

Prise en main de la partie schématique

La programmation se fait toujours à travers un projet. La première chose à apprendre est donc de savoir créer et gérer un projet. On rappelle qu'un projet permet de rassembler plusieurs types de fichiers ensembles.

1. Comment démarrer un projet

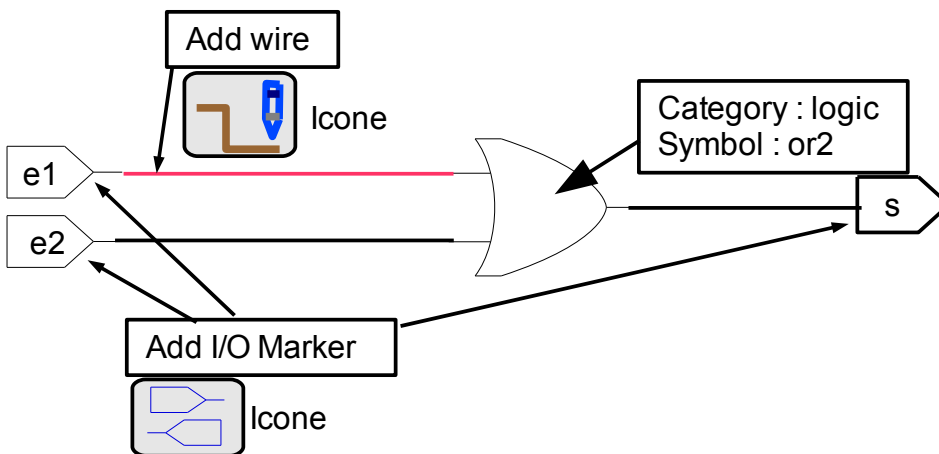
La première chose que l'on se demande est quel est le FPGA cible que l'on utilise et quelle sorte de boîtier on utilise ? En ce qui nous concerne ce sera un SPARTAN 3E 500 (**xc3s500e**) dans un boîtier FG320.

On choisit ensuite le type de langage : pour nous ce sera du schématique... et on se laisse guider.

Notre premier projet sera composé de deux fichiers : un fichier de schéma (extension .sch) et un fichier de contrainte (extension .ucf)

2. Saisir un schéma

On va commencer par un exemple très simple mais qui nous permettra de comprendre comment l'environnement fonctionne.



On a deux entrées que l'on va relier à deux interrupteurs (sw0 et sw1 positionnés respectivement en L13 et L14) et une sortie qui va nous servir à allumer une led (led0 en F12).

Dans l'ordre, il est bon de commencer par les composants, puis par les connexions (wire) pour terminer par les I/O Marker.

3. Saisir le fichier ucf

C'est un fichier texte dont la syntaxe est relativement simple. Il existe un outil graphique pour faire cela mais nous utiliserons un éditeur de texte simple.

```
net "e1" loc="L13";
net "e2" loc="L14";
net "s" loc="F12";
```

Seul la documentation de la carte sur laquelle est votre FPGA vous permet de trouver comment s'appellent les broches de votre FPGA pour les mettre dans le fichier ucf correspondant. Cette documentation vous sera toujours fournie.

Notez que pour ouvrir un fichier ucf, il ne faut pas double-cliquer dessus : cliquer simplement et aller chercher (sur la gauche) dans la fenêtre process :

User Constraints

Edit constraints (text).

4.Compilation

La compilation a comme objectif de réaliser un fichier binaire (d'extension .bit) qui sera téléchargé dans le FPGA. On clique dans la fenêtre projet sur le fichier le plus haut dans la hiérarchie (pour le moment ce sera notre schéma) puis dans la fenêtre process on choisit "Generate Programming File" (double clic)

5.Téléchargement

Le téléchargement dans la carte se fait avec Impact (trouvé dans la fenêtre process) :

Configure Target Device

Manage Configuration Project (Impact)

6.Convention de noms pour les portes

Pour s'y retrouver parmi les nombreuses portes logiques disponibles, nous donnons les indications suivantes. Les noms des portes sont de la forme pNbM

- p est un nom de porte comme and, or, nor, nand, xor et xnor représentant le ET, le OU, le OU-NON le ET-NON, le OU-Exclusif, et l'identité.
- N est le nombre d'entrées de la porte, typiquement de 2 à 9 et même parfois 16.
- b est une lettre suivie d'un chiffre M variant de 1 à typiquement 5 désignant le nombre d'entrées complémentées. Certains composants n'ont pas cette extension, par exemple le ou exclusif va de xor2 à xor9.

La suite est importante pour les projets un peu plus conséquents et peut être passée en première lecture.

7.Renommer un fil (net)

Renommer un fil est utile pour relier un fil à une entrée sans dessiner cette liaison.

Pour renommer un fil, **ne pas passer** par un click droit sur le fil en essayant de trouver "Object Property", mais plutôt :

- edit -> Rename -> rename Selected net puis donner le nom puis OK.

8.Création de plusieurs feuilles

Quand votre dessin devient complexe il faut le répartir sur plusieurs feuilles. Pour se faire :

- click droit dans la feuille -> object properties New

Pendant que vous y êtes, il est possible de changer la taille de la feuille ou des feuilles.

9.Création d'un symbole

Il pourra nous arriver de rassembler un schéma complexe dans un symbole. Ceci peut être fait ainsi :

- File -> New -> Symbol

Utilisation du sorcier (Wizard)

- Tools -> symbol wizard
- Next

Donner un nom au symbole et ajouter le nom des entrées et sorties puis next.

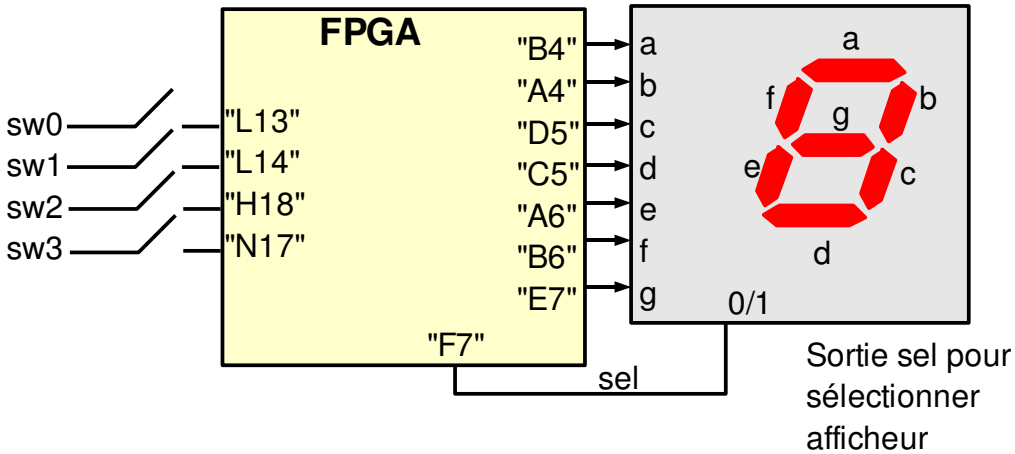
Remarque : Le "symbol wizard" permet de transformer directement votre schéma en cours en symbole si vous cochez "using schematic" et il vous trouve alors tout seul l'ensemble des entrées/sorties.

TP1 - Transcodeur binaire 7 segments

On va réaliser un transcodeur binaire décimal vers un afficheur 7 segments. L'objectif de ce TP est d'utiliser toutes les techniques classiques de synthèse combinatoire. On utilisera un FPGA de type SPARTAN 3E 500 (**xc3s500e**) dans un boîtier FG320.

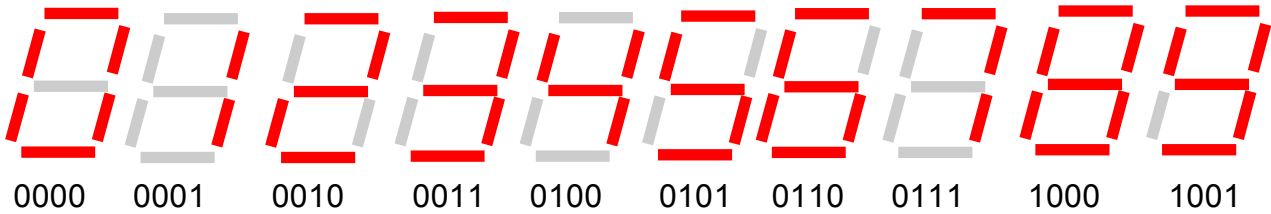
1. Présentation du sujet

Un schéma présente de manière symbolique ce que l'on cherche à faire.



sel est choisi à 0 pour sélectionner l'afficheur de droite.

La valeur binaire (de 0 à 9) choisie sur les interrupteurs est convertie pour être affichée :



2. Fichier ucf

Pour éviter de chercher nous donnons quelques contenus du fichier ucf :

```
NET "SW<0>" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<1>" LOC = "L14" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<2>" LOC = "H18" | IOSTANDARD = LVTTTL | PULLUP ;
NET "SW<3>" LOC = "N17" | IOSTANDARD = LVTTTL | PULLUP ;
# ===== 6-pin header J1 =====
# These four connections are shared with the FX2 connector
#NET "J1<0>" LOC = "B4" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
#NET "J1<1>" LOC = "A4" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
#NET "J1<2>" LOC = "D5" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
#NET "J1<3>" LOC = "C5" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
```

```
# ===== 6-pin header J2 =====
# These four connections are shared with the FX2 connector
#NET "J2<0>" LOC = "A6" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
#NET "J2<1>" LOC = "B6" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
#NET "J2<2>" LOC = "E7" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
#NET "J2<3>" LOC = "F7" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 6 ;
```

Pour savoir comment sont connectés nos afficheurs à notre carte, il faut télécharger la documentation des afficheurs : dans google [digilent pmodssd](#) (donnée un peu plus loin dans ce document)

3. Table de vérité

Réaliser la table de vérité correspondant au cahier des charges si on doit sortir un 1 pour afficher un segment. Réfléchissez sur la première ligne avant de poursuivre.

sw3	sw2	sw1	sw0	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1							
0	0	1	0							
0	0	1	1							
0	1	0	0							
0	1	0	1							
0	1	1	0							
0	1	1	1							
1	0	0	0							
1	0	0	1							

4. Synthèse de la sélection

Pour forcer à des valeurs prédéfinies :

- Mettre à 1 avec : Catégorie générale -> symbol Vcc
- Mettre à 0 avec : Catégorie générale -> symbol gnd

A simplifier : de toute façon les noms des portes ne sont pas corrects car fait avec 0 pour allumer!
Utiliser deux types de synthèses seulement !!!!

5. Synthèse du segment "a"

Sans chercher à simplifier, rechercher l'équation disjonctive correspondant au segment a. Donner le schéma ET/OU, puis implanter avec des portes "et" (= and4b3) et une porte "ou" (= or2)

6. Synthèse du segment "b"

Montrer que b se réalise avec une porte and2b1 et une porte ou exclusif xor2. Si vous ne connaissez pas bien le ou exclusif faites la synthèse à partir de la forme disjonctive non simplifiée (2 portes and4b2 et une or2)

7. Synthèse du segment "c"

Montrer qu'une seule porte suffit pour la synthèse.

8. Synthèse du segment "d"

On vous demande de trouver la forme conjonctive simplifiée et d'en faire une synthèse avec les portes nécessaires synthèse OU/ET.

9. Synthèse du segment "e"

On vous demande de trouver la forme conjonctive simplifiée et d'en faire une synthèse OU/ET et de la transformer en synthèse en NORs.

10. Synthèse du segment "f"

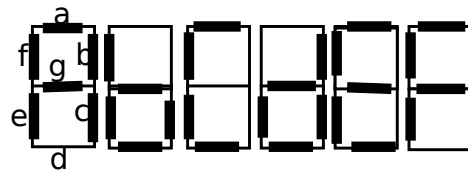
On vous demande de trouver la forme disjonctive simplifiée et d'en faire une synthèse ET/OU et de la transformer en synthèse en NANDs.

11. Synthèse du segment "g"

Faire une synthèse ET/OU utilisant deux portes and3xx et une porte or2.

12. Et pour finir le TP en beauté...

S'il vous reste du temps en fin de TP vous pouvez refaire un décodage complet hexadécimal en ajoutant les affichages de A,b,C,d,E,F à la suite de 0->9. L'entrée n'est plus décimale mais hexadécimale. Vous utiliserez les synthèses de votre choix.



TP2 - Des tables de vérités aux schémas

L'objectif de ce TP est d'augmenter un peu la productivité en schématique en utilisant, non plus les portes standards, mais des composants permettant d'implanter directement des tables de vérité dans notre FPGA (sans les simplifier). On utilisera encore un SPARTAN 3E 500 (xc3s500e) dans un boîtier FG320.

1. Des tables de vérité à l'hexadécimal

Une table de vérité de trois entrées peut être représentée par un nombre 8 bits que l'on convertit en hexadécimal. Soit donc la table de vérité suivante (trois entrées notées e0, e1 et e2, une sortie notée s) :

e2	e1	e0	s	
0	0	0	0	b0
0	0	1	1	b1
0	1	0	1	b2
0	1	1	0	b3
1	0	0	1	b4
1	0	1	0	b5
1	1	0	1	b6
1	1	1	0	b7

Vous pouvez synthétiser la table de vérité à l'aide d'un seul nombre sur 8 bit (poids faible en haut) :

- en binaire ce nombre vaut : 0b01010110

- en hexadécimal ce nombre vaut : 0x56 (**X"56"** en VHDL)

Aucune simplification à faire ici

La valeur hexadécimale 56 est la valeur qu'il vous faudra utiliser dans la partie INIT d'une (catégorie) LUT avec un composant **lut3**.

