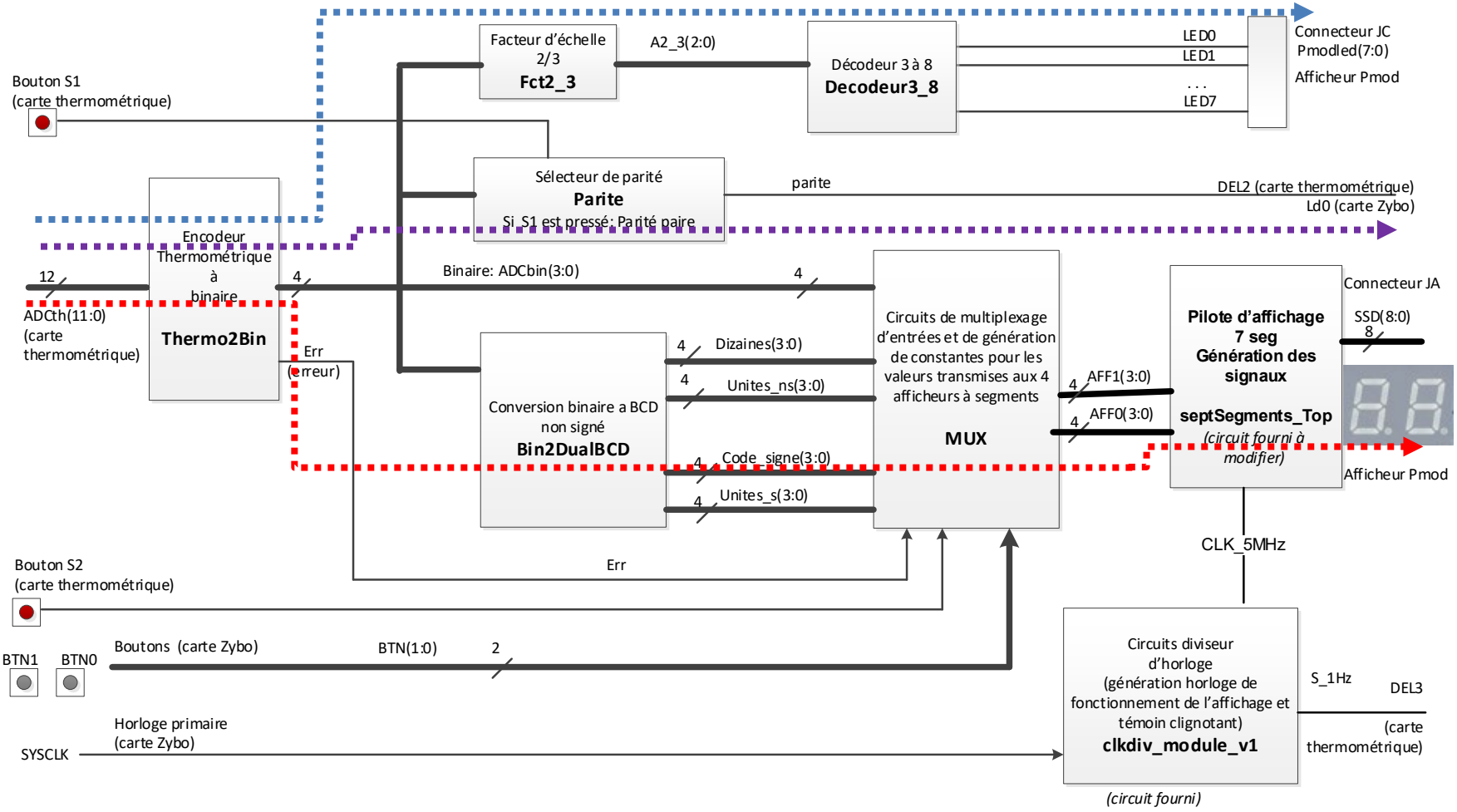

Figure 1.3 – Architecture proposée dans le FPGA

La DEL D3 sert d'indicateur de fonctionnement. Elle clignote lorsque le FPGA est bien programmé. Ne pas la simuler.

La rédaction des spécifications, décrites dans les prochaines sections de ce document, est organisée de « gauche à droite » et de « haut en bas » tel que décrit dans la Figure 1.3. Vos développements VHDL suivront également cette méthodologie à l'exception du module thermo2bin qui devra être réalisé en dernier étant donné sa complexité par rapport aux autres modules. Assurez-vous d'avoir rédigé et vérifié vos modules avant de procéder à l'intégration afin de minimiser le temps passé au déverminage.

1.3.1. Thermo2Bin

Spécification : Le module *Thermo2Bin* transforme les 12 bits ADC_{th} (11 : 0) encodés thermométriquement en une valeur binaire sur 4 bits non signée *ADCbin*(3 :0). Cette dernière alimente 4 sous-modules. À moins d'une erreur de fabrication du circuit imprimé ou des comparateurs, il est théoriquement impossible[2] que la sortie d'un comparateur soit *1* si son voisin LSB ne l'est pas déjà. Nous allons tout de même détecter les codes erronés et générer une erreur dans un tel cas (*erreur = 1*). Le module *Thermo2Bin* devrait être réalisée en dernier car elle comporte des difficultés au niveau du nombre de bits en entrée (12 bits) et par le fait qu'il est plus facile à déboguer quand on est capable d'afficher des données et de les interpréter (*hint* de conception).

Entrées : ADC_{th} (11 :0), $ADC_{th\,i} \in \{0, 1\}$, où i = 0, .. à 11

Sorties : *ADCbin*(3 :0), *ADCbin*(i) $\in \{0, 1\}$ où i = 0, .. à 3

$$erreur = \begin{Bmatrix} 1 \text{ si erreur dans le code thermométrique} \\ 0 \text{ si aucune erreur dans le code thermométrique} \end{Bmatrix}$$

Type de réalisation : Ce module doit être réalisée par une description VHDL comportant des équations logiques simplifiées obtenues à partir de tableaux de Karnaugh provenant d'une démarche complète de conception, par exemple "*a <= B XOR C;*".

Note de conception : Ce module est relativement complexe et doit être fait en dernier. Dans ce module, il est **exigé** de découper le module en un schéma bloc similaire à la Figure 1.3. L'objectif est de vous habituer à découper un module complexe en plus petits modules plus facile à traiter. Idéalement, cette astuce vous amènera à réutiliser plusieurs fois le(s) même(s) bloc(s), offrant une économie de vos efforts. On recommande de vous occuper du traitement des données en premier et ensuite de vous occuper de l'erreur. Pour découper le travail en plus petits morceaux, il faut vous poser la question à savoir quelles sont les entrées et sorties de chaque sous-bloc (combien de bits et quelle représentation des nombres : signé, non signé, C2 etc…) et quelle est la fonction de ce sous-bloc. Quand les entrées et sorties sont complétés dans une table de vérité réduite mais complète, vous pouvez alors simplifier avec Karnaugh et établir vos équations. Vous pouvez réutiliser des blocs déjà faits dans d'autres modules pour effectuer le travail si cela est justifié. Nous vous laissons décider comment faire.

NOTE IMPORTANTE : Il pourrait sembler stratégique de créer une table de vérité avec seulement les cas thermométriques pertinentes, mais en réalité cette table est incomplète et ne vous permettra pas d'exécuter la

[2] Ceci signifie qu'il devrait être possible d'utiliser les indifférents pour simplifier davantage les équations.

procédure de conception complète et rigoureuse que vous tentez d'apprendre. Ce faux raccourci vient au contraire vous nuire, et mènera inéluctablement vers des mauvaises notes dans la validation et le rapport, c'est-à-dire « compétence non atteinte » ! Le découpage avec les entrées et sortie doit être maîtrisé pour le projet de session. Il est impératif de le faire intégralement au moins une fois. Ce genre de découpage sera votre travail d'ingénieur plus tard.

Liste des tests Test 1 : permuter tous les cas admissibles du code thermométrique et comparer le résultat avec la valeur prévue.
Test 2 : valider que tous les autres cas génèrent un "1" sur le signal d'erreur.

Comme ce module inclut plusieurs sous-modules, il faut faire attention à créer une liste de tests qui ne tient pas compte de la méthode d'implémentation de vos sous-modules. Habituellement, les tests sont rédigés/codés par une tierce personne qui ne sait pas comment votre module a été fait. Il faut voir *Thermo2bin* comme une boite noire.

*Pour le 2e test, réfléchissez à un moyen efficace qui ne nécessite pas la permutation de presque 2^{12} possibilités (il y a au moins 2 bonnes réponses !).

1.3.2. Fct2_3

Spécification : On désire afficher l'amplitude du signal V_{in} sur les 8 DEL (*LD7* .. *LD0*) sur le PMOD8LD. Une seule DEL sera allumée à la fois et correspondra à l'amplitude de V_{in}. Le meilleur circuit pour n'allumer qu'une seule DEL est le *décodeur3_8* décrit à la section 1.3.3. On doit lui fournir une valeur sur 3 bits *A2_3*(2 :0) correspondant à la DEL à allumer. Comme il y a 12 entrées et seulement 8 DEL, il faut effectuer une contraction de l'information. C'est le rôle du module *Fct2_3* qui multiplie la valeur de *ADCbin*(3 :0) par 2/3 (0,66). En circuit logique, on peut prendre une approche calquée sur les processus mathématiques et réaliser un module général de multiplication et de division ; ceci est très fastidieux. On peut aussi simplement créer une table de vérité et associer les valeurs de sortie désirées. Dans notre cas, nous allons utiliser une méthode alternative qui nécessite la compréhension de la notation positionnelle décimale. Comme la multiplication se fait avec un opérande de valeur fixe (facteur de 0,666), on peut décomposer cette opération par une série d'additions. Pour ce faire, il faut comprendre qu'une multiplication/division par 2 en base 2 consiste à décaler vers la gauche/droite les bits que nous avons entre les mains. Prenons un exemple le chiffre 3_{10} en base 10 :

$$3_{10} = 0011_2$$

Si l'on multiplie par 2 -> $2 \times 3_{10} = 6_{10} = 0110_2$. On voit que les bits sont décalés vers la gauche et qu'un zéro a été inséré à droite en base 2. Il en est de même pour une division par 2 en binaire où l'on décale vers la droite et insère un zéro à gauche.

La question est de savoir maintenant comment faire une multiplication par un nombre qui n'est pas une puissance de 2^n. Prenons exemple avec une multiplication d'un nombre *A* par 0,9. Nous ne pouvons multiplier directement par 0,9 mais nous pouvons trouver une combinaison linéaire de puissance de 2^n qui pourrait s'en approcher. Par exemple, 0,9 ~ 0,5 + 0,25 + 0,125 + 0,015625 + 0,0078125 + …

En termes de puissance de 2^n, nous retrouvons $0{,}9 \sim 2^{-1} + 2^{-2} + 2^{-3} + 2^{-6} + 2^{-7} + \ldots$ Cette somme est tronquée car nous n'arrivons pas à la valeur exacte de 0,9 mais nous en avons tout de même une bonne approximation (i.e. 0,989). À partir de cela, il nous est possible de faire une multiplication par une somme d'addition en effectuant les opérations suivantes.

$$N \times 0{,}9 \sim N \times (2^{-1} + 2^{-2} + 2^{-3} + 2^{-6} + 2^{-7})$$

En distribuant le *A*, on retrouve

$$N \times 2^{-1} + N \times 2^{-2} + N \times 2^{-3} + N \times 2^{-6} + N \times 2^{-7}$$

Cette opération se trouve être une suite d'addition du nombre *N* décalé par une puissance de 2^{-n}. La façon de réaliser cette opération est de brancher une version décalée de *N* sur l'opérande A de l'additionneur et une seconde version décalée du nombre *N* sur l'opérande B. Vous remarquerez immédiatement que l'opération comporte certains problèmes au niveau des résultats car nous avons une approximation entre les mains et non un calcul exact. Les problèmes sont d'autant plus marqués si le nombre disponible de bits est petit ; ce qui est normal car il y a moins de plage dynamique pour représenter les nombres. Il existe des moyens pour maximiser la plage dynamique, par exemple, le C_{in} d'un additionneur 4 bit peut être utilisé pour élargir la plage dynamique et reproduire le comportement d'une addition à la position 2^{-1}.

Entrées : *ADCbin*(3 :0), *ADCbin*(i) ∈ {0, 1}

Sorties : *A2_3*(2 :0), *A2_3*(i) ∈ {0, 1}

Type de réalisation : Ce module doit être réalisée en utilisant l'additionneur 4 bit développé dans le laboratoire de l'APP.

Liste des tests Test 1 : Valider les 16 possibilités.

1.3.3. Décodeur 3_8

Spécification : Un décodeur 3 à 8 permet de sélectionner une seule des 8 sorties en fonction de l'adresse de 3 bits présentée en entrée.

Entrées : *A2_3*(2 :0), *A2_3*(i) ∈ {0, 1}

Sorties : *LED*(7 :0), *LED*(i) ∈ {0, 1}

Type de réalisation : Ce module doit être réalisée par une structure ***autre*** qu'un « CASE » dans un process, par exemple avec une structure « WHEN ».
Note : le bit le moins significatif est à *LED*0 et le plus significatif est à *LED*7.

Liste des tests Test 1 : Valider les 8 possibilités.

1.3.4. Parité

Tout échange de données entre un module électronique peut être entaché par des erreurs de transmission selon la qualité du canal de communication. La qualité du canal se mesure à travers le *bit error rate* (BER). Selon notre tolérance au taux d'erreur, le coût de transmission des données peut être plus on moins important. Par exemple, si on envoie des échantillons de voix et qu'il y a un seul bit erroné, la personne qui écoutera la conversation ne le remarquera probablement pas. Dans d'autres cas, c'est plus critique et des protocoles avec contrôle des erreurs ont été développés. C'est le cas des transmissions de paquets Ethernet de type TCP/IP qui renverront un paquet au complet si une erreur de transmission est détectée. Cependant, comment détecter que des informations comportent une ou des erreurs, voire corriger cette/ces erreur(s) lors de la transmission ? Plusieurs mécanismes existent pour faire ce travail. Parmi les méthodes plus utilisées, on retrouve la somme de contrôle, mieux connue en anglais par le terme *checksum*. Dans ce cas, le transmetteur calcule la somme de tous les octets (bytes) du paquet sur 8, 16, ou 32 bits et l'ajoute à la fin du paquet. Le récepteur fait également le même travail et vérifie si la somme correspond à la somme envoyée. Si la somme est identique, l'intégrité du paquet est considérée valide et son utilisation est

maintenue. On peut appliquer le même genre de processus à des octets individuels. Par exemple, on ajoute alors un bit de *parité* à l'octet indiquant si la somme de ses bits individuels constituants est paire ou impaire. Cette somme se calcule en comptant littéralement le nombre de 1 par exemple, si on a le nombre $1001\ 0110_2$, on a quatre 1, donc un nombre pair de 1. Ainsi, si 1 bit de l'octet reçu est erroné, le décompte des bits sera différent du bit de parité calculé et l'erreur sera détectée. Cette approche est cependant limitée à un seul bit car si 2 bits changent alors l'erreur ne sera pas détectée. Malheureusement, la définition de parité paire ou impaire change selon les domaines. Pour notre cas, nous opterons pour qu'une parité paire corresponde à un décompte pair de 1 incluant le bit de parité. Identiquement, une parité impaire correspond à un décompte impair de 1 incluant le bit de parité. Les tables suivantes présentent le cheminement pour produire un train de bits, appelé aussi symbole, pour des nombres de 3 bits auxquels est ajouté le bit de parité paire et impaire.

Tableau 1. Calcul de la parité paire

Nombre décimal	Nombre binaire et hexadécimal	Bit de parité ajouté si parité **paire**	Symbole envoyé sur la ligne de transmission, composé du bit de parité ajouté (souligné) et de la charge utile (nombre de la 2e colonne).
0	000 (0_{16})	0	0000 (0_{16})
1	001 (1_{16})	1	1001 (9_{16})
2	010 (2_{16})	1	1010 (A_{16})
3	011 (3_{16})	0	0011 (3_{16})
4	100 (4_{16})	1	1100 (C_{16})
5	101 (5_{16})	0	0101 (5_{16})
6	110 (6_{16})	0	0110 (6_{16})
7	111 (7_{16})	1	1111 (F_{16})

Tableau 2. Calcul de la parité impaire

Nombre décimal	Nombre binaire et hexadécimal	Bit de parité ajouté si parité **impaire**	Symbole envoyé sur la ligne de transmission, composé du bit de parité ajouté (souligné) et de la charge utile (nombre de la 2e colonne).
0	000 (0_{16})	1	1000 (8_{16})
1	001 (1_{16})	0	0001 (1_{16})
2	010 (2_{16})	0	0010 (2_{16})
3	011 (3_{16})	1	1011 (B_{16})
4	100 (4_{16})	0	0100 (4_{16})
5	101 (5_{16})	1	1101 (D_{16})
6	110 (6_{16})	1	1110 (E_{16})
7	111 (7_{16})	0	0111 (7_{16})

En résumé, si on veut transmettre le nombre décimal 2 avec un bit de parité paire, on voit qu'il y a un seul 1 dans le nombre. Pour avoir une parité paire, on doit avoir un nombre pair de 1 dans le symbole et on ajoutera un 1 en MSB et on transmettra le symbole A_{16}. Pour compléter cet exemple, nous allons, cette fois-ci, nous placer du côté

du récepteur. Supposons que le protocole de transmission repose sur une parité paire et que le symbole F_{16} est reçu. Le récepteur compte le nombre de 1 dans le symbole, soit 4, et indique qu'il n'y a pas d'erreur de transmission. Le récepteur transmettra le nombre 7 aux modules subséquents. Pour vous en assurer, revenez à la table et faites l'exercice inverse. Par contre, si le récepteur reçoit le symbole 8_{16}, il va trouver un seul 1 dans le symbole et déclarer une erreur de transmission. Le système de réception décidera comment traiter cette erreur. Tel qu'indiqué précédemment, l'ajout du bit de parité comporte des limitations. En guise de dernier exemple, supposons que l'on veuille transmettre le nombre 1 (soit 001_2) avec une parité paire, nous allons donc transmettre le symbole 9_{16} i.e. 1001_2. Supposons également qu'il y ait 2 bits erronés dans la transmission soit le MSB et le LSB. Les bits reçus vont passer de 1001_2 à 0000_2. Comme le récepteur sait que le protocole prévoit une parité paire, il regarde le nombre de 1 dans 0000_2 et indique qu'il n'y a pas d'erreur et déterminera qu'il a reçu le nombre 0, ce qui est faux. Il existe d'autres types de codage de l'information qui permettent de s'affranchir de ce problème. Vous les verrez plus tard. Si vous désirez comprendre comment l'information se transmet avec le bit de parité, nous vous invitons à consulter le site https://ipc2u.com/articles/knowledge-base/the-main-differences-between-rs-232-rs-422-and-rs-485/ à la section « Structure of the transmitted data in RS-232 ». Vous y verrez que les bits sont mis en série incluant ou non un bit de parité.