Pour initialiser une LUT, double cliquer dessus et remplir le champ INIT.

Combien de bits nécessite l'initialisation d'une **lut4** à 4 entrées ? Combien de chiffres hexadécimaux ?

2. Transcodeur hexadécimal vers sept segments et LUTs

On va reprendre le transcodeur binaire hexadécimal vers un afficheur 7 segments. L'objectif de ce TP est d'utiliser toutes les techniques classiques de synthèse combinatoire spécifiques aux FPGA pour réaliser un composant déjà réalisé au TP précédent, et ce, même si vous n'avez pas eu le temps de finir ce TP en question.

A partir de la table de vérité complète de la question "12 Et pour finir le TP en beauté...", réaliser la synthèse complète du décodeur sept segments (en moins d'une heure). Il s'agit de dessiner et configurer 7 **lut4**.

3. Les multiplexeurs

Les multiplexeurs représentent une autre technique pour éviter de simplifier et faire une synthèse. Encore une fois, la table de vérité suffit.

Soit donc la table de vérité suivante :

S2	S1	S0	s	
0	0	0	0	D0
0	0	1	1	D1
0	1	0	1	D2
0	1	1	0	D3
1	0	0	1	D4
1	0	1	0	D5
1	1	0	1	D6
1	1	1	0	D7

On a renommé les entrées pour qu'elles correspondent aux entrées de sélection du MUX
La dernière colonne a été renommée
Dernière colonne + avant dernière dit à quoi relier les entrées du MUX

Aucune simplification à faire ici
--

Le principe de synthèse est le suivant :

- les entrées de la table de vérité iront sur les entrées de sélections du multiplexeur (S0,S1,S2)
- les entrées D0..D7 sont reliées à des un ou des zéros.

Le problème est que la taille d'un multiplexeur augmente très rapidement avec le nombre d'entrées.

Travail à réaliser : Utiliser un multiplexeur "m16_1e" en lieu et place d'une LUT pour la synthèse d'un segment quelconque de la synthèse précédente. **Ne pas oublier de mettre l'entrée E à Vcc.**

4.Les multiplexeurs (suite)

Peut-on vraiment oublier les tableaux de Karnaugh et autres techniques combinatoires pour autant ? Non car vous n'aurez pas toujours un multiplexeur de la bonne taille... et il faut alors improviser.

Retirer une variable

Travail à réaliser : on vous demande de chercher parmi les segments a, ..., g s'il y en a un qui ne dépend que de trois variables. Pour cela on utilise ce que l'on peut, Karnaugh par exemple mais on ne cherche pas à simplifier de façon optimale mais en faisant des regroupements susceptibles de retirer une variable. Si on trouve on implante avec un MUX "m8_1e". **Ne pas oublier de mettre l'entrée E à Vcc.**

Il nous manque une variable

Travail à réaliser : On va implanter un segment quelconque avec un MUX "m8_1e". Il nous manque donc une entrée. Puisqu'on a trois entrées de sélection et quatre entrées générales, on choisit 3 entrées générales que l'on amène sur les trois entrées de sélection, puis sur les 8 entrées générales D0,..., D7 on apporte une des 4 fonctions suivantes de l'entrée laissée pour compte : Vcc, gnd, swi ou /swi.

Il nous manque deux variables

Travail à réaliser : On va implanter un segment quelconque avec un MUX "m4_1e". Il nous manque donc deux entrées. Puisqu'on a deux entrées de sélection et quatre entrées générales, on

choisit 2 entrées générales que l'on amène sur les deux entrées de sélection et sur les 8 entrées générales D0,..., D7 on apporte une des 16 fonctions des deux entrées laissées pour compte.

5. Complément sur usbmod et l'ucf associé.

ucf pour 8 sorties sur usbmod

```
# These four connections are shared with the J1 6-pin accessory header
NET "PA<0>" LOC = "B4" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<1>" LOC = "A4" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<2>" LOC = "D5" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<3>" LOC = "C5" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<4>" LOC = "A6" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<5>" LOC = "B6" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<6>" LOC = "E7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PA<7>" LOC = "F7" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
```

ucf pour 8 entrées sur usbmod

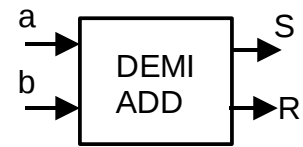

```
NET "PB<0>" LOC = "A13" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<1>" LOC = "B13" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<2>" LOC = "A14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<3>" LOC = "B14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<4>" LOC = "C14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<5>" LOC = "D14" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<6>" LOC = "A16" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
NET "PB<7>" LOC = "B16" | IOSTANDARD = LVCMOS33 | SLEW = FAST | DRIVE = 8 ;
```

TP3 Arithmétique (Schématique)

L'arithmétique consiste à implanter des fonctions de base de l'arithmétique, c'est à dire addition, soustraction et multiplication. L'utilisation de l'arithmétique dans un FPGA doit suivre ses propres règles. Nous allons commencer par examiner l'addition, d'abord sans se préoccuper du fait que l'on utilise un FPGA, puis on cherchera à utiliser les composants Xilinx. On utilisera encore un SPARTAN 3E 500 (**xc3s500e**) dans un boîtier FG320.

1. Du demi-additionneur à l'additionneur 1 bit

Les opérations d'addition sur 1 bits sont très simples et donnent un résultat sur 2 bits. Le bit de poids fort est appelé R (pour retenue= carry en anglais) et le poids faible est appelé S (pour somme).

	<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>b</th> <th>a</th> <th>R</th> <th>S</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> </tr> <tr> <td>0</td> <td>1</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>0</td> <td>0</td> <td>1</td> </tr> <tr> <td>1</td> <td>1</td> <td>1</td> <td>0</td> </tr> </tbody> </table>	b	a	R	S	0	0	0	0	0	1	0	1	1	0	0	1	1	1	1	0	$S = a \oplus b$ $R = a \cdot b$																						
b	a	R	S																																									
0	0	0	0																																									
0	1	0	1																																									
1	0	0	1																																									
1	1	1	0																																									
	<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>R_{n-1}</th> <th>b_n</th> <th>a_n</th> <th>00</th> <th>01</th> <th>11</th> <th>10</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td>0</td> <td style="border: 2px solid black;">1</td> <td>0</td> </tr> <tr> <td>1</td> <td>0</td> <td style="border: 2px solid black;">1</td> <td style="border: 2px solid black;">1</td> <td style="border: 2px solid black;">1</td> <td style="border: 2px solid black;">1</td> <td style="border: 2px solid black;">1</td> </tr> </tbody> </table> $S_n = a_n \oplus b_n \oplus R_{n-1}$ $R_n = a_n \cdot b_n + R_{n-1} \cdot (a_n \oplus b_n)$	R_{n-1}	b_n	a_n	00	01	11	10	0	0	0	0	0	1	0	1	0	1	1	1	1	1	<table border="1" style="border-collapse: collapse;"> <thead> <tr> <th>R_{n-1}</th> <th>b_n</th> <th>a_n</th> <th>00</th> <th>01</th> <th>11</th> <th>10</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td style="border: 2px solid black;">1</td> <td style="border: 2px solid black;">1</td> <td>0</td> <td>0</td> <td style="border: 2px solid black;">1</td> </tr> <tr> <td>1</td> <td style="border: 2px solid black;">1</td> <td>0</td> <td>0</td> <td>0</td> <td style="border: 2px solid black;">1</td> <td>0</td> </tr> </tbody> </table> S_n	R_{n-1}	b_n	a_n	00	01	11	10	0	0	1	1	0	0	1	1	1	0	0	0	1	0
R_{n-1}	b_n	a_n	00	01	11	10																																						
0	0	0	0	0	1	0																																						
1	0	1	1	1	1	1																																						
R_{n-1}	b_n	a_n	00	01	11	10																																						
0	0	1	1	0	0	1																																						
1	1	0	0	0	1	0																																						

Remarquez que R_n n'est pas simplifié au mieux pour faire apparaître un OU exclusif.

Travail à réaliser

Implanter un additionneur 1 bit avec deux ou-exclusifs à deux entrées et trois portes NANDs. Pour les tests, les entrées seront reliées à des interrupteurs et les sorties à des LEDs.

Copier 4 fois le schéma précédent et montrer que l'on peut réaliser ainsi un additionneur 4 bits. N'oubliez pas que la somme de deux nombres de 4 bits peut donner un nombre de 5 bits.

2. Additionneur et soustracteur sur 1 bit

On cherche maintenant à réaliser un circuit capable d'additionner ou de soustraire sur un seul bit. Puisque l'on sait réaliser un additionneur, on va se concentrer sur le soustracteur. Les trois entrées du soustracteur seront aussi appelées a_n , b_n et R_{n-1} . Un soustracteur sur 1 bit réalise l'opération $a_n - b_n - R_{n-1}$. Le résultat est sur deux bits en complément à deux.

Préparation

On cherche à représenter un nombre positif ou négatif sur deux bits seulement. Avec le complément à deux seules les valeurs 1, 0, -1 et -2 sont représentables.

Trouver les quatre représentations en complétant pour les nombres négatifs.

nombre	binaire	nombre	binaire	nombre	binaire	nombre	binaire
1	(01) ₂	0	(00) ₂	-1		-2	

Chercher la table de vérité du soustracteur 1 bit.

a	b	Rin	Rout	D	Valeur décimale (a - b - Rin)
0	0	0			0
0	0	1			-1
0	1	0			-1
0	1	1			-2
1	0	0			1
1	0	1			0
1	1	0			0
1	1	1			-1

Travail à réaliser

Réaliser un schéma en modifiant celui de l'additionneur en lui ajoutant deux inverseurs.

Sachant qu'un ou exclusif a la propriété d'inverser ou pas, remplacer les deux inverseurs par deux ou exclusifs. Le schéma obtenu possède une quatrième entrée S de sélection qui permet de choisir si l'on fait une addition ou une soustraction.

Implanter ce schéma en schématique Xilinx et tester.

3. Additionneur 4 bits

Comme indiqué en introduction, si vous avez de l'arithmétique à faire il faut mieux utiliser des composants tout faits. En effet, tout FPGA moderne possède un certain nombre de parties pré-câblées réalisant ces fonctions. On utilisera ainsi le composant **ADD4** dans la catégorie "arithmetic". Les entrées de ce composant seront des interrupteurs extérieurs mais ses sorties seront destinées à être affichées sur un afficheur sept segments... et c'est là que les choses se compliquent un peu.

Préparation

Montrer que dans le cas où la somme de deux nombres BCD sur 4 bits dépasse 9 (cette somme n'est pas BCD mais binaire à cause de ADD4), il suffit d'ajouter 6 au résultat pour obtenir le nombre BCD correct.

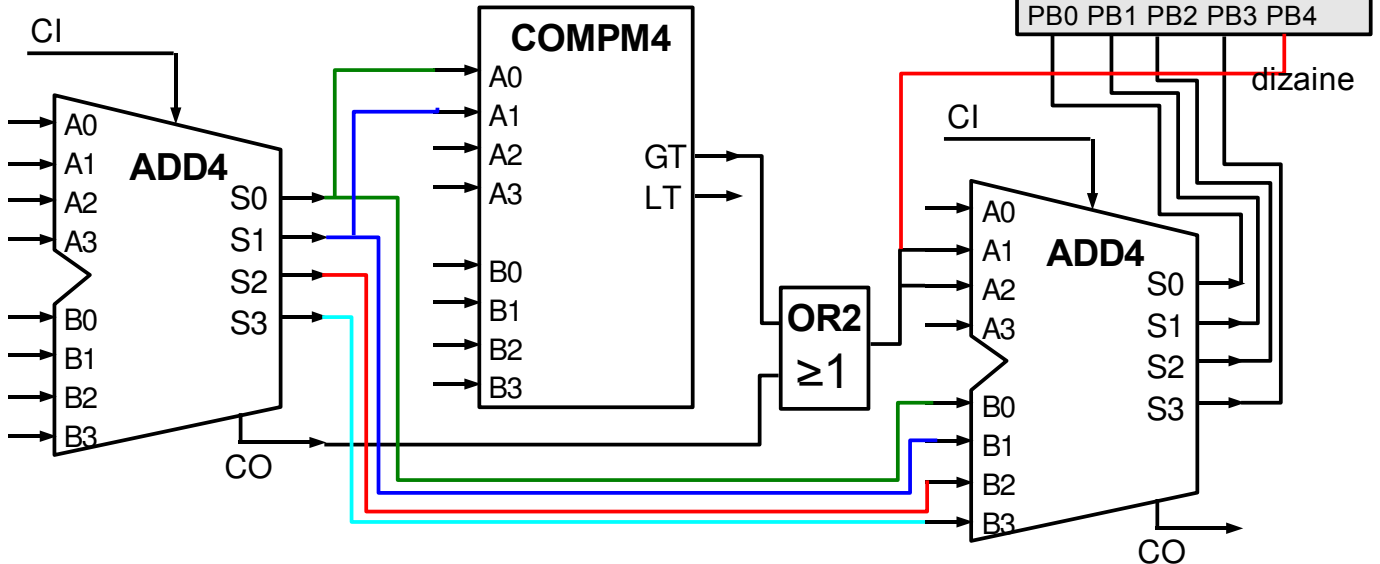
Travail à réaliser

On va se contenter d'utiliser le composant **ADD4** de Xilinx dans le cas d'entrées BCD. Cela veut dire que chacune des entrées ne peut dépasser $(9)_{10} = (1001)_2$.

On vous demande d'utiliser deux composants **ADD4** de Xilinx pour obtenir la somme BCD (sur 5 bits) de deux nombres BCD (sur 4 bits). Le deuxième additionneur est là pour ajouter 6 en cas de dépassement de 9. Il vous faut naturellement ajouter la logique capable de détecter un dépassement de 9. Un comparateur **COMPM4** et un OU devraient suffire.

Les étudiants débutants sont invités à tester d'abord un additionneur **ADD4** seul avec l'afficheur **USBMOD**, puis à ajouter le comparateur **COMPM4** et montrer que pour un résultat qui dépasse strictement 15 (autrement dit quand C0 vaut 1) le comparateur ne s'en rend pas compte !!! d'où la présence du **OR2** qui résoud ce problème.

Schéma à compléter



Cet ensemble est-il capable de gérer correctement la retenue CI de **ADD4** (à gauche) ? Autrement dit si CI passe à un, le résultat final est-il encore correct ?

4. Additionneur/soustracteur 4 bits

L'additionneur soustracteur existe comme composant tout fait **ADSU4** chez Xilinx, mais nous allons le réaliser à partir de **ADD4**. pour en comprendre le fonctionnement de manière intime, en particulier la notion de complément à deux.

Préparation

Calculer les limites d'une représentation en complément à deux sur 4 bits notées B_4 et $-(B_4+1)$ dans la suite.

Si sel = 0 on additionne et le nombre résultat se trouve en sortie du premier additionneur sur 5 bits (ou en sortie du deuxième additionneur mais avec comme 5^o bit CO_1 et non CO_2)

Si sel = 1 les compléments sont réalisés et on additionne 1 à l'aide de CI_1 . On fait donc un complément à deux de $B = /B+1$, mais on reste sur 4 bit. Le premier additionneur fait donc $S=A+/B+1=A-B$. Montrer que si B dépasse B_4+1 , cela ne fonctionne plus. Prendre par exemple $B=B_4+2$ et faire son complément à deux en montrant qu'alors le bit à gauche vaut 0 et qu'ainsi ce nombre ne peut être interprété comme négatif.

Dans le cas de la soustraction, Sel=1, montrer à l'aide de quelques exemples qu'un résultat positif positionne CO_1 à 1 tandis qu'un résultat négatif le positionne à 0 (choisir B entre 1 et B_4+1 pour la soustraction).

Pour afficher le résultat de la soustraction correctement sur un éventuel afficheur 7 segments et une diode LED de signe, il faut refaire un complément à deux pour retrouver la valeur absolue. C'est

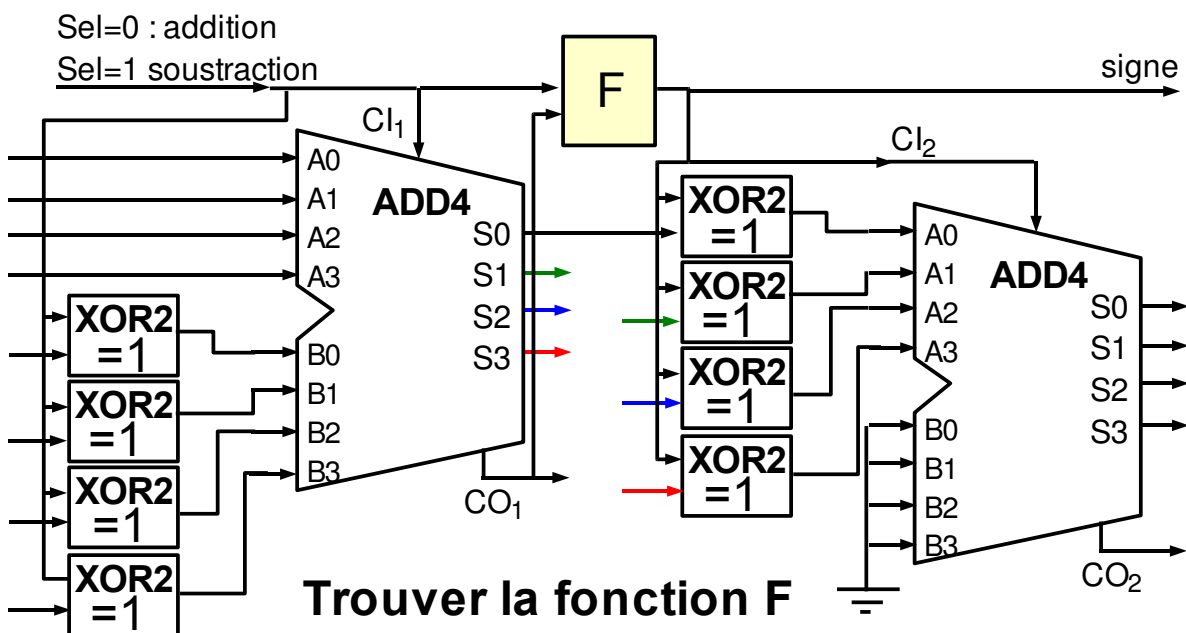
l'objet du deuxième additionneur. Ainsi, si résultat positif LED éteinte et valeur absolue sur afficheur et si résultat négatif LED allumée et valeur absolue sur afficheur. Le schéma complet est présenté ci-dessous.

Travail à réaliser

Chercher la fonction F du schéma ci-après.

Implanter l'ensemble et tester. Donner les plages pour lesquelles l'affichage du résultat d'une addition fonctionne correctement si le résultat est considéré comme binaire sur 5 bits (CO₁,S₃,S₂,S₁,S₀). Comme il est sur 5 bits on ne peut pas l'afficher sur un ou deux afficheurs

Tester si l'affichage d'une soustraction fonctionne correctement pour les valeurs précalculées dans la préparation en restant en binaire ou en utilisant un l'afficheur **USBMOD**.



TP4 Introduction à VHDL

On utilisera encore et toujours un SPARTAN 3E 500 (xc3s500e) dans un boîtier FG320.

1. Au commencement était la table de vérité...

Soit la table de vérité suivante :

a3	a2	a1	a0	s1	s0	
0	1	0	1	1	1	<code>library IEEE;</code>
0	1	1	0	0	1	<code>use IEEE.STD_LOGIC_1164.ALL;</code>
1	1	0	1	1	0	<code>ENTITY demo IS PORT(a : in STD_LOGIC_VECTOR(3 DOWNTO 0); -- 4 entrées s : out STD_LOGIC_VECTOR(1 DOWNTO 0)); -- 2 sorties END demo;</code>

Il manque 13 lignes à cette table, qui sont retirées, mais donnent des 0 en sortie

Notez que la partie gauche de la table de vérité appelée partie SI concerne les entrées, et la partie droite (appelée ALORS) concerne les sorties. La donnée d'une table de vérité permet directement de trouver l'entité. Elle permet aussi d'écrire directement l'architecture avec un "with select when" :

```

1      ARCHITECTURE mydemo OF demo IS
2      BEGIN
3          WITH a SELECT                --style with select when
4              s <= "11" WHEN "0101", -- premiere ligne
5                  "01" WHEN "0110", -- deuxieme ligne
6                  "10" WHEN "1101", -- troisieme ligne
7                  "00" WHEN OTHERS; -- pour les 13 lignes retirées
8      END mydemo;
```

Travail à réaliser

exo1 : Refaire le transcodeur complet d'affichage sept segments du TP1. Ce transcodeur sera utilisé très souvent par la suite, il sera donc important d'en mettre une copie de côté.

exo2 : Modifier l'exo1 pour gérer une sortie de votre choix par une équation (le reste est géré par un with select when). Il faudra naturellement aussi modifier le "with select when" !

2. Puis naquit l'horloge....

L'horloge est l'apanage du séquentiel. Les fronts d'horloge sont utilisés avec un "IF" entouré forcément d'un "PROCESS".

```

1      -- devant entité ajouter : use IEEE.STD_LOGIC_ARITH.ALL;
2      -- devant entité ajouter : use IEEE.STD_LOGIC_UNSIGNED.ALL;
3      PROCESS (clk) BEGIN
4          IF clk'event and clk='1' THEN -- ou IF rising_edge(clk) THEN
5              q <= q + 1;
6          END IF;
7      END PROCESS;
```

Ce qui se trouve après le "IF clk'event" est une équation de récurrence : q se trouve à gauche de

l'équation et doit par conséquent être considéré comme une sortie. Mais il se trouve aussi à droite, il doit par conséquent être considéré comme une entrée.

Les compteurs permettent de diminuer la fréquence d'horloge. Vous pouvez compléter vos connaissances sur les compteurs en VHDL sur le site http://fr.wikibooks.org/wiki/TD3_VHDL_Compteurs_et_registres

Travail à réaliser

Exo3 : Vous disposez d'une horloge 50MHz en broche "C9" de votre composant FPGA SPARTAN 3E. Réaliser à l'aide d'un compteur 24 bits une horloge d'environ 3 Hz. Sortie sur une LED, le clignotement de celle-ci doit être visible à l'œil.

Indications

```

1      # ==== Clock inputs (CLK) ====
2      NET "clk" LOC = "C9" | IOSTANDARD = LVCMOS33 ;
3      # ==== Discrete LEDs (LED) ====
4      # These are shared connections with the FX2 connector
5      NET "LED<0>" LOC = "F12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
6      NET "LED<1>" LOC = "E12" | IOSTANDARD = LVTTTL | SLEW = SLOW | DRIVE = 8 ;
    
```

Idem à http://fr.wikibooks.org/wiki/TD3_VHDL_Compteurs_et_registres#Exercice_2 avec un nombre de bits différent.

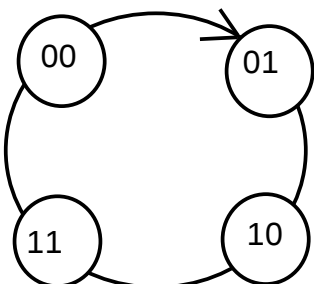
L'horloge que l'on vient de réaliser sera utilisée presque systématiquement par la suite. Cette année elle sera très vite remplacée par un processeur (dès le TP5).

Par abus de langage on appellera compteur dans la suite un élément séquentiel qui comporte une horloge mais qui ne compte pas forcément en binaire. Dans ce dernier cas, son équation de récurrence ne peut donc pas s'écrire simplement à l'aide d'une addition.

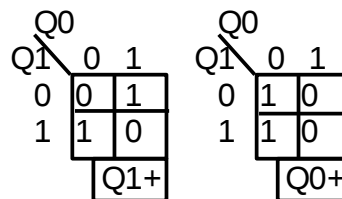
3.Compteur décimal à sortie directe sept segments (sur un digit)

Rappels

Avant toute chose, il nous faut nous demander comment utiliser des équations de récurrences booléennes



État présent		État futur	
Q1	Q0	Q1 ⁺	Q0 ⁺
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0



Nous avons à gauche un diagramme d'évolution, au centre un tableau état présent état futur et à droite les tableaux de Karnaugh permettant de trouver les équations de récurrences.

Équations de récurrence :

$$Q_1^+ = Q_1 \oplus Q_0$$

$$Q_0^+ = /Q_0$$

Voici les extraits importants du programme VHDL correspondant :

```

1      PROCESS (clk) BEGIN
2          IF clk'event and clk='1' THEN -- ou IF rising_edge(clk) THEN
3              -- équations de récurrences ici
4              q1 <= q1 xor q0;
5              q0 <= not q0;
6          END IF;
7      END PROCESS;
    
```

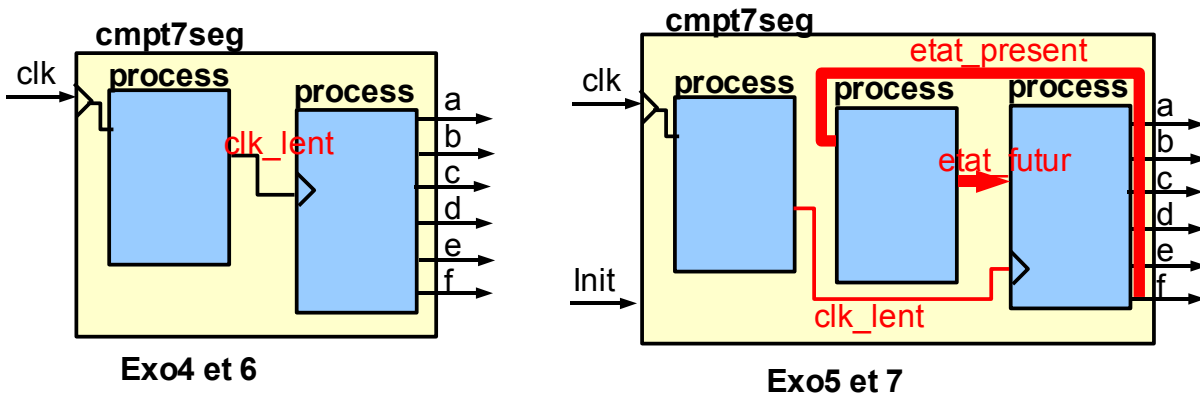
Préparation

Nous désirons réaliser ici un compteur mais qui donne sa sortie directement sur un afficheur 7 segments. Ce compteur comporte donc 7 sorties (a, b, c, d, e, f et g). Un tel compteur possède un nombre d'états de $2^7=128$ mais n'en utilise en fait que 10.