Spécification : Ce module a pour objectif de calculer le bit de parité à ajouter au nombre *ADCbin*(3 :0) et de l'afficher sur la DEL2 (*D*2) de la carte *ADC thermométrique*. L'interrupteur *S1* de la carte *ADC thermométrique* indiquera si la parité paire (*S1* pressé) ou impaire (*S1* non pressé) doit être affichée. Si le bit de parité calculé vaut 1, la DEL *D*2 doit être allumée. Le mot allumé est souligné car ce n'est pas parce qu'un 1 logique sort du FPGA que la DEL sera allumée. Il faut regarder si la DEL est en *pull-up* ou en *pull down* et il faut tenir compte de la chaine d'inverseurs entre le FPGA et la DEL pour l'allumer correctement. Cela signifie que même si notre calcul est correct, il est possible que nous devions inverser le signal pour des raisons matérielles. À ce moment, on réalise le calcul correctement et on ajoute un inverseur dans le code VHDL avec un commentaire qui explique cette inversion. Vous devez, de toute façon, vous intéresser à cette chaine d'inverseurs pour des raisons de compatibilité des signaux. Vérifiez en même temps la polarité pour faire allumer ou non la DEL. Notez que la DEL *D*1 est absente de la carte *ADC thermométrique*. N'oubliez pas de bien positionner le cavalier JP2 en position JP2-1 et JP2-2 pour que ce soit le signal du FPGA qui fasse allumer/éteindre la DEL *D*2. Le cavalier JP2 permet de vérifier le fonctionnement de la DEL *D*2. Si la DEL D2 ne fonctionne pas, vérifiez la compatibilité des signaux entre le FPGA et la DEL D2.

Pour vous rassurer dans vos calculs, vous allez également envoyer l'information de la parité à la DEL *LD0* de la carte Zybo.

Entrées : *ADCbin*(3 :0), *ADCbin*(i) ∈ {0, 1}
S1, *S1* ∈ {0, 1}
Sorties : *Parite*, *Parite* ∈ {0, 1}

Type de réalisation : Ce module doit être réalisée en VHDL à l'aide d'une structure CASE.

Liste des tests Test 1 : Valider les 16 possibilités avec une parité paire.
Test 2 : Valider les 16 possibilités avec une parité impaire.

Note. La vérification sur DEL2 et sur LD0 permettra de déterminer si la valeur désirée sort du FPGA. Cela ne signifie pas que la DEL s'allumera si un problème de compatibilité est présent. Il faut séparer les 2 cas.

1.3.5. Bin2DualBCD

Spécification : Ce module a pour objectif de transformer l'entrée *ADCbin*(3 :0) non signée en deux représentations BCD différentes. Une première paire de sorties (Dizaines/Unites_ns) donnent une valeur BCD non signée, alors que la deuxième paire donne une valeur BCD de *ADCbin*(3 :0) soustraite de 5 (Unite_s) et un code pour le bit de signe (*Code_signe*). Avant de réaliser ces modules, vous devez vous poser les questions sur les plages dynamiques des chiffres à afficher et vous pencher sur les modifications à apporter au module bin2_7seg afin de pouvoir générer les bons codes.

Dans la réalité, ce module se compose de 3 sous-modules ; Bin2DualBCD_NS, Moins_5 et Bin2DualBCD_S selon le schéma suivant :

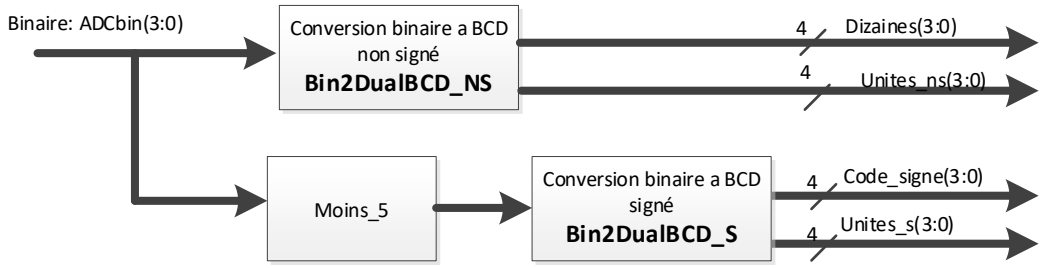

Figure 1.4 Description des sous-modules de Bin2DualBCD

Entrées : *ADCbin*(3 :0), *ADCbin*(i) ∈ {0, 1}

Sorties : *Dizaines*(3 :0), *Dizaines*(i) ∈ {0, 1}
Unites_ns(3 :0), *Unites_ns*(i) ∈ {0, 1}
Code_signe(3 :0), *Code_signe*(i) ∈ {0, 1}
Unite_s(3 :0), *Unite_s*(i) ∈ {0, 1}

Type de réalisation : Ce module doit être réalisée par différentes approches de description VHDL selon les sous-modules décrits ci-bas.

Liste des tests Test 1 : Valider les 16 possibilités.

1.3.5.1. Bin2DualBCD_NS

Objectif : Ce sous-module a pour objectif de transformer l'entrée *ADCbin*(3 :0) non signée en une représentation BCD pour permettre l'affichage des dizaines et des unités.

Entrées : *ADCbin*(3 :0), *ADCbin*(i) ∈ {0, 1}

Sorties : *Dizaines*(3 :0), *Dizaines*(i) ∈ {0, 1}
Unites_ns(3 :0), *Unites_ns*(i) ∈ {0, 1}

Type de réalisation : Ce sous-module doit être réalisée en VHDL à l'aide d'une structure CASE.

Liste des tests Test 1 : Valider les 16 possibilités.

1.3.5.2. Moins_5

Objectif : Ce sous-module a pour objectif de soustraire 5 à tous les nombres à afficher. Le résultat de la soustraction doit être un nombre signé sur 4 bits. La soustraction doit être effectuée en complément à 2.

Entrées : *ADCbin*(3 :0), *ADCbin*(i) ∈ {0, 1}

Sorties : *Moins5*(3 :0), *Moins5*(i) ∈ {0, 1}

Type de réalisation : Ce sous-module doit être réalisée par l'additionneur 4 bits avec propagation de la retenue réalisé dans le laboratoire de l'APP.

Note : déterminer le nombre nécessaire de bits pour effectuer la soustraction.

Liste des tests Test 1 : Valider les 16 possibilités.

1.3.5.3. Bin2DualBCD_S

Objectif : Ce sous-module est très similaire au module *Bin2DualBCD_NS* à l'exception qu'il faut tenir compte du signe négatif qui peut apparaître dans certains cas. Avant de réaliser cette fonction, vous devez vous pencher sur les modifications à apporter au module bin2_7seg afin de pouvoir générer les bons codes sur les afficheurs 7 segments.

Entrées : *Moins5*(3 :0), *Moin5*(i) ∈ {0, 1}

Sorties : *Code_signe*(3 :0), Code_signe(i) ∈ {0, 1}
Unite_s(3 :0), *Unite_s*(i) ∈ {0, 1}

Type de réalisation : Ce sous-module doit être réalisée en VHDL en utilisant une structure CASE.

Liste des tests Test 1 : Valider les 16 possibilités.

Note : Remarquez qu'il n'y a plus de dizaines à afficher et que le signe *moins* doit être exprimé sur plusieurs bits. Cela semble invraisemblable car + ou – s'encode facilement sur 1 bit…. Il faut bien comprendre comment encoder l'information vers le module suivant pour un affichage correct du symbole.

1.3.6. Mux

Spécification : Ce module est relativement complexe de prime abord quand on ne comprend pas bien le fonctionnement des multiplexeurs. Examinez bien leur fonctionnement avant de vous lancer dans leur conception. Selon la valeur de l'entrée *erreur*, des *BTN*(1 :0) ou de l'interrupteur *S2* de la carte *ADC thermométrique*, le multiplexeur dirigera les bonnes entrées de 4 bits au module *affhexPmodSSD* qui veillera à leur affichage sur deux afficheurs 7 segments *DAFF(1 :0), DAFF*0 étant le LSB, localisés sur un module PmodSSD.

Entrées : *ADCbin*(3 :0), *ADCbin*(i) ∈ {0, 1}, note, ceci est équivalent à un affichage en hexadécimal
Dizaines(3 :0), *Dizaines*(i) ∈ {0, 1}
Unites_ns(3 :0), *Unites_ns*(i) ∈ {0, 1}
Code_signe(3 :0), *Code_signe*(i) ∈ {0, 1}

Unite_s(3 :0), *Unite_s*(i) ∈ {0, 1}
erreur, *erreur* ∈ {0, 1}
BTN(1 :0), *BTN*(i) ∈ {0, 1}
S2, S2 ∈ {0, 1}

Sorties : *DAFF0*(3 :0), DAFF0(i) ∈ {0, 1} LSB
DAFF1(3 :0), DAFF1(i) ∈ {0, 1}

Les informations présentées en début d'annexe sont recopiées et présentent les informations à afficher en fonction des différentes entrées.

- BTN0 et BTN1 non pressés : affiche la valeur BCD non signée sur l'afficheur à 7 segments ;
- BTN0 pressé : affiche la valeur en hexadécimal sur l'afficheur à 7 segments ;
- BTN1 pressé : affiche la valeur BCD soustraite de 5 sur l'afficheur à 7 segments ;
- BTN0 et BTN1 pressés simultanément : affiche le code « Er » sur l'afficheur à 7 segments ;
- BTN2, BTN3 : non utilisés ;
- S1 (pressé) : parité paire sur *LD0* de la carte Zybo et sur la DEL2 de la carte ADC thermométrique ;
- S1 (non pressé) : parité impaire sur *LD0* de la carte Zybo et sur la DEL2 de la carte ADC thermométrique ;
- S2 : (pressé) : afficher le code « Er » sur l'afficheur à 7 segments.