Remplir la table de transitions ci-dessous

	Etat présent							Etat Futur						
	a	b	c	d	e	f	g	a+	b+	c+	d+	e+	f+	g+
zero	1	1	1	1	1	1	0							
un	0	1	1	0	0	0	0							
deux	1	1	0	1	1	0	1							
trois	1	1	1	1	0	0	1							
quatre	0	1	1	0	0	1	1							
cinq	1	0	1	1	0	1	1							
six	1	0	1	1	1	1	1							
sept	1	1	1	0	0	0	0							
huit	1	1	1	1	1	1	1							
neuf	1	1	1	1	0	1	1							

Travail à réaliser (si peu de temps aller directement à l'exo 6)



Exo4 :

- 1°) Écrire les équations logiques a+, b+, c+, d+, e+, f+, g+ en fonction des entrées a, b, c, d, e, f, g.
- 2°) Il nous reste un problème important à résoudre, c'est l'initialisation. En effet pour bien faire il

faudrait étudier les $127-10=117$ états restants pour voir comment ils se connectent sur notre cycle. C'est un travail important qu'il est impossible de réaliser à moins d'y passer 117 soirées (à raison d'une transition par soirée) soit presque 4 mois !!!

Pour éviter cela on va prévoir une entrée d'initialisation appelée Init au cas où à la mise sous tension on se trouve dans un état non prévu. Cette entrée fonctionnera de manière synchrone, lorsqu'elle sera à 1 un front d'horloge provoquera l'affichage du 0 en sortie du compteur.

Écrire maintenant les équations de récurrence trouvées en 2°) en ajoutant convenablement l'entrée Init.

3°) Vous êtes prêt maintenant à écrire le programme VHDL complet. Écrire ces équations dans le langage VHDL. Implanter en utilisant et modifiant l'horloge réalisée dans la section 2 pour pouvoir l'utiliser dans le composant : **on utilisera un process pour le diviseur de fréquence et un process pour le compteur donc une seule entité.**

```

1      ENTITY cmpt7seg IS
2          PORT (CLK_50MHz : IN STD_LOGIC;
3              a,b,c,d,e,f,g : OUT STD_LOGIC);
4      -- ou alors : s7segs : out std_logic_vector(6 downto 0));
5      END cmpt7seg;
```

Vous venez ainsi d'apprendre à assembler deux blocs avec des process. Le travail d'assemblage de blocs est une tâche qui sera souvent demandée dans ces TPs. Elle sera réalisée avec une technique différente dans le prochain TP.

Exo5 : Refaire le même travail avec une entrée d'initialisation.

```

1      ENTITY cmpt7seg IS
2          PORT (CLK_50MHz, Init : IN STD_LOGIC;
3              a,b,c,d,e,f,g : OUT STD_LOGIC);
4      -- ou alors : s7segs : out std_logic_vector(6 downto 0));
5      END cmpt7seg;
```

Remarque

Il existe une méthode qui évite le passage par les équations de récurrences : on part du graphe d'évolution et on obtient le programme VHDL :

```

1      library IEEE;
2      use IEEE.STD_LOGIC_1164.ALL;
3      ENTITY cmpt IS PORT (
4          clk: IN STD_LOGIC;
5          q : INOUT STD_LOGIC_VECTOR(1 DOWNTO 0));
6      END cmpt;
7      ARCHITECTURE acmpt OF cmpt IS -- comment éviter les equations
8          BEGIN
9              PROCESS (clk) BEGIN
10                 IF (clk'EVENT AND clk='1') THEN
11                     CASE q is --style case when
12                         WHEN "00" => q <="01"; -- premiere transition
13                         WHEN "01" => q <="10"; -- deuxieme transition
14                         WHEN "10" => q <="11"; -- troisieme transition
15                         WHEN OTHERS => q <="00"; -- quatrieme transition
16                     END CASE;
17                 END IF;
18             END PROCESS;
19     END acmpt;
```

Le style "case when" est au graphe d'évolution ce que le "with select when" est à la table de vérité : un moyen d'éviter les équations.

Travail à réaliser

Exo6 : Refaire le même travail que l'exo5 en utilisant un **case when** et deux **process** : un pour créer l'horloge lente et un pour le compteur set segments..

```

1         ENTITY cmpt7seg IS
2           PORT (CLK_50MHz : IN STD_LOGIC;
3             s_7segs : OUT STD_LOGIC_VECTOR(6 DOWNTO 0));
4         END cmpt7seg;
```

Remarquez l'utilisation du STD_LOGIC_VECTOR au lieu des sorties individuelles. Le fait qu'il soit en OUT implique que l'on utilisera un signal interne pour les "CASE".

Il est possible d'utiliser un programme utilisant deux "process" comme indiqué dans le cours en pages 137, ..., 140. En voici un exemple :

```

1         library IEEE;
2         use IEEE.STD_LOGIC_1164.ALL;
3         ENTITY cmpt IS PORT (
4           clk: IN STD_LOGIC;
5           q : OUT STD_LOGIC_VECTOR(1 DOWNTO 0));
6         END cmpt;
7         ARCHITECTURE acmpt OF cmpt IS -- comment éviter les equations
8         SIGNAL etat_present, etat_futur :STD_LOGIC_VECTOR(1 DOWNTO 0);
9         BEGIN
10          -- calcul de l'état futur en fonction de l'état present
11          PROCESS(etat_present) BEGIN
12            CASE etat_present is --style case when
13              WHEN "00" => etat_futur <="01"; -- premiere transition
14              WHEN "01" => etat_futur <="10"; -- deuxieme transition
15              WHEN "10" => etat_futur <="11"; -- troisieme transition
16              WHEN OTHERS => etat_futur <="00";
17            END CASE;
18          END PROCESS;
19          -- le futur devient le present
20          PROCESS(clk) BEGIN
21            IF rising_edge(clk) THEN
22              etat_present <= etat_futur;
23            END IF;
24          END PROCESS;
25          -- sans arrêt
26          q<= etat_present
27        END acmpt;
```

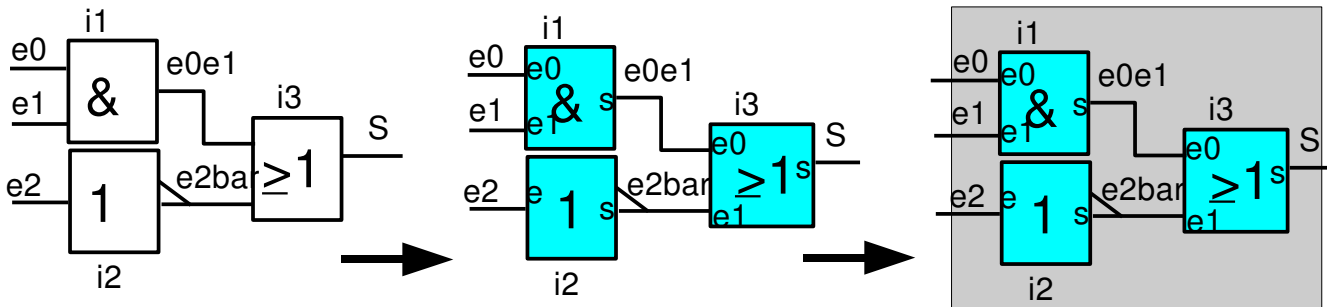
Exo7 : Refaire le même travail que l'exo6 en utilisant un style deux "process" et en gérant l'initialisation. Votre programme aura trois process en fait puisqu'il faut ajouter la division de fréquence.

TP5 Schématique et VHDL

1. Association de composants

Dans cette section, nous allons apprendre à utiliser le style de programmation structurée, c'est à dire à comprendre comment associer des composants ensemble.

Commencez par comprendre la notion de hiérarchie :

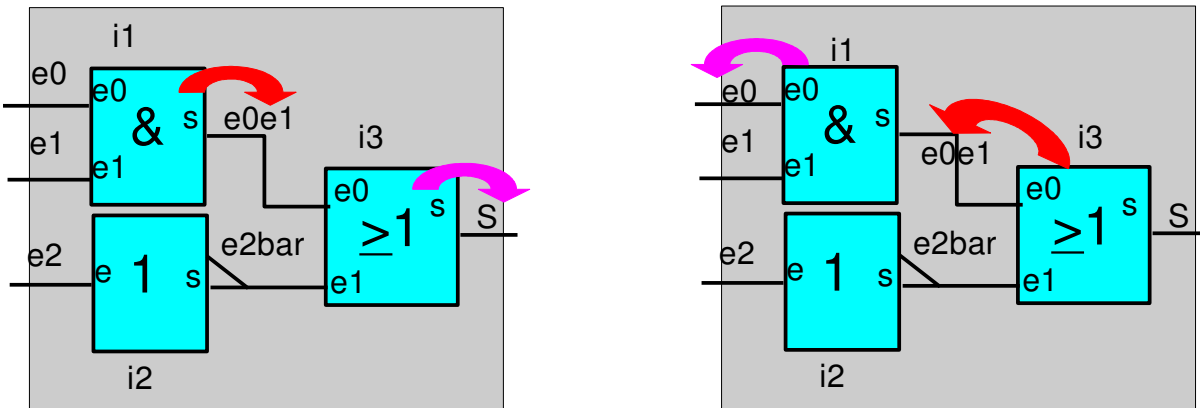


Dans la figure de droite vous avez quatre rectangles. Il y aura donc dans votre projet VHDL quatre entités et quatre architectures. Il y a un rectangle gris qui contient trois rectangles bleus.

Passer du temps à comprendre la figure ci-dessous : pour les "port map".

Voir aussi pour les explications :

http://fr.wikibooks.org/wiki/TD1_VHDL#Assembler_des_composants_en_VHDL



i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);

i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>S);

Travail à réaliser

Exo1 : reprendre le problème du TP précédent (**exo4**) en séparant le diviseur de fréquence et le compteur à sortie directe et en regroupant le tout dans un super composant.

Remarque : la décomposition du problème précédent en deux composants peut sembler, à juste titre, inutile. Il existe cependant des cas où elle ne peut absolument pas être évitée. Un exemple important est celui des mémoires que nous allons aborder en section 3.

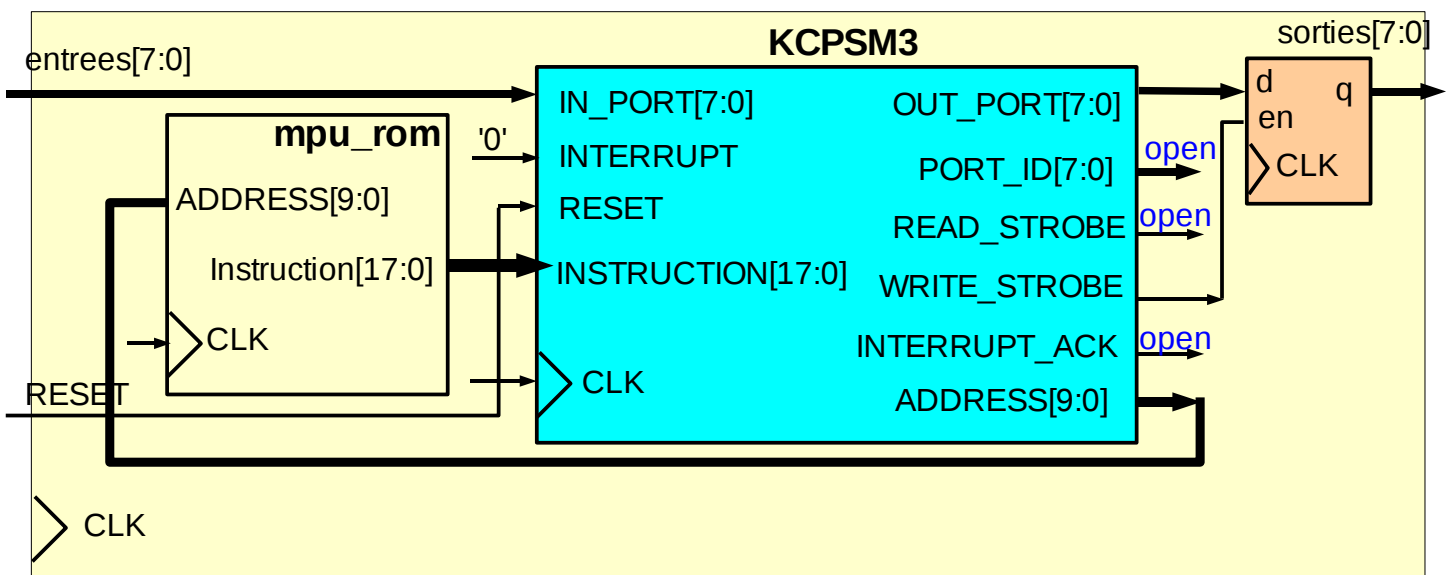
2. Ajouter un processeur avec un couplage faible

Le développement des produits complexes modernes se fait avec un processeur qui peut être embarqué dans le FPGA ou externe. Le couplage entre le processeur et la logique externe (appelée généralement périphérique dans la suite) peut être de deux types :

- faible : le processeur peut être facilement remplacé par autre chose (dans notre cas un simple compteur pourrait le remplacer),
- fort : le périphérique est incapable de fonctionner seul. L'exemple typique est la carte vidéo : une carte vidéo sans processeur est complètement inutile !

LO11 a comme objectif de vous amener petit à petit aux couplages forts (peut être sans y parvenir complètement ?)

Voici le schéma de base du processeur que nous allons utiliser, sans entrer dans les détails :



La mise en œuvre est assez simple puisque l'on vous fournit mpu_rom qui est la mémoire programme, KCPSM3 qui est le processeur picoBlaze.

Soit en VHDL (avec 2 composants seulement) :

```

1      library IEEE;
2      use IEEE.STD_LOGIC_1164.ALL;
3      entity tp5exo1 is
4          port (
5              clk, reset : in std_logic;
6              entrees : in std_logic_vector(7 downto 0);
7              sorties : out std_logic_vector(7 downto 0)
8          );
9      end tp5exo1;
10
11     architecture atp5 of tp5exo1 is
12         component kcpsm3
13             Port (
14                 address : out std_logic_vector(9 downto 0);
15                 instruction : in std_logic_vector(17 downto 0);
16                 port_id : out std_logic_vector(7 downto 0);
17                 write_strobe : out std_logic;
18                 out_port : out std_logic_vector(7 downto 0);
19                 read_strobe : out std_logic;
20                 in_port : in std_logic_vector(7 downto 0);

```

```

20         interrupt : in std_logic;
21         interrupt_ack : out std_logic;
22         reset : in std_logic;
23         clk : in std_logic);
24     end component;
25     component mpu_rom
26     Port (
27         address : in std_logic_vector(9 downto 0);
28         instruction : out std_logic_vector(17 downto 0);
29         clk : in std_logic);
30
31     signal s_sorties, s_sorties2 : std_logic_vector(7 downto 0);
32     signal s_write_strobe : std_logic;
33     signal s_address : std_logic_vector(9 downto 0);
34     signal s_instruction : std_logic_vector(17 downto 0);
35     begin
36         i1:kcpsm3 port map(address => s_address,
37             instruction => s_instruction,
38             port_id => open,
39             in_port => entrees,
40             out_port => s_sorties,
41             read_strobe => open,
42             write_strobe => s_write_strobe,
43             interrupt =>'0',
44             interrupt_ack => open,
45             reset => reset,
46             clk => clk);
47         i2: mpu_rom port map(address => s_address,
48             instruction => s_instruction,
49             clk => clk);
50         -- mémorisation sorties
51         process (clk) begin
52             if rising_edge(clk) then
53                 if s_write_strobe = '1' then
54                     s_sorties2 <= s_sorties;
55                 end if;
56             end if;
57         end process;
58         sorties <= s_sorties2;
59     end atp5;

```

Travail à réaliser

Exo2 : Réaliser le projet et assembler le tout pour sortir sur des LEDs et montrer que ce processeur réalise un ensemble de signaux de fréquences qui doublent chaque fois que l'on se déplace d'un cran vers les poids faible. La fréquence du poids fort doit être de l'ordre du Hertz. Fixez "entrées" à x"B8" pour un bon fonctionnement.

Les deux composants essentiels kcpsm3 et mpu_rom vous seront donnés.

Nous utiliserons ce bloc très souvent dans la suite de ces TP. Il faut donc le mettre de côté et être capable de le retrouver. Il nous arrivera de changer **mpu_rom** mais nous le signalerons alors. Il vous arrivera même (mais plus tard) de réaliser **mpu_rom**.

3.Compteur à sortie sept segments avec mémoire

Nous allons utiliser un compteur normal, c'est à dire qui compte en binaire sur 4 bits suivi d'une mémoire pour le transcodage. L'horloge de ce compteur sera réalisée par le poids faible de sorties du processeur de la section 2.

Les mémoires sont difficiles à utiliser en VHDL car la façon pour les implémenter dépend du fondeur de FPGA. Ainsi la façon de faire présentée dans cette section n'est vraie que pour les FPGA Xilinx.

Si vous voulez que votre outil de synthèse infère une RAM/ROM il vous faut écrire votre RAM/ROM avec un style spécifique. Nous présentons ici un exemple sur lequel nous nous appuyerons dans la suite : ROM avec sortie Registered

```

1      --
2      -- ROMs Using Block RAM Resources.
3      -- VHDL code for a ROM with registered output (template 1)
4      --
5      library ieee;
6      use ieee.std_logic_1164.all;
7      use ieee.std_logic_unsigned.all;
8      entity rams_21a is
9          port (clk : in std_logic;
10             en   : in std_logic;
11             addr : in std_logic_vector(5 downto 0);
12             data : out std_logic_vector(19 downto 0));
13     end rams_21a;
14     architecture syn of rams_21a is
15         type rom_type is array (63 downto 0) of std_logic_vector (19 downto 0);
16         signal ROM : rom_type:=
17             (X"0200A", X"00300", X"08101", X"04000", X"08601",   X"0233A",
18             X"00300", X"08602", X"02310", X"0203B", X"08300", X"04002",
19             X"08201", X"00500", X"04001", X"02500", X"00340", X"00241",
20             X"04002", X"08300", X"08201", X"00500", X"08101", X"00602",
21             X"04003", X"0241E", X"00301", X"00102", X"02122", X"02021",
22             X"00301", X"00102", X"02222", X"04001", X"00342", X"0232B",
23             X"00900", X"00302", X"00102", X"04002", X"00900", X"08201",
24             X"02023", X"00303", X"02433", X"00301", X"04004", X"00301",
25             X"00102", X"02137", X"02036", X"00301", X"00102", X"02237",
26             X"04004", X"00304", X"04040", X"02500", X"02500", X"02500",
27             X"0030D", X"02341", X"08201", X"0400D");
28     begin
29         process (clk)
30         begin
31             if (clk'event and clk = '1') then
32                 if (en = '1') then
33                     data <= ROM(conv_integer(addr));
34                 end if;
35             end if;
36         end process;
37     end syn;

```

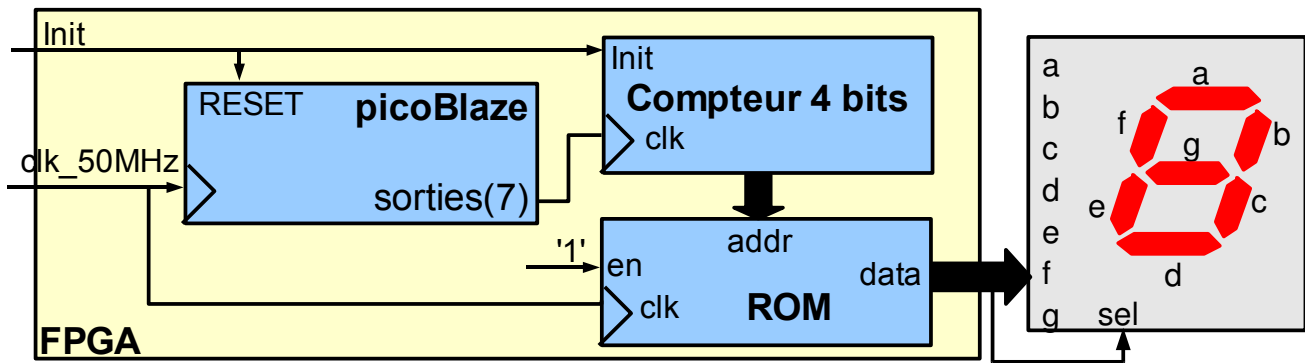
Travail à réaliser

Exo3 : dimensionner la ROM nécessaire à votre problème. Calculer les valeurs hexadécimales contenues dans cette ROM.

Réaliser un compteur 4 bits avec initialisation asynchrone.

Reprendre le diviseur de fréquence de la section 2

Assembler le tout

**Remarque :**

Votre fichier rapport de compilation doit faire apparaître une ROM : dans "detailed reports" chercher "Synthesizing Unit <rams_21a>", et vous verrez :

```
WARNING:Xst:1781 - Signal <ROM> is used but never assigned. Tied to default value.
Found 16x8-bit ROM for signal <$varindex0000> created at line 62.
Found 8-bit register for signal <data>.
Summary:
inferred 1 ROM(s).
inferred 8 D-type flip-flop(s).
Unit <rams_21a> synthesized.
```

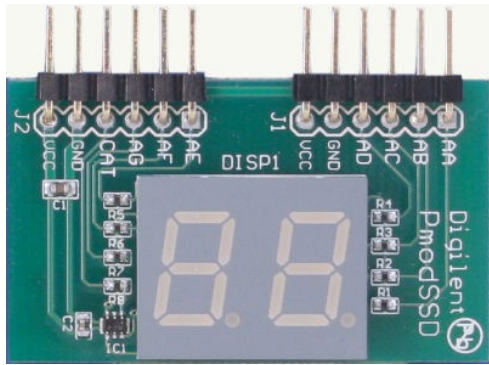
En fait le fait qu'il détecte une ROM à ce point n'est pas suffisant pour être sûr qu'il l'implante tel quel. Dans la fenêtre "Design", il faut bien cliquer bouton droit sur "Synthesize -XST". Mais ensuite il faut sélectionner "Process properties" (et pas "design goals and strategies"). Là il faut sélectionner "HDL options" dans la "fenêtre" "Category" . Et ensuite cocher "rom extract" et pour "rom style" sélectionner "block". On peut voir cela en démarrant PlanAhead après le placement routage complet de l'application (avec un .UCF) depuis la même fenêtre "Design" en développant l'arborescence de "Place and Route" et en lançant "Analyze Timing / Floorplan Design (PlanAhead)".

Dans PlanAhead, on ne voit pas forcément le chevelu et les blocs utilisés par l'application. Il faut être sur l'onglet "Device" et cliquer sur les 3 icônes (bande verticale d'icônes à côté du dessin du circuit) "show/hide loc constraints" "show/hide bundle nets" et "show/hide I/O nets".

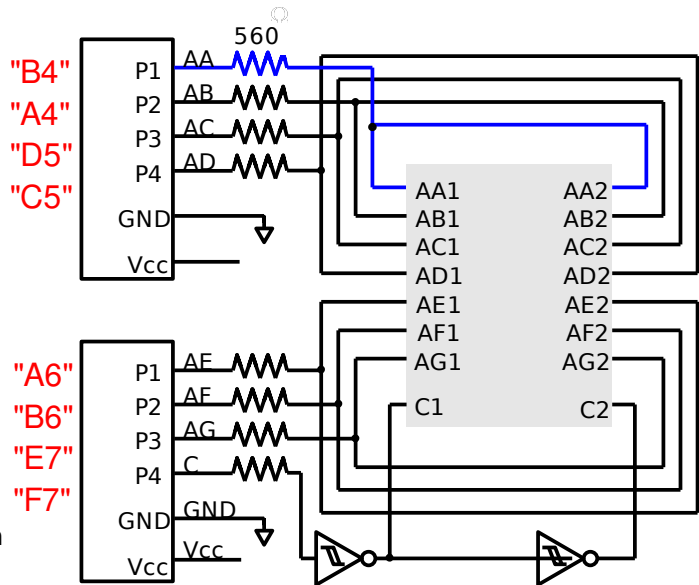
Nous ne changerons pas les options de compilation dans la suite et utiliserons une autre méthode pour le forcer à utiliser les mémoires internes.

4.Compteur à sortie sept segments sur deux digits

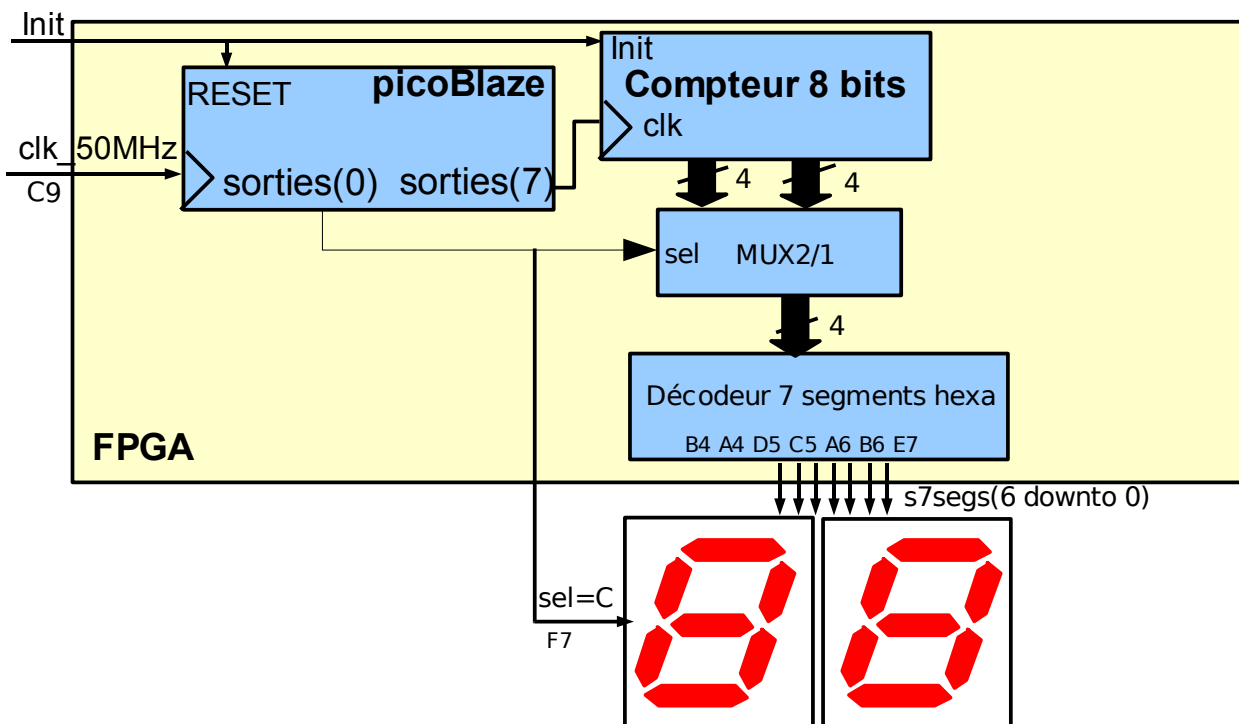
Cette partie doit être réalisée avec les afficheurs sept segments reliés à la carte. Vous êtes assez avancé maintenant pour comprendre la documentation de l'afficheur.