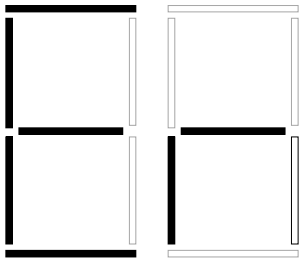

Figure 1.5 – Afficheurs à 7 segments en cas d'erreur

Type de réalisation : Ce module doit être réalisée par une fonction décrite en VHDL utilisant des IF..THEN et CASE.

Liste des tests Test 1 : Quand BTN(1 :0) = « 00 », les entrées *Dizaines*/*Unite_ns* sont recopiées à la sortie.
Test 2 : Quand BTN(1 :0) = « 01 », l'entrée *ADCbin* est recopiée à la sortie (MSB = 0).
Test 3 : Quand BTN(1 :0) = « 10 », les entrées *Code_signe* / *Unite_s* sont copiées à la sortie.
Test 4 : Quand BTN(1 :0) = « 11 », le code d'erreur est présenté à la sortie.
Test 5 : Quand S2 = '1', le code d'erreur est présenté à la sortie.
*Les tests 1,2 et 3 doivent tester les conditions avec au moins 2 valeurs différentes

1.3.7. septSegments_Top

Spécification : Ce module est déjà réalisé et un de ses sous-modules doit être modifié. La mise en œuvre vous est procurée en VHDL et contient 2 sous-modules :

1) *septSegments_encodeur* : Contient une table qui transforme une entrée binaire à une valeur *segm*(6 :0) déterminant les segments pour la formation d'un chiffre/caractère. Vous devez modifier cette table pour y ajouter des caractères manquants et nécessaires à la problématique.

2) *septSegments_refreshPmod* : (ne pas modifier) Ce module comporte une machine à états finis (processus *mef*). La MEF se compose d'un compteur à 1 bit cyclant par les états 0, 1, -> 0, …. Elle permet de lire séquentiellement les entrées *DA(3 :0)* et *DA(7 :4)* pour alimenter consécutivement a) les anodes communes de chaque chiffre et b) la bonne combinaison des segments (cathodes) des DEL de l'afficheur selon les données présentes à $Jpmod_i$ (Figure 1.5). Même si les caractères sont allumés séquentiellement sur les deux afficheurs à 7 segments, la rapidité du cycle trompe l'œil et le chiffre semble constant.

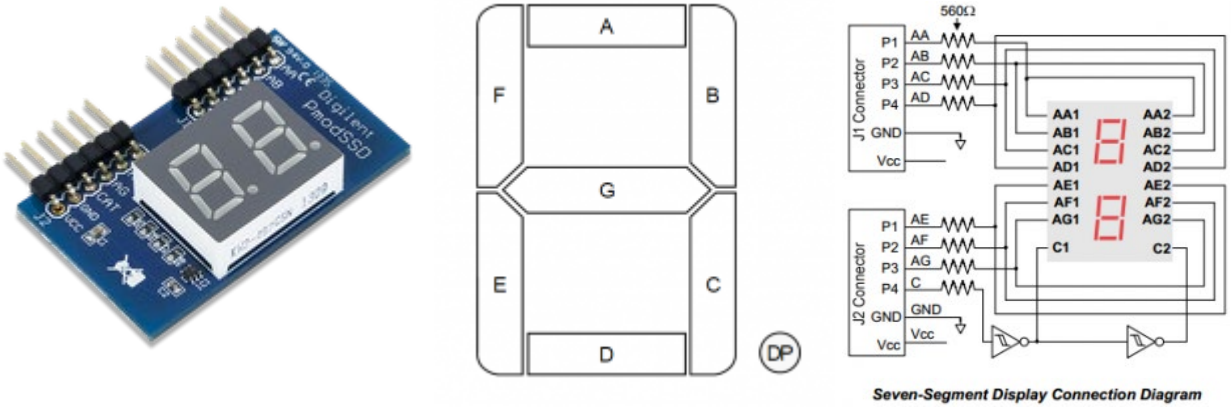

Figure 1.6 – Signaux de l'afficheur à 7 segments (selon réf. PmodSSD. www.digilentinc.com)

Les machines à états finis seront étudiées à l'APP2 portant sur la logique séquentielle et ne font pas l'objet de l'APP1. Cependant, la table de transformation binaire à 7 segments est combinatoire et fait partie des connaissances matières à examen.

Entrées et sorties : La déclaration des entrées-sorties du module septSegments_Top est définie ainsi en VHDL :

```
entity septSegments_Top is
generic (const_CLK_MHz: integer := 100);      -- horloge en MHz, typique 100 MHz
    Port ( clk     : in   STD_LOGIC; -- horloge systeme, typique 100 MHz (preciser par le constante)
           i_AFF0  : in   STD_LOGIC_VECTOR (3 downto 0); -- donnee a afficher sur 4 bits : chiffre hexa position 0
           i_AFF1  : in   STD_LOGIC_VECTOR (3 downto 0); -- donnee a afficher sur 4 bits : chiffre hexa position 1
           o_AFFSSD_Sim   : out string(1 to 2);
           o_AFFSSD       : out  STD_LOGIC_VECTOR (7 downto 0) -- sorties directement adaptees au connecteur PmodSSD
         );
end septSegments_Top;
```

La table de correspondance des segments pour un chiffre est définie dans le module septSegments_encodeur du code *VHDL*. Il est possible de modifier la table pour le besoin, notamment pour afficher les symboles *moins* et *r* requis pour résoudre la problématique. Parcourez la table d'affichage et vous trouverez certainement une solution.

```
Segment:
    process (donn)
```

```
    begin
      case donn is
--                                  "gfedcba"
              when "0000" => segm  <= "1000000"; -- 0
              when "0001" => segm  <= "1111001"; -- 1
              when "0010" => segm  <= "0100100"; -- 2
              when "0011" => segm  <= "0110000"; -- 3
              when "0100" => segm  <= "0011001"; -- 4
              when "0101" => segm  <= "0010010"; -- 5
              when "0110" => segm  <= "0000010"; -- 6
              when "0111" => segm  <= "1111000"; -- 7
              when "1000" => segm  <= "0000000"; -- 8
              when "1001" => segm  <= "0010000"; -- 9
              when "1010" => segm  <= "0001000"; -- A
              when "1011" => segm  <= "0000011"; -- b
              when "1100" => segm  <= "1000110"; -- C
              when "1101" => segm  <= "0100001"; -- d
              when "1110" => segm  <= "0000110"; -- E
              when "1111" => segm  <= "0001110"; -- F
              when others => segm  <= "1111111";
       end case;
    end process;
```

Liste des tests Test 1 : Permuter et valider tous les caractères hexadécimaux utiles
Test 2 : Permuter et valider tous les caractères BCD utiles
Test 3 : Afficher le caractère « r »
Test 4 : Afficher le signe négatif

- **Note : Ne pas simuler ce module car la simulation est trop longue à faire. Ce circuit implique une machine à états finis qui est vue à l'APP 2. Vous réaliserez ces tests une fois que le code sera dans le FPGA.**

1.3.8. Diviseur d'horloge : synchro_module_v2

Spécification : Ce module est déjà réalisée et n'est pas à modifier. À partir de l'horloge maîtresse de la carte (125 MHz), elle génère un signal d'horloge divisé à 5 MHz (*o_clk_5MHz)* et un second de 1 Hz *(o_S_1Hz)*. Le signal d'horloge de 5 MHz convient au circuit de l'afficheur à 7 segments décrit précédemment et l'horloge de 1 Hz sert de témoin lumineux pour indiquer que notre code est fonctionnel et que la programmation dans le FPGA s'est bien déroulée. L'horloge 1 Hz est branché sur la DEL *D*3 sur la carte *ADC thermométrique*. Cette pratique permet de distinguer entre une erreur de conception et une erreur de programmation/compilation. C'est extrêmement utile ; vous le constaterez.

Voici la déclaration des entrées-sorties du module :

```
entity synchro_module_v2 is
generic (const_CLK_syst_MHz: integer := 100);
    Port (
           clkm        : in STD_LOGIC;      -- Entrée  horloge maitresse
           o_clk_5MHz  : out  STD_LOGIC;    -- horloge divisee via bufg
           o_S_1Hz     : out  STD_LOGIC     -- Signal temoin 1 Hz (0,99952 Hz)
           );
end synchro_module_v2;
```

Note : vous n'avez pas à modifier cette section au niveau du FPGA. Il faut juste vous assurer que la DEL *D*3 clignote tel qu'attendu lors de la programmation du FPGA.

Liste des tests Pas de tests pour ce module. Il est entièrement validé.

1.3.9. Module App_combi_top (top)

Spécification : Le module à la cime gère l'interconnexion de tous les sous-modules et implémente l'intégration de ceux-ci (Figure 1.3.). Ainsi la fonction *Thermo2bin* transforme les 12 bits[3] $ADC_{th}(11:0)$ du convertisseur thermométrique en une représentation binaire non signée sur 4 bits nommée *ADCbin*(3 :0). Ces données alimentent 4 sous-systèmes : *Fct2_3*, *Parite*, *Mux* et *Bin2DualBCD*. L'objectif des modules précédents est de respectivement permettre l'affichage de l'amplitude de la tension mesurée sous diverses formes comme un bit mouvant sur une série de DEL, le calcul de la parité (paire et impaire) de la tension numérisée, le choix du type d'affichage de la tension numérisée et la génération des codes BCD et BCD soustraite de 5. La pression des boutons (*BTN*0 et *BTN*1) sélectionnera le type d'affichage du niveau thermométrique ou un code d'erreur. Le bouton S2 affichera également le code d'erreur. Enfin, les signaux d'entrée ADC_{th} n'existe pas dans le gabarit de projet fourni. Il faut les ajouter et leur assigner les bonnes pattes physiques du FPGA.