Seven-Segment Display Connection Diagram



La broche de sélection de l'afficheur notée "sel" en tp1 est notée "C" ici dans la documentation officielle. On a représentée en bleu la connexion de P1 (située en broche "B4") vers les deux segments a. C'est la même ! Cela veut dire que si l'on veut afficher deux valeurs différentes il faut les envoyer alternativement : c'est le rôle du multiplexeur qui est commandé par Q(19). Si un clignotement est présent sur l'afficheur, on prendre Q(18) voir Q(17) au lieu de Q(19). Il est important de comprendre pourquoi.



Travail à réaliser

Exo4 : Implanter l'ensemble présenté ci-dessus. Le décodeur sept segments sera au choix, la mémoire précédente ou le décodeur demandé comme premier exercice de ce TP. Le résultat doit être un affichage de 00 à FF... Attention, le poids faible doit être à droite !!!

Indication

Une aide pour implanter un multiplexeur peut être trouvée dans le polycopié de cours p119 ou ici : http://fr.wikiversity.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language/Exercices/TP_1#D.C3.A9codeur_8_bits_vers_deux_afficheurs_avec_sa_correction

5.Compteur BCD sur deux digits

Nous cherchons maintenant à modifier le compteur du TP précédent (exo3) pour qu'il affiche de 00 à 99.

Travail à réaliser

Exo5 : Modifier le compteur 8 bits en le décomposant en deux compteurs BCD (0 à 9 à 0 ...) cascadables.

Indication

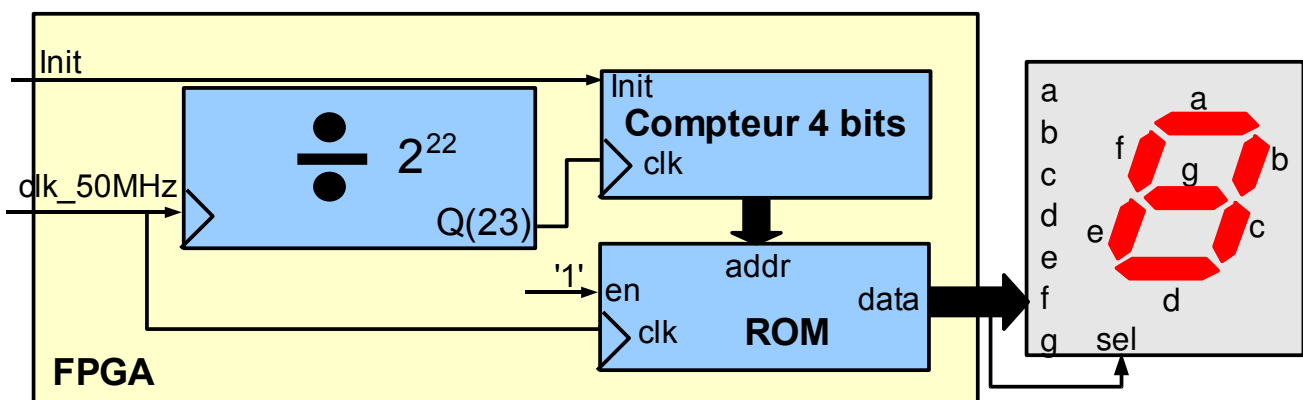
Une aide pour réaliser un compteur BCD cascadable peut être trouvée ici :

http://fr.wikibooks.org/wiki/TD3_VHDL_Compteurs_et_registres#Compteur_BCD_cascadable

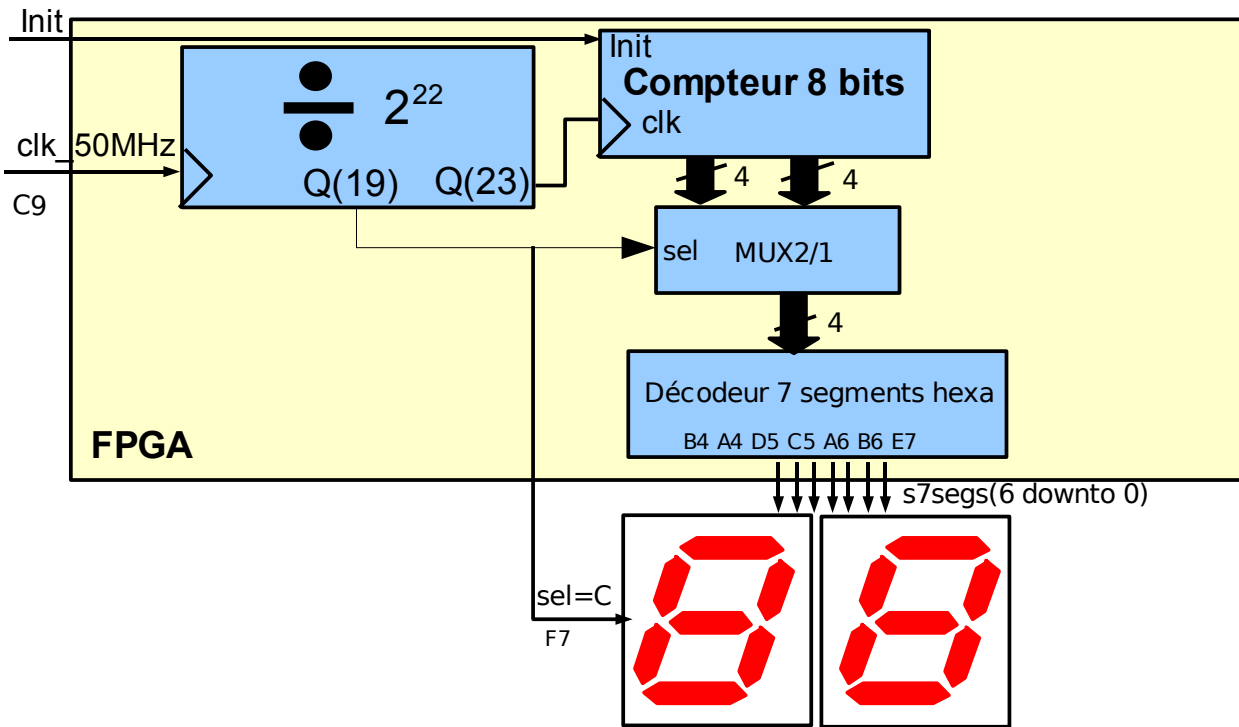
6.ANNEXES du TP5

Nous donnons pour information comment étaient réalisées les horloges lentes pour le LO11 du printemps 2011.

Exo 3 : l'exercice 3 sans processeur peut se réaliser ainsi : la fréquence lente est obtenue avec un simple compteur :



Exo 4 : l'exercice 4 sans processeur quant à lui peut être simplement réalisé par le schéma ci-dessous.



Ces versions avec compteur simple seront reprises dans le TP 7 pour lequel nous travaillerons sans processeur.

Nous donnons aussi pour information le programme assembleur du picoBlaze pour cette année :

```

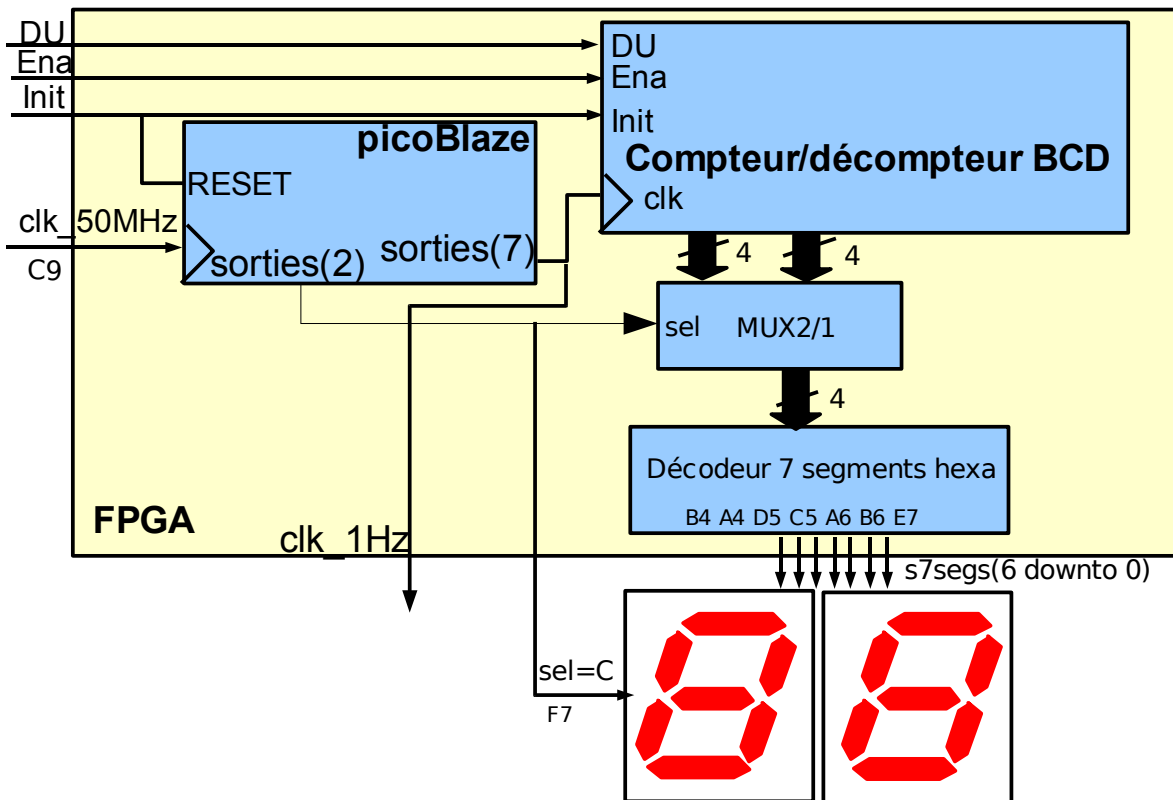
1      ; LO11 2012 tp5exo2
2      ;constant MAX,b8      ;; pour avoir 1 hz avec double boucle d'attente
3      ; NAMEREG s0,index   ; rename register s0 as index
4      NAMEREG s7,i
5      NAMEREG s6,j
6      NAMEREG s4,led
7      NAMEREG s0,max
8
9
10     LOAD led,00
11     loop:  OUTPUT    led,00
12           CALL    wait
13           ADD     led,01
14           jump   loop
15     ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
16     ;=== double boucle d'attente===
17     wait:  INPUT   max,00
18           LOAD   i,max
19
20     w:
21           LOAD   j,max
22     w_1:
23           SUB    j,01
24           JUMP  NZ, w_1
25           SUB   i,01
26           JUMP  NZ,w
           RETURN

```

TP6 Les compteurs BCD et applications en VHDL

1. Compteur/décompteur BCD sur deux digits

Nous cherchons maintenant à modifier le compteur du TP précédent pour qu'il soit capable, à l'aide d'une entrée supplémentaire (DU = Down/Up), de décompter ou de compter en BCD.



Travail à réaliser

Exo1 : Modifier le compteur 4 bits du TP précédent pour qu'il soit capable de compter et de décompter et puisse être cascadié.

```

1      library ieee;
2      use ieee.std_logic_1164.all;
3      use IEEE.STD_LOGIC_ARITH.ALL;
4      use IEEE.STD_LOGIC_UNSIGNED.ALL;
5      -- compteur exo5 TP 5
6      entity compteur is port (
7          clk :in std_logic;
8          init,en : in std_logic;
9          enout: out std_logic;
10         s: out std_logic_vector(3 downto 0)
11     );
12     end entity;
13
14     architecture behavior of compteur is
15         signal n : std_logic_vector(3 downto 0);
16     begin
17         increment : process(clk,init) begin
18             if init ='1' then

```

```

19         n <= (others => '0');
20     elsif clk'event and clk='1' then
21         if en='0' then
22             if n<9 then
23                 n <= n + 1 ;
24             else
25                 n<="0000";
26             end if;
27         end if;
28     end if;
29 end process;
30 s <= n;
31 process(n,en) begin --gestion de sortie cascadable
32     if en='0' and n=9 then
33         enout <= '0';
34     else
35         enout <='1';
36     end if;
37 end process;
38 end behavior;

```

Le mettre dans l'ensemble ci-dessus pour un test.

Je mémorise :

Apprenez par cœur ou sachez retrouver cette façon de gérer un "reset" et un "ena" (parfois aussi appelé "en" pour enable). D'abord avec un "reset" synchrone :

```

1     architecture behavior of compteurbcd is
2         signal n : std_logic_vector(3 downto 0);
3     begin
4         increment : process(clk) begin
5             if clk'event and clk='1' then
6                 if reset = '1' then
7                     n <= (others => '0');
8                 elsif en='0' then
9                     if ud = '1' then --up
10                        if n<9 then
11                            n <= n + 1 ;
12                        else
13                            n <= (others => '0');
14                        end if;
15                    else -- down

```

et maintenant avec un "reset" asynchrone :

```

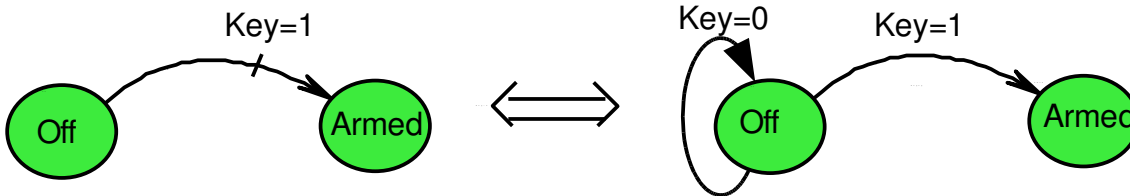
1     architecture Behavioral of Counter2_VHDL is
2         signal temp: std_logic_vector(3 downto 0);
3     begin
4         process (Clock,Reset) begin
5             if Reset='1' then
6                 temp <= "0000";
7             elsif (Clock'event and Clock='1') then
8                 if en='0' then
9                     if temp="1001" then
10                        temp<="0000";
11                 else
12                     temp <= temp + 1;

```

Quand tout fonctionne correctement **mettre de côté pour une utilisation dans l'exercice suivant**

2. Du graphe d'évolution au graphe d'états (pour information)

Nous avons abordé les graphes d'évolution pour nous intéresser maintenant aux graphes d'états.