Entrées : *S1, S1* ∈ {0, 1}
S2, S2 ∈ {0, 1}
BTN0, BTN0 ∈ {0, 1}
BTN1, BTN1 ∈ {0, 1}
ADCth(11 :0), $ADCth_i \in \{0, 1\}$, où i = 0, .. à 11

Sorties : DEL2, DEL2 ∈ {0, 1}
DEL3, DEL3 ∈ {0, 1}
SSD(8 :0), SSD(i) ∈ {0, 1} où i = 0..7
LED(7:0), LED(i) ∈ {0, 1} où i = 0..7

Liste des tests Si les tests unitaires sont complets, les tests d'intégration n'ont pas nécessairement à couvrir tous les cas de figure possibles et imaginables. Il faut, par contre, maintenir un certain niveau de redondance, qui renforce l'ensemble du processus de tests.

Test 1 : Correspondance correcte pour toutes les entrées thermométriques valides en mode BCD non signé.
Test 2 : Correspondance correcte pour toutes les entrées thermométriques valides en mode hexadécimal.
Test 3 : Correspondance correcte pour toutes les entrées thermométriques valides en mode BCD signé avec soustraction de 5.
Test 4 : Déplacement linéaire et correcte sur la série de 8 DEL pour toutes les entrées thermométriques valides.
Test 5 : La sortie du FPGA[4] permet d'allumer la DEL2 pour les valeurs impaires lorsque la parité est en mode impaire, et de l'éteindre pour les valeurs paires
Test 6 : La sortie du FPGA permet d'allumer la DEL2 pour les valeurs paires lorsque la parité est en mode paire, et de l'éteindre pour les valeurs impaires
Test 7 : L'affichage montre « Er » quand BTN(1:0)="11"

---

[3] Notez que nous avons 12 bits dans lesquels une série de *1* peut se déplacer des LSB vers les MSB. Notre encodage contient également la valeur « 0000 0000 0000 » qui doit être considérée comme possible. Cela fait que nous avons 12 bits mais 13 niveaux dans le code thermométrique.

[4] On pointe clairement ici que ce doit être la « sortie du FPGA » qui permet d'allumer une DEL et non la vérification de la « DEL allumée » car dans ce dernier cas des problèmes sur l'électronique périphérique au FPGA pourrait faire en sorte que la DEL n'allume pas même si la sortie du FPGA est à la bonne tension. Ceci est un problème différent.

Test 8 : L'affichage montre « Er » quand SW='1'
Test 9 : L'affichage montre « Er » pour tous les codes thermométriques invalides.

1.4. Gestion et génération de la liste de tests

La spécification ci-haut suggère des tests à effectuer pour valider vos modules ainsi que l'intégration à haut niveau. Vous allez remarquer qu'il y a des recoupements entre les tests, tant au niveau du stimuli d'entrée que des résultats escomptés. Ceci apparaît plus clairement lorsqu'on synthétise l'ensemble des tests dans une liste des tests unitaires et d'intégration (Table I), qui aide à planifier, distribuer ou optimiser le travail.

Table 1 – Liste des tests unitaires et d'intégration

Modules		Tests	Codé	Réussite/échec
APP_combi_top (intégration)				
	1	Sortie BCD non signée avec tous les codes themo valides		
	2	Sortie Hexadécimal avec tous les codes themo valides		
	3	Sortie "moins 5" BCD signé avec tous les codes themo valides		
	4	DEL(7:0) uniforme avec tous les codes themo valides		
	5	Led Parité Paire (2 valeurs thermo)		
	6	Led Parité Impaire (2 valeurs thermo)		
	7	Afficher Er avec boutons à "11"		
	8	Afficher Er avec S2 pressé		
	9	Afficher Er pour tous les codes thermo invalides		
Thermo2Bin				
		Séquence de tous les niveaux validesÀ vous de définir et à remettre à la validation		
		Séquence démontrant tous les cas invalides		
Fct2_3				
		Table de vérité (16 cas)		
Décodeur 3-8				
		Table de vérité (8 cas)		
Parité				
		Led Parité paire (16 valeurs hexa)		
		Led Parité impaire (16 valeurs hexa)		
Bin2DualBCD				
		Table de vérité (16 cas)		
Bouton2Bin et Mux				
		Au moins 2 valeurs pour chaque cas		
		"00" -> copie Dizaines et Unites_ns à la sortie		
		"01" --> copie ADCbin à la sortie		
		"10" --> copie Code_sigen/Unite_s à la sortie		

	"11" --> code d'erreur pour affhexPmodSSD S2 --> code d'erreur pour affhexPmodSSD		
affhexPmodSSDseptSegment_top			
	Afficher les caractères hexadécimaux utiles Afficher les caractères BCD utiles Affichage de la lettre "r" Affichage du signe négatif		
Synchro_v2			
	Pas de tests		

Banc de test : Les tests pour vos modules de circuits numériques sont réalisés dans un *banc de test*, un segment de code VHDL qui se comporte beaucoup plus comme les fonctions de code séquentiel avec lesquels vous êtes familiers. Deux exemples de bancs de tests vous sont fournis à l'APP :

- Un banc de test générique avec des exemples combinatoires (page de l'APP, section de l'atelier Xilinx), propice aux tests unitaires.
- Un banc de test adapté au module cime, inclut dans le projet du laboratoire.

Pour tester vos modules, nous vous suggérons de créer un banc de test par module. Vous pouvez donc sécuriser votre banc de test et votre module une fois qu'il est validé, et vous assurer ainsi de ne pas modifier l'un ou l'autre par accident. Il est possible de mettre côte à côte tous les modules dans un seul banc de test, mais ceci risque de devenir surpeuplé de code et plus difficile à naviguer.

Optionnel - automatisation des tests : Pour vérifier vos modules, l'inspection visuelle de vos chronogrammes de simulation est certainement la plus facile au niveau de la logistique et simplicité du code. Malgré sa saveur répétitive lors de déverminage, cette méthode n'est pas trop lourde pour un projet de la taille de l'APP. Cependant, il est entièrement possible d'automatiser l'analyse des résultats des tests, où le banc de test applique un stimulus d'entrée, attend quelques nanosecondes, puis compare la sortie du circuit avec un résultat attendu prédéfini (par exemple, la sortie d'une table de vérité). Ces tests automatisés ne sont pas demandés à l'APP, mais si vous souhaitez les tenter, consultez le banc de test du laboratoire pour un exemple (lignes 120 à 130 et ligne 164 en commentaire). L'exemple montre comment créer un signal qui sonde virtuellement un signal, et une *assertion* qui génère un message en cas de divergence. Notez que Vivado doit compiler tout le code VHDL en utilisant la version 2008 du VHDL, qui pourrait ne pas être appliqué correctement si vous ajoutez vos propres fichiers VHDL (Figure 1.6). Les bancs de tests de très haut niveau font mieux : ils génèrent des valeurs aléatoires, calculent le résultat anticipé, font la comparaison et génèrent un rapport sur l'ensemble des tests (GEI815, en S8).

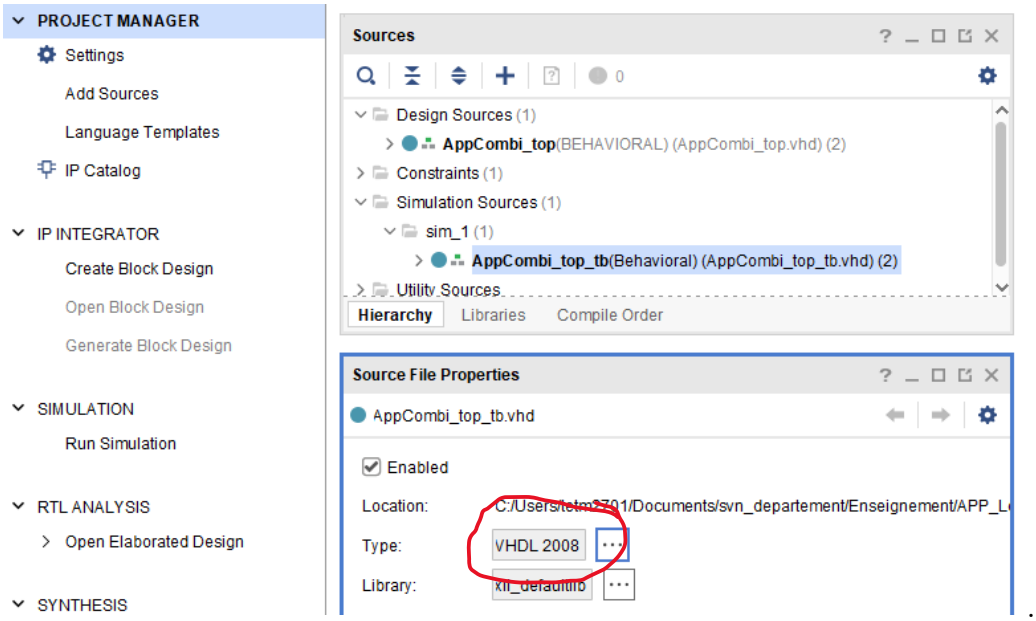

Figure 1.7 – Case à vérifier pour le VHDL 2008.