La transition devient réceptive, c'est à dire qu'elle décrit ce qui se passe quand la réceptivité est vraie et quand la réceptivité est fausse (donc suppression des oreilles de Mickey)

Préparation

Relire le cours p136 - 143 sur la programmation d'une machine à états pour essayer de comprendre comment programmer un graphe d'états.

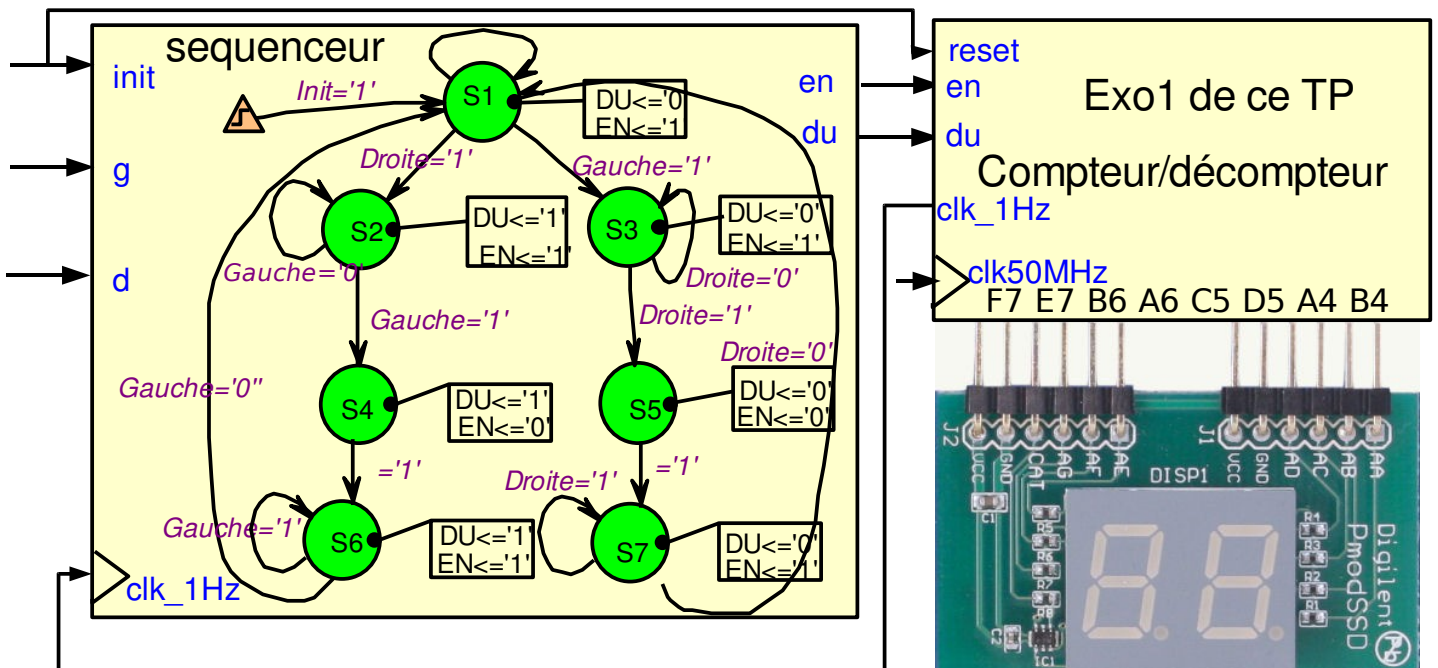
3. Le compteur de passages

Un ensemble de deux capteurs infrarouge perçoit le passage d'une personne et en fonction de son sens de passage incrémente ou décrémente un compteur qui décodé affiche le nombre de personnes sur deux afficheurs 7 segments.

Il nous faudra comme composants :

- deux compteurs-décompteurs décimaux (cascadés) ainsi que la logique d'affichage. C'est ce que l'on a fait dans l'exercice **exo1** précédent.
- une logique de contrôle pour détecter le sens de passage

Seule la logique de contrôle est spécifiée à l'aide d'un graphe d'évolution.



Remarques : En cours p 139, les actions sont notées dans les états. Il n'y a aucune différence entre les deux notations. On pourrait utiliser le picoBlaze en lieu et place du graphe d'évolution (section 5).

Préparation

Que se passe-t-il si Droite et gauche arrivent en même temps ? Modifier le graphe d'évolution pour gérer seulement le cas **Droite='1'&Gauche='0'** d'une part et **Gauche='1'&Droite='0'** d'autre part.

(Optionnel) Chercher le graphe d'états correspondant au problème à partir du graphe d'évolution donné.

(Optionnel) Écrire les équations de récurrence faisant intervenir l'entrée d'initialisation "Init" si vous désirez les utiliser, autrement passer directement à la section suivante.

Travail à réaliser

Exo 2 :

Programmer directement avec un style "case when" votre séquenceur.

On vous demande d'implanter le graphe d'états en lui ajoutant l'ensemble compteur décompteur de l'exo 1 précédent (et le processeur pour l'horloge lente).

L'utilisation des boutons d'USBMOD pour éviter les rebonds est optionnelle.

Indication :

Voici en condensé comment on programme un graphe d'évolution en style deux process :

```

1      -- cette partie dépend du graphe d'état : elle gère les transitions
2      combinatoire: process(etat,d,g) begin
3          case etat is
4              when s1 => if (d='1' and g='0' ) then
5                          next_etat <= s2;
6                          elsif (g='1' and d='0' ) then
7                              next_etat <= s3;
8                          else
9                              next_etat <= s1;
10                         end if;
11                 when s2 => --.....
12                 --.....
13                 when others => next_etat <= s1;
14             end case;
15         end process;
16         -- cette partie ne change pas tant que reset est asynchrone
17         sequenceur: process(clk,reset) begin
18             if reset = '1' then
19                 etat <= s1;
20             elsif rising_edge(clk) then
21                 etat <= next_etat;
22             end if;
23         end process;
24         -- cette partie gère les sorties avec "when else" ou "with select when"
25         en <= '1' when etat=s4 else
26             '1' when etat=s5 else
27             '0';

```

4.Un compteur de passages un peu plus sophistiqué

On cherche maintenant à réaliser ce même séquenceur mais en donnant la possibilité aux

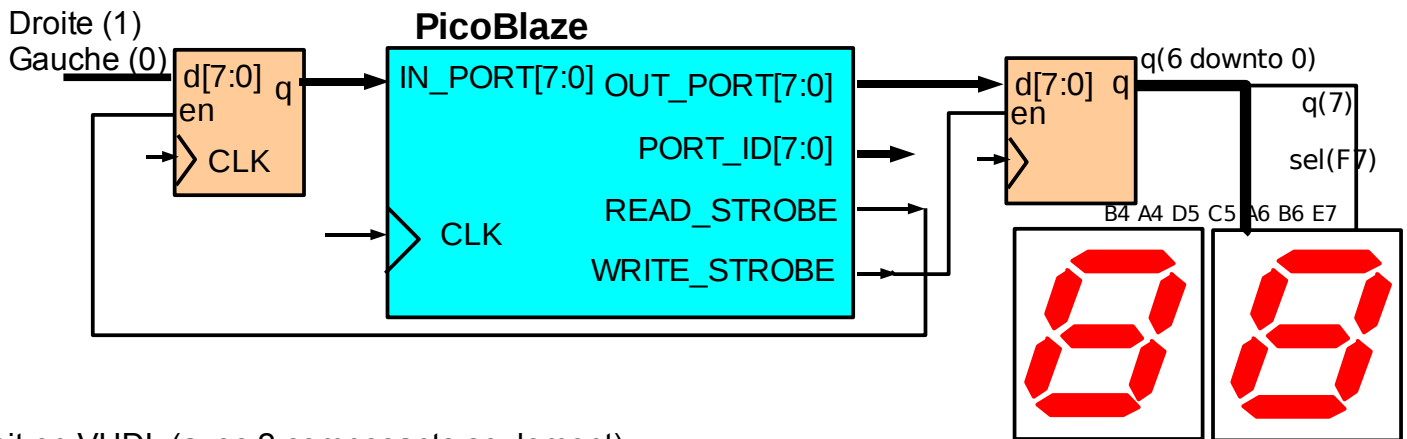
personnes de faire demi-tour entre les deux capteurs (c'est à dire de changer d'avis).

Travail à réaliser

Exo3 : Montrer qu'un graphe d'état de 9 états est nécessaire pour gérer ce problème. Écrire les équations de récurrence ou implanter avec un "case when" puis tester.

5.Un compteur de passage dans le picoBlaze

Il est naturellement possible de ne pas cantonner le picoBlaze au rôle de générateur d'horloge mais en faire le maître de ce montage : réaliser le comptage, le séquenceur et la gestion des afficheurs et même la gestion des rebonds. L'ajout du port d'entrée n'est pas obligatoire.



Soit en VHDL (avec 2 composants seulement) :

```

1      library IEEE;
2      use IEEE.STD_LOGIC_1164.ALL;
3      entity tp6exo4 is
4          port (
5              clk,reset : in std_logic;
6              entrees : in std_logic_vector(7 downto 0);
7              sorties : out std_logic_vector(7 downto 0)
8          );
9      end tp6exo4;
10
11     architecture atp6 of tp6exo4 is
12         component kcpsm3
13             Port (
14                 address : out std_logic_vector(9 downto 0);
15                 instruction : in std_logic_vector(17 downto 0);
16                 port_id : out std_logic_vector(7 downto 0);
17                 write_strobe : out std_logic;
18                 out_port : out std_logic_vector(7 downto 0);
19                 read_strobe : out std_logic;
20                 in_port : in std_logic_vector(7 downto 0);
21                 interrupt : in std_logic;
22                 interrupt_ack : out std_logic;
23                 reset : in std_logic;
24                 clk : in std_logic);
25         end component;
26         component mpu_rom
27             Port (
28                 address : in std_logic_vector(9 downto 0);
29                 instruction : out std_logic_vector(17 downto 0);
30                 clk : in std_logic);
31         end component;

```

```
30
31     signal s_sorties, s_sorties2, s_entrees: std_logic_vector(7 downto 0);
32     signal s_write_strobe,s_read_strobe : std_logic;
33     signal s_address : std_logic_vector(9 downto 0);
34     signal s_instruction : std_logic_vector(17 downto 0);
35     begin
36         i1:kcpsm3 port map(address => s_address,
37             instruction => s_instruction,
38             port_id => open,
39             in_port => s_entrees,
40             out_port => s_sorties,
41             read_strobe => s_read_strobe,
42             write_strobe => s_write_strobe,
43             interrupt =>'0',
44             interrupt_ack => open,
45             reset => reset,
46             clk => clk);
47         i2: mpu_rom port map(address => s_address,
48             instruction => s_instruction,
49             clk => clk);
50         -- mémorisation des entrées
51         process(clk) begin
52             if rising_edge(clk) then
53                 if s_read_strobe = '1' then
54                     s_entrees <= entrees;
55                 end if;
56             end if;
57         end process;
58         -- mémorisation sorties
59         process(clk) begin
60             if rising_edge(clk) then
61                 if s_write_strobe = '1' then
62                     s_sorties2 <= s_sorties;
63                 end if;
64             end if;
65         end process;
66         sorties <= s_sorties2;
67     end atp6;
```

TP7 Le compteur de passages avec sortie RS232

On reprend le TP précédent (ce qui permettra d'en faire une correction **sans picoBlaze**) et on lui ajoutera un élément qui envoie la valeur du compteur sur la RS232. On va construire de toute pièce l'ensemble permettant l'envoi sur une liaison série pour le recevoir dans "gtkterm".

Ressource UTTLO11.zip de mon site dans le répertoire /TP7 fichier median.vhd

1. Introduction (tirée d'Internet)

Les liaisons séries asynchrones sont très utilisées. La plus connue est celle qui est utilisée sur les PC. Asynchrone signifie que les données sont envoyées de l'émetteur vers le récepteur sans négociation préalable. C'est au récepteur de se synchroniser sur l'émetteur. Pour ce faire l'émetteur doit envoyer un bit de START ses données (de 5 à 8 bits) suivies ou non d'un bit de parité et de 1 ou plusieurs bits de stop. Pour qu'une liaison série fonctionne, il est nécessaire de configurer les 2 extrémités pour qu'elles utilisent la même parité, le même nombre de bits de stop (1, 1,5 ou 2) la longueur des données échangées (5, 6, 7, ou 8 bits).

La norme RS232 définit les valeurs des tensions que doivent fournir et reconnaître les interfaces séries des matérielles.

- Un 0 logique est reconnu pour une tension allant de +8 à +40V.
- Un 1 logique est reconnu pour une tension allant de -8 à -40V.

Généralement, les signaux envoyés sont compris entre -12 et + 12 V.

Sur une liaison série au repos on doit observer un 1 logique.