2. Quelques notions d'électronique

2.1. Les buffers 3-états

Les notions électroniques seront approfondies dans d'autres APP de la S4. Cependant, quelques notions doivent être abordées pour comprendre le comportement de certains circuits. Jusqu'à présent vous avez vu que les circuits logiques se caractérisent par 2 états : un niveau logique *1* ou élevé représenté par une tension (~5 V, ~3,3 V, ~1,8 V…) selon la technologie utilisée et un niveau logique bas ou *0* (~0 V). Pour bien comprendre les buffers 3 états, il est essentiel de comprendre le lien entre les tensions et les niveaux logiques et pour cela, il faut descendre au niveau des transistors. Nous nous limiterons à la technologie *Complementary Metal Oxyde Semiconductor* (CMOS) qui est plus simple d'approche à cet effet. Nous supposerons que la technologie opère à une tension d'alimentation de 3,3 V. Dans la technologie CMOS, nous retrouvons 2 types de transistors : les NMOS et les PMOS (figure 2.1a). La particularité des transistors NMOS est qu'ils conduisent si la tension à leur entrée dépasse un certain seuil (~ 0,7 V). Ainsi, si une tension de 3,3 V est appliquée à I2, le transistor NMOS de la figure 2.1a créera un chemin de très faible résistance entre la sortie Q et Vss i.e. R2 ≈ 0 Ohm (fig. 2.1b). Une tension avoisinant ~0 V apparaît alors à la sortie Q si toutes les autres résistances à ce nœud sont très grandes, ce qui est habituellement le cas[5]. D'un autre côté, la résistance R2 sera de plusieurs centaines de MOhms si le transistor n'est pas en conduction (i.e. I2 ≈ 0 V). Le transistor PMOS agit de façon inverse au NMOS et devient une faible impédance si une tension inférieure à ~(Vdd – 0,7 V) est appliquée à sa grille. Ainsi, si une tension ~ 0 V est appliquée à ce transistor alors la sortie Q sera reliée à Vdd par une faible résistance et ~ 3,3 V apparaitront à Q si les autres résistances reliées à ce nœud sont élevées. Si une tension ~ 3,3 V est appliquée sur I1, alors le PMOS présentera une résistance de plusieurs centaines de MOhms. Les tailles des transistors PMOS et NMOS sont habituellement ajustées pour que les résistances en conduction soient similaires. Les grilles de ces transistors (i.e. I1 et I2) sont en réalité des capacités qui présentent une haute impédance jusqu'à des fréquences d'opération relativement élevées. En technologie CMOS, nous allons faire en sorte qu'un seul des transistors (NMOS ou PMOS) soit en conduction à la fois. Il faut absolument éviter que les 2 transistors conduisent en même temps (I1 à *0* et I2 à *1*) car cela créerait un court-circuit entre Vdd et Vss pouvant amener à griller la porte logique étant donné l'amplitude du courant qui y passerait. Un exemple typique de fonctionnement d'un circuit logique est l'inverseur situé à la figure 2.1c. Supposons que l'entrée A soit à ~3,3V, le NMOS conduira mais pas le PMOS amenant ainsi une tension ~ 0 V à Q. À l'inverse, si l'entrée A est ~0 V, seul le PMOS conduira amenant ainsi une tension ~ 3,3 V à la sortie Q créant une inversion du niveau logique i.e. Q = not (A). Faites attention à un une croyance malheureusement très répandue chez les novices en circuits logiques qui pensent que la tension de sortie provient de l'entrée A; ce qui n'est pas le cas. La tension passe plutôt par le canal du transistor, reliant une ou l'autre des alimentations (Vdd ou Vss) à la sortie en fonction du signal de l'entrée A.

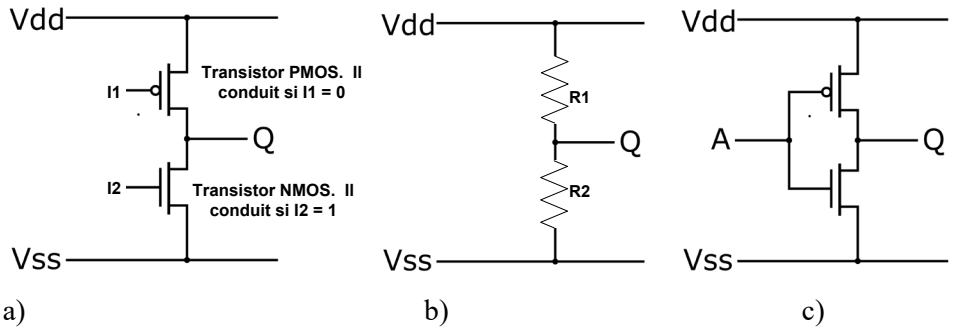

Figure 2.1 – Technologie CMOS a) schéma électrique CMOS, b) résistance équivalente et c) un inverseur

[5] On verra plus loin comment s'assurer que cette condition soit remplie

Dans les cas énumérés précédemment, nous avons expliqué qu'un seul des transistors doit idéalement conduire et jamais les deux en même temps. Il existe un quatrième cas de figure lorsqu'aucun des transistors ne conduit (I1 est à ~3,3 V et I2 à 0 V). À ce moment, la sortie Q est reliée aux 2 alimentations (Vdd et Vss) à travers 2 résistances très élevées (i.e. R1 ≈ R2 ≈ plusieurs MOhms ; figure 2.1 b). C'est ce qu'on appelle une *haute impédance*. Cette caractéristique est présente dans le buffer 3 états représenté par le symbole suivant :

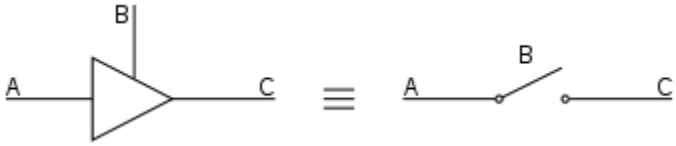

Figure 2.2 – Le buffer 3 états

Le buffer 3 états est principalement utilisé dans les bus de données. Ces bus se retrouvent dans les processeurs d'ordinateur pour échanger de l'information entre différentes sous-sections (32 ou 64 bits actuellement) sur les mêmes fils, par exemple une paire de barrettes de mémoire DDR dans votre ordinateur personnel. L'information transitée peut être bidirectionnelle. Il faut en tout moment que chacun des sous composants n'écrivant pas sur le bus présente une haute impédance à ce bus. Seul le sous composant écrivant sur le bus peut présenter une faible résistance à Vss ou Vdd. La façon de faire cela est de relier ensemble les sorties de tous les buffers 3 états et de n'activer qu'un seul buffer à la fois par une unité de synchronisation qui active les *enables* En_i (figure 2.3). Par exemple, si seul En_1 était activé alors que tous les autres En_i ne l'étaient pas, le signal A_1 sera alors recopié sur le bus de données et la donnée serait disponible à tous les récepteurs. Pour bien comprendre ce qui se passe au niveau électrique, vous pouvez dessiner un schéma des résistances comme à la figure 2.1c pour vous apercevoir que les sorties en haute impédance consommeront très peu de courant et que la faible résistance du buffer activé sera capable d'amener facilement le fil du bus de donnée à ~ 0 V ou ~ 3,3 V selon le cas.

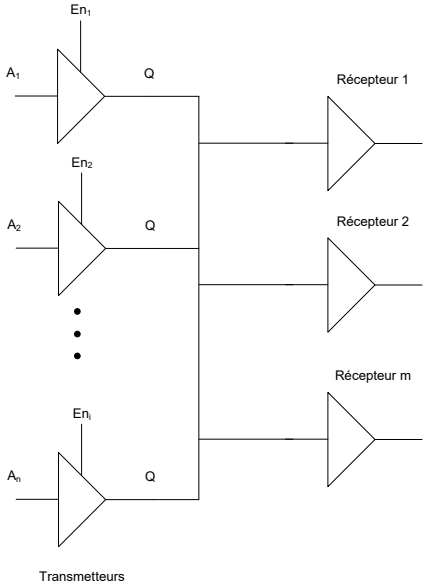

Figure 2.3 – Branchement de buffers 3 états alimentant un fil d'un bus de données.

La Figure 2.3 représente une série de buffers 3 états branchés sur plusieurs récepteurs ; un cas représentatif d'un fil dans un bus de données. La création d'un bus de 32 ou 64 bits exige de recopier 32 ou 64 fois cette structure.

2.2. Niveaux logiques

L'évolution des technologies a été fulgurante au niveau électronique. Beaucoup de gens connaissent la loi de Moore qui stipule que le nombre de transistors pouvant être intégré dans une puce double tous les 18 mois. L'industrie a suivi assez bien cette loi statistique à travers le temps. Cependant, doubler le nombre de transistors signifie fabriquer des transistors plus petits dont les oxydes de grilles sont plus minces et qui supportent des tensions de plus en plus faibles (ex. 5 V, 3,3 V, 1,2 V, 1 V et 0,9 V) avant de claquer. Le type de technologie électronique (ex. CMOS, TTL, ECL etc) affecte également la tension d'opération. Pour ces raisons, plusieurs tensions d'alimentation (Vdd ou Vcc) se retrouvent dans la littérature (figure 2.4). Ces différentes tensions amènent des problèmes de compatibilité intergénérationnelle (CMOS3.3V vs CMOS1.8V) et intertechnologique (CMOS vs TTL). Cette compatibilité est à deux niveaux : compatibilité des tensions et compatibilité des courants. Jusqu'à présent, vous avez vu que les circuits logiques possédaient des niveaux logiques *0* ou *1* mais quand un niveau de tension est-il reconnu comme un niveau logique *0* ou *1* ? Les manufacturiers adoptent une terminologie commune à ce niveau et parlent de

- Tension d'entrée haute minimale (V_{ih_min})
- Tension d'entrée basse maximale (V_{il_max})
- Tension de sortie haute minimale (V_{oh_min})
- Tension de sortie basse maximale (V_{ol_max})

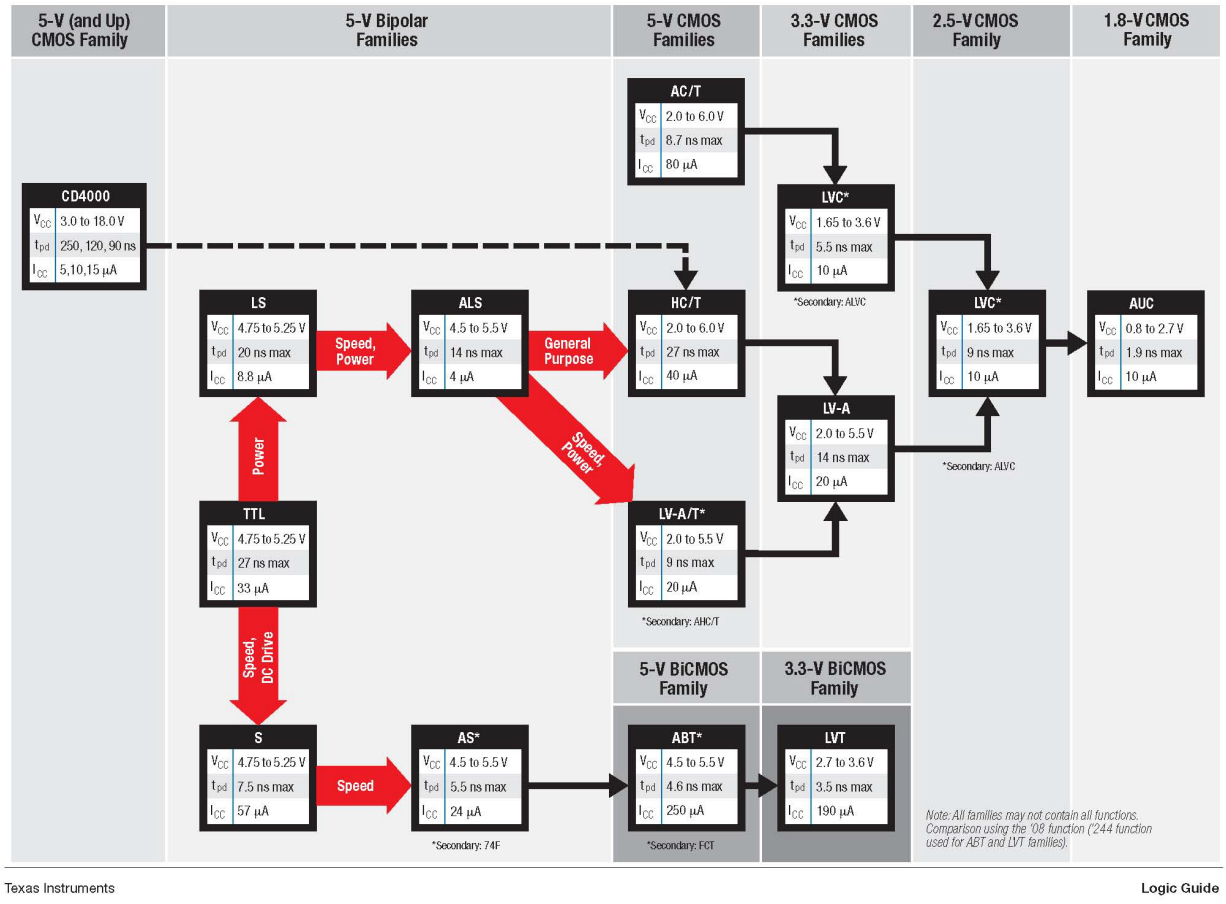

Figure 2.4 – Différentes technologies disponibles sur le marché [6]

[6] Tiré du document sdyu001z.pdf – Logic Guide de la compagnie Texas instrument (www.ti.com).

La tension V_{ih_min} correspond à la tension *minimale* à laquelle la porte logique garantie la reconnaissance du signal comme un *1* alors que la tension V_{il_max} représente la tension *maximale* à laquelle la porte logique garantie la reconnaissance du signal comme un *0*. Comme vous le remarquerez, les tensions V_{ih_min} et V_{il_max} sont différentes l'une de l'autre et il existe un écart entre les 2 où le manufacturier indique un état imprévisible (Figure 2.5). Afin de s'assurer du bon fonctionnement de la porte logique, il faut s'assurer que la tension qui sera amenée à l'entrée de la porte par une autre porte logique sera en tout temps dans sa zone de validité électrique . À ce niveau, les manufacturiers caractérisent leur technologie et indiquent dans une fiche technique quelle sera la tension minimale pour un niveau logique haut V_{oh_min} et maximale pour un niveau logique bas V_{ol_max}. Ainsi, pour s'assurer de la compatibilité, il faut donc s'assurer que la tension V_{oh_min}, qui correspond à la tension minimale produite par une technologie dans un état élevé, de la porte précédente soit supérieure à la tension V_{ih_min} de la porte suivante. De même, il faut s'assurer que la tension V_{ol_max}, qui correspond à la tension maximale produite par une technologie à un état bas, sera inférieure à la tension V_{il_max}. Les équations suivantes représentent les opérations à réaliser.

$$V_{oh\_min} > V_{ih\_min}$$

$$V_{ol\_max} < V_{il\_max}$$

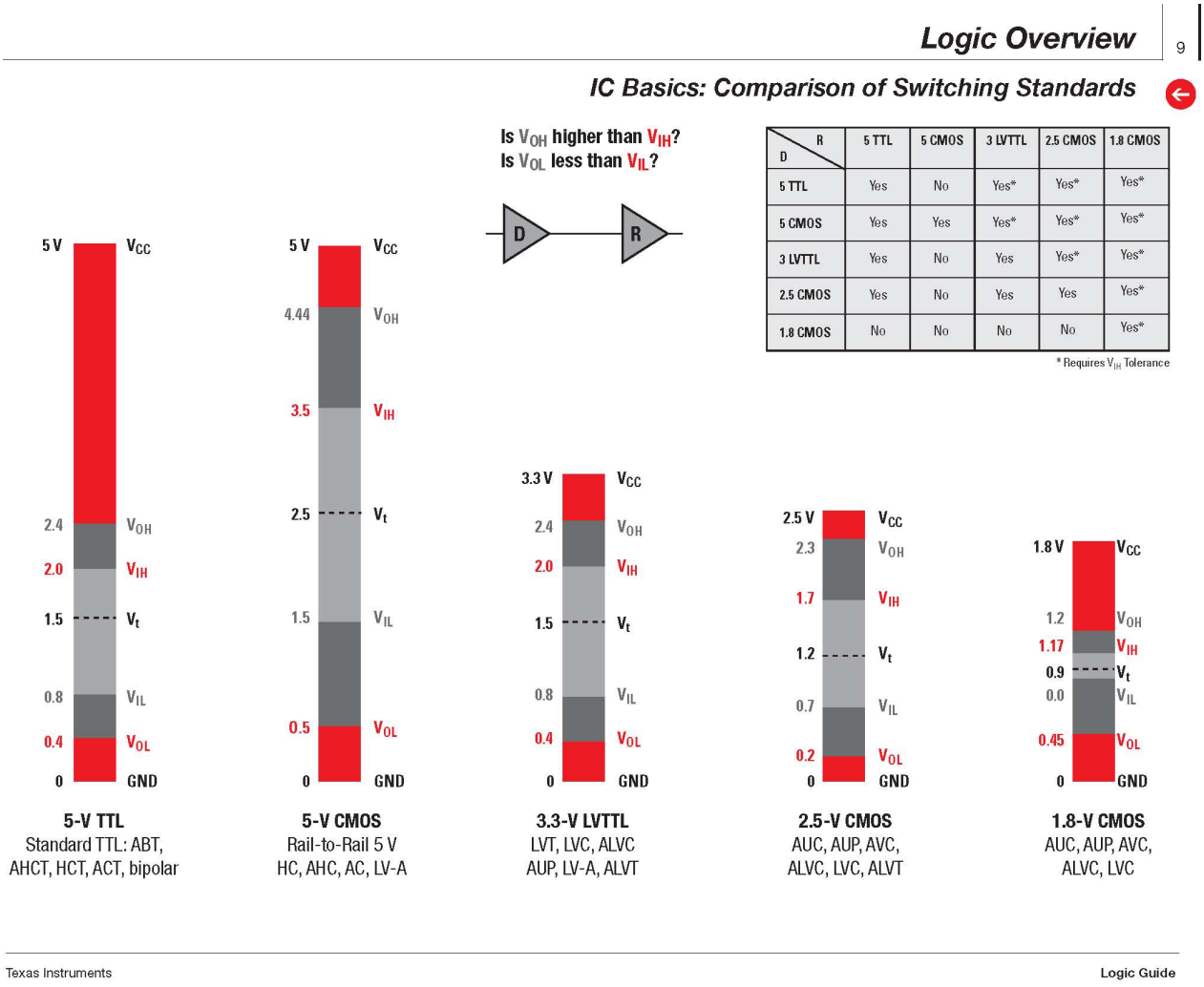

Figure 2.5 – Compatibilité des signaux logiques au niveau des tensions pour différentes technologies

Les tensions de sortie dépendent des technologies utilisées. Ainsi une technologie CMOS assure habituellement un *0* si le signal est < ~30% de la tension d'alimentation et un *1* si ce signal est > ~70%. Les fiches techniques des 74LS00 et le 74HC00 présentent les caractéristiques désirées pour différentes tensions d'alimentation et différentes températures. En regardant attentivement ces spécifications du manufacturier à température ambiante et à une alimentation de ~5 V, on peut voir que la technologie CMOS peut alimenter en tension la technologie TTL mais pas le contraire. En effet, les nombres suivants peuvent être tirés de ces spécifications du manufacturier.

CMOS (74HC00)				TTL (74LS00)			
V_{oh_min}	V_{ol_max}	V_{ih_min}	V_{il_max}	V_{oh_min}	V_{ol_max}	V_{ih_min}	V_{il_max}
4,4 V	0,1 V	3,15 V	1,35 V	2,7 V	0,5 V	3,5 V	2,0 V

Exemple: si une porte CMOS est devant une porte TTL

$V_{oh\_min} > V_{ih\_min}$ 4,4 V > 3,5 V -> Oui

$V_{ol\_max} < V_{il\_max}$ 0,1 V < 2,0 V -> Oui

Exemple si la porte TTL est devant une porte CMOS

$V_{oh\_min} > V_{ih\_min}$ 2,7 V > 3,15 V -> **NON**

$V_{ol\_max} < V_{il\_max}$ 0,5 V < 1,35 V -> Oui.

2.3. Fanout ou sortance

Si les niveaux de tension sont compatibles, une seconde vérification doit être effectuée afin de s'assurer de la fonctionnalité de la compatibilité des technologies. Cette vérification relève de la capacité d'une porte à fournir le courant nécessaire à (aux) l'entrée(s)[7] des autres portes logiques en aval. Cette vérification peut être effectuée par le calcul du *fanout*. Ce dernier consiste à comptabiliser le nombre d'entrées de portes logiques que peut alimenter la sortie d'une porte. Ce nombre se calcule par le ratio des courants en sortie sur le courant en entrée. Évidemment si ce ratio est inférieur à 1, la porte n'est pas capable de pousser ou tirer assez de courant pour alimenter l'entrée d'une autre porte et il y a incompatibilité. Tout comme les tensions, les ratios de courants sont calculés pour des niveaux haut et bas à travers les caractéristiques I_{ih}, I_{il}, I_{ol} et I_{oh}. Cela provient du fait que certaines technologies ne poussent pas autant de courant qu'elles peuvent en tirer. Dans certains cas, comme pour les portes CMOS, le courant de fuite du transistor d'entrée est très minime (~ 1 µA) et identique pour I_{ih} et I_{il}. Les manufacturiers vont alors fusionner ces 2 caractéristiques sous l'acronyme I_{in}.
En regardant les fiches techniques des composantes 74HC00 et 74LS00 pour une alimentation de 3,3 V, il est possible de calculer les ratios suivants en valeur absolue :

$|I_{oh\_CMOS} / I_{ih\_TTL}| =$ |-25 mA / 20 uA| = 1250 (on utilise ici le V_{in} standard à 2,7 V (qui est le plus proche) et non le V_{in} à 7 V)

[7] Note. On parle bien du nombre d'entrées et non du nombre de portes. Une porte peut avoir plusieurs entrées ex une porte NAND a 2 entrées. Chacune des entrées tirera ou poussera du courant selon le niveau logique appliqué. Ainsi, si les 2 entrées d'une porte NAND sont branchées ensembles à la sortie d'une porte NOR, cette dernière verra deux fois plus de courant tiré ou poussé par sa sortie. C'est pour cette raison, qu'il faut compter le nombre d'entrées/sorties et non le nombre de porte.

$|I_{ol\_CMOS} / I_{il\_TTL}|= |25\ mA / -0{,}4\ mA| = 62{,}5$

Une porte HC00 est donc capable d'alimenter 62 entrées de portes TTL au maximum. Vous noterez également que des signes négatifs sont présents dans les spécifications des manufacturiers. Un signe négatif indique que le courant sort de la porte logique. Une deuxième attention doit être apportée à I_{oh} et I_{ol} du 74HC00. Le manufacturier ne fournit que le courant total pouvant être poussé ou tiré d'une patte du circuit et il l'appelle I_{out}. Le courant est le même dans les deux cas i.e. 25 mA.
Lors de la recherche des spécifications I_{ih} et I_{il} il arrive que le manufacturier n'indique pas clairement cette valeur. Ceci est plus fréquent pour les technologies CMOS dont l'entrée est un condensateur. À ce moment, il n'y a pas de courant entrant ou sortant dans l'entrée du dispositif sauf un petit courant de fuite. Le manufacturier a tendance à exprimer les I_{ih} et I_{il} à travers le terme I_I pour identifier le courant de leak. Habituellement la valeur sera exprimée sous forme ± 1 uA. C'est le cas pour le 74V1T04.

Les caractéristiques électriques du FPGA sont regroupés sur le site WEB de l'APP.

2.4. Délais de propagation

Toutes les portes logiques nécessitent un certain temps pour réagir ; elles sont limitées en bande passante. Les technologies les plus récentes commutent généralement plus rapidement que les anciennes technologies car les capacités parasites à charger/décharger sont plus petites (Figure 2.4). Les manufacturiers utilisent une terminologie spécifique pour caractériser les temps de propagation dans les portes logiques appelée t_{PHL} et t_{PLH} et correspondent au délai pour une transition de niveau haut à bas ou de niveau bas à haut. Ces délais sont requis pour calculer la fréquence maximale d'un circuit et éviter les problèmes d'erreur de synchronisation abordée à l'APP de logique séquentielle (voir section *Static hazard* de votre livre et dans Wakerly).

3. Les XOR et XNOR

3.1. XOR et XNOR à 2 variables.

Pour certains circuits, il est préférable de travailler avec des Ou exclusifs ou XOR. Le XOR est particulièrement adapté pour réaliser des circuits d'additionneurs, de calcul de parité ou encore d'inversion de bits. Cependant, il faut être en mesure de les reconnaître lors des simplifications dans les tables de Karnaugh. Débutons par la définition de base du XOR. Très similaire au OR au premier coup d'œil, le XOR a cependant sa sortie à zéro pour les combinaisons A=B tel que le démontre la table de vérité à 2 variables (Tableau 2) où l'on retrouve les variables d'entrées A et B, le OR, le XOR et son inverse le XNOR.

Tableau 3 – OR, XOR XNOR

A	B	A + B	A ⊕ B	(A ⊕ B)'
0	0	0	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	0	1

On peut extraire les équations directement de la table de vérité pour chacune des fonctions

A + B = A'B + AB' + AB
A ⊕ B = A'B + AB'
(A ⊕ B)' = AB + A'B'

Il faut remarquer la structure de l'équation du XOR où chaque variable est en inversée à chacun son tour (A prime avec B, A avec B prime), alors que les 2 premières variables du XNOR ne sont pas inversées et les 2 dernières le sont.

Il est aussi possible de visualiser les XOR et XNOR à partir des tableaux de Karnaugh où les 1 sont en diagonal pour les deux fonctions.

Tableau 4 Table de Karnaugh pour un XOR

A / B	0	1
0	0	1
1	1	0

Tableau 5 Table de Karnaugh pour un XNOR

A / B	0	1
0	1	0
1	0	1

3.2. XOR et XNOR à 3 variables

Le XOR à 3 variables se complexifie par rapport à sa version à deux variables et il faut être particulièrement prudent. La démarche suivante permettra de le faire convenablement et en minimisant les risques d'erreur. En partant de l'équivalence

$$A \oplus B \oplus C = (A \oplus B) \oplus C = A \oplus (B \oplus C),$$

il est possible de remplir la table de vérité. Pour chacune des lignes de la table de vérité, on commence par appliquer l'équation XOR sur les deux premières entrées A et B, ce qui produit un résultat intermédiaire. On le combine ensuite en XOR avec C. Par exemple si A = B = C = 0 on a $(A \oplus B) = (0 \oplus 0) = 0$, et $(0) \oplus C$ est encore égal à 0. On peut inscrire cette valeur à la ligne « 000 ». Similairement pour la seconde ligne où l'on retrouve A = B = 0 et C = 1, le calcul donne $(0 \oplus 0) \oplus 1 = 1$. Compléter cet exercice pour les six autres cas.

Tableau 6 Table de vérité pour XOR3 et XNOR3

A	B	C	A⊕B⊕C	(A⊕B⊕C)'
0	0	0	0	1
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

À partir du Tableau 5, on peut remplir la table de Karnaugh du XOR retrouvé au Tableau 6.

Tableau 7 Tableau de Karnaugh pour un XOR3

A / BC	0	1
00	0	1
01	1	0
11	0	1
10	1	0

On remarque que les 1 forment un zig zag. Réaliser la preuve inverse à partir du tableau de Karnaugh est un peu plus compliquée car l'extraction ne suit pas la logique usuelle des tableaux de Karnaugh. Il faut extraire chacun des termes pris individuellement et faire un peu d'algèbre de Boole. De la Figure 3, on peut donc extraire :

$$F = AB'C' + A'B'C + ABC + A'BC'.$$

Par simplification, on obtient,

$$F = B'(AC' + A'C) + B\ (AC \text{ ou } A'C').$$

On remarque ici que de prendre 2 termes en diagonale dans le tableau de Karnaugh ne donne ***PAS*** un XOR ni un XNOR. Cependant, en observant les termes dans les parenthèses, on peut y retrouver les équations caractéristiques du XOR et du XNOR à 2 variables. On peut réécrire :

$$F = B'(A \oplus C) + B\ (A \oplus C)'.$$

Faisons un changement de variable $W = A \oplus C$ pour mieux visualiser la prochaine étape. On peut alors réécrire :

$$F = B'W + BW',$$

ce qui est un XOR. On peut alors réécrire

$$F = B \oplus W = B \oplus (A \oplus C), \text{ CQFD.}$$

Pour vous en convaincre, réalisez le même exercice pour le XNOR. Vous pouvez également faire le même exercice pour 4 variables et reconnaitre les diagonales dans le tableau de Karnaugh.