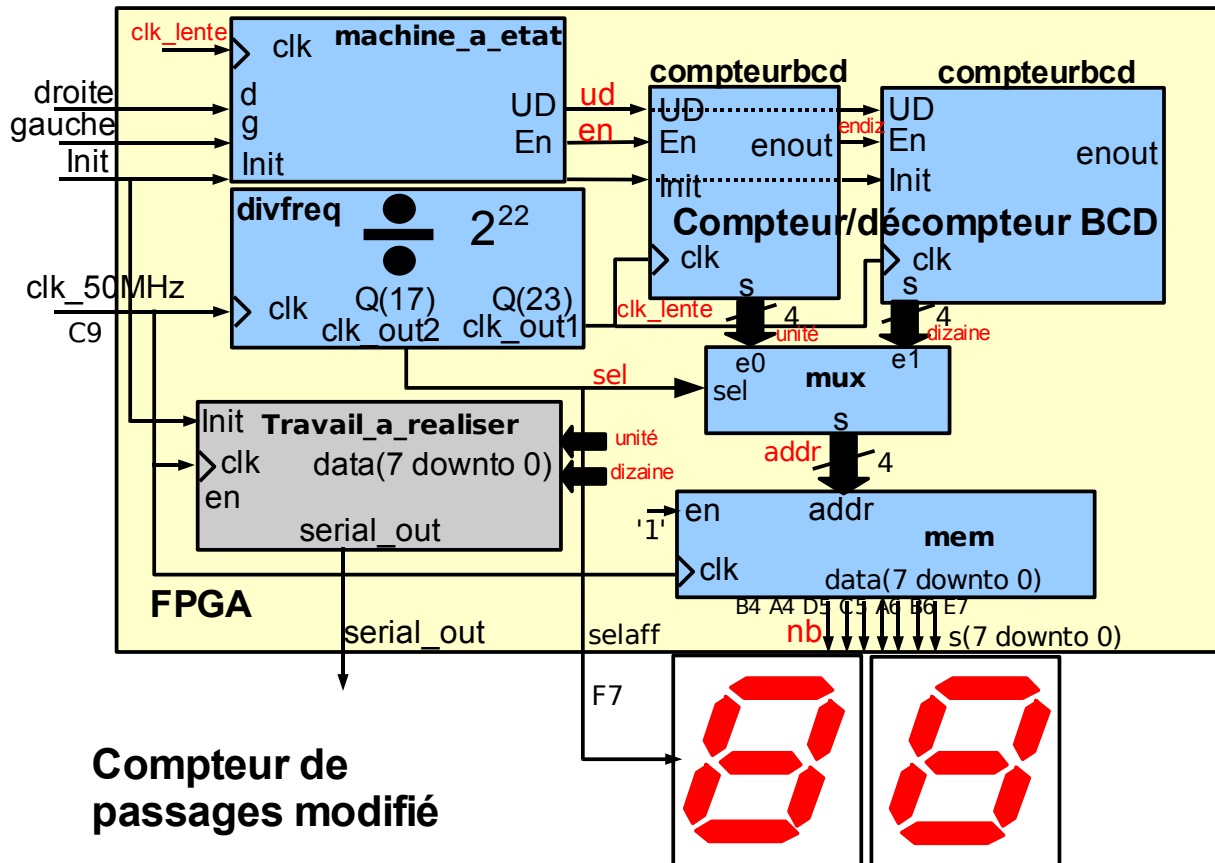
Pour faire un échange de données bidirectionnel entre 2 liaisons séries il faut au minimum 3 fils.

- Un pour les données qui circulent dans un sens.
- Un pour les données qui circulent dans l'autre sens.
- Un pour la masse électrique des signaux.

Cette liaison à 3 fils est une liaison minimum. Elle nécessite une collaboration logicielle active entre les 2 machines pour contrôler le transfert des informations. Un mécanisme souvent utilisé est le protocole XON XOFF.

2. Rappel : compteur de passages

La correction du compteur de passages a été utilisée pour un médian. Elle est disponible sur mon site personnel (ressource UTTLO11.zip de mon site dans le répertoire /TP7 fichier median.vhd). Nous en rappelons le schéma légèrement modifié pour les besoins de ce TP.



Nous allons ajouter un bloc complet (bloc gris du dessin) à cette solution pour pouvoir envoyer le résultat du compteur sur une liaison série reliée à votre ordinateur. Le logiciel utilisé sur votre ordinateur est gtkterm (qui est l'équivalent de l'hyperterminal Windows)

3. Construction du générateur de baud

La liaison série est une liaison sans horloge. Elle est cependant caractérisée par une vitesse de transmission.

Nous désirons transmettre à 19200 bauds. Pour cela il nous faut réaliser un signal 16 fois plus rapide que la fréquence correspondante soit 307200 Hz.

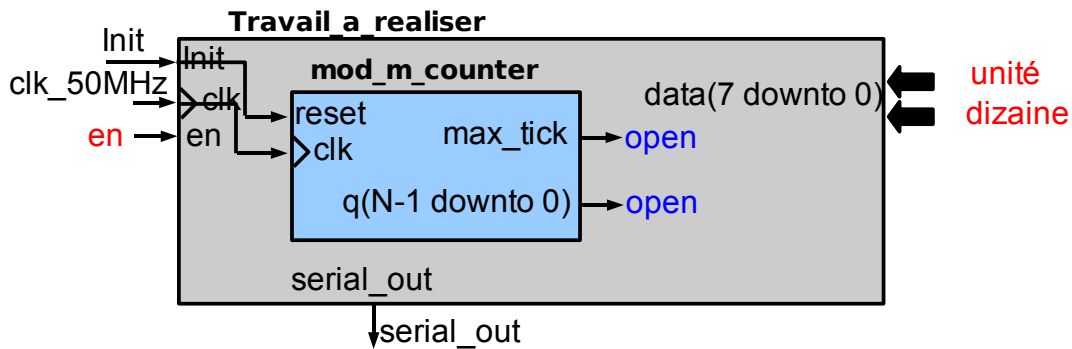
Préparation

On désire réaliser la fréquence précédente à partir du 50MHz en détectant une valeur fixe (maximale) d'un compteur. Quelle doit être la valeur maximale ? Combien de bit nécessite ce compteur (c'est la valeur de N qui sera déterminée par un [generic map](#)).

Travail à réaliser

On désire implanter ce générateur de baud à l'aide d'un compteur présenté dans le livre de P. P. CHU "FPGA prototyping By VHDL examples" en page 83, disponible sur mon site dans UTTLO11.zip en fin du fichier median.vhd du répertoire /TP7.

On présente sous forme schématique le travail à réaliser **et à insérer dans le compteur de passages** comme indiqué sur la première figure :



On rappelle qu'il est possible d'utiliser le mot clé "open" du VHDL quand on désire laisser des sorties ouvertes (non reliées) pour que le "port map" énumère tous les fils du composant en cours de câblage, et que les noms notés en rouge sont des signaux.

Pour tester utiliser un scope si vous avez la connectique correspondante. Autrement, ce que l'on vous demande c'est de faire constater que vous avez réalisé un programme qui est conforme au dessin et qui se compile. Si vous voulez le télécharger dans votre carte pour constater qu'il fonctionne toujours comme un compteur de passages vous pouvez naturellement le faire.

Indication : nous vous proposons de réaliser l'entité et l'architecture "travail_a_realiser" dans un fichier séparé qu'il faudra ajouter au projet. Le câblage se fera dans le corrigé du compteur de passages. Ce câblage variera peu au fur et à mesure de l'avancement de ce TP (mais quand même un peu). Vous pouvez relier max_tick à serial_out pour ce test mais il vous faudra le retirer immédiatement pour la suite !

4. Construction de la liaison série

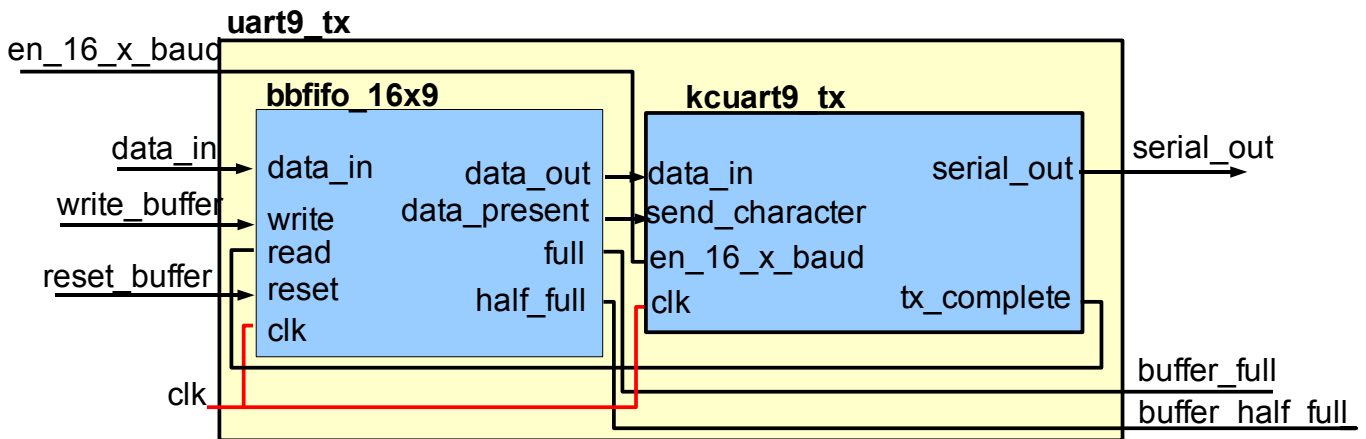
Présentation

On désire construire une liaison série qui va envoyer une donnée fixe au rythme de la modification du compteur (des unités). Le signal "en" ci-dessus (dans la première figure de ce TP) est un bon candidat pour lancer l'envoi dans le buffer mais comme vous allez vite vous en apercevoir, il dure trop longtemps.

La partie transmission de la liaison série est réalisée de manière standard par ce que Xilinx appelle des macros et que je préfère appeler des **logicores** qui sont des composants gratuits fournis par Xilinx (contrairement aux alliancecores qui sont payants)

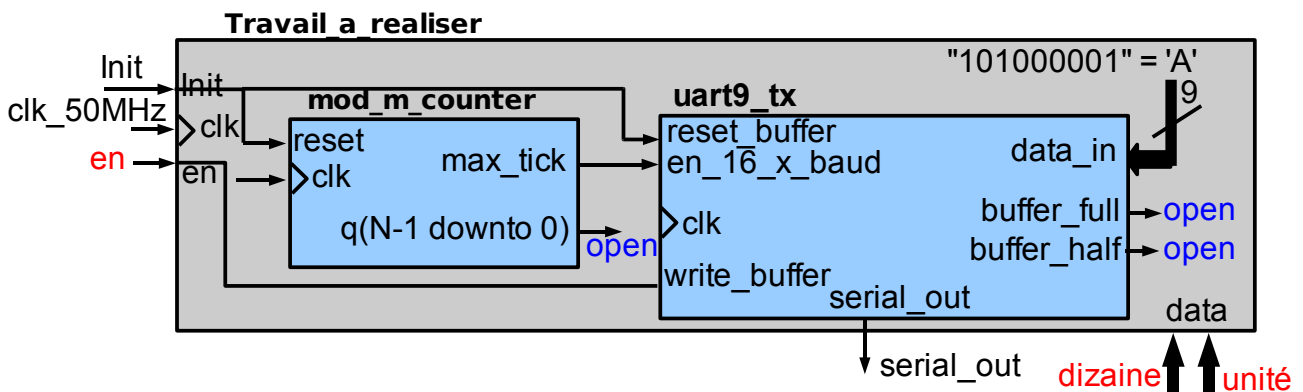
Travail à réaliser

On vous donne l'ensemble matériel qui réalise la partie transmission de la liaison série.



Le schéma ci-dessus fait apparaître 3 rectangles, il est donc composé de trois entités (dont les noms sont en gras) et de trois architectures. La partie gauche est un buffer de 16 données de 9 bits : **si vous insérez les trois fichiers dans votre projet, la connexion comme indiquée dans le schéma ci-dessus est toute faite**. On peut trouver ces fichiers dans **UTTLO11.zip** téléchargeable dans le site perso de S.Moutou.

Insérer ce logicore dans le composant travail à réaliser comme indiqué dans le schéma ci-dessous :



On reviendra plus tard sur le pourquoi des neuf bits et donc pourquoi le code ASCII de 'A' qui est "01000001" (sur 8 bits) se transforme en "101000001"(sur 9 bits).

Tester avec gtkterm configuré comme ci-dessous (à part peut-être le PORT) :

PORT :/dev/ttyS1, vitesse : 19200, Parité : odd, Bits 8, Bit de stop : 1, contrôle de flux : none

Vous verrez apparaître une série de 'A' et non pas un seul 'A' ! Trouver pourquoi. Faire constater.

N'oubliez pas de définir serial_out dans votre fichier ucf : un des signaux ci-dessous :

```
# ==== RS-232 Serial Ports (RS232) ====
#NET "RS232_DCE_RXD" LOC = "R7" | IOSTANDARD = LVTTTL ;
#NET "RS232_DCE_TXD" LOC = "M14" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = SLOW ;
#NET "RS232_DTE_RXD" LOC = "U8" | IOSTANDARD = LVTTTL ;
#NET "RS232_DTE_TXD" LOC = "M13" | IOSTANDARD = LVTTTL | DRIVE = 8 | SLEW = SLOW ;
```

Indication : Pour relier 2 équipements via une liaison série la norme RS232 prévoit 2 brochages

différents (DTE et DCE). Le brochage type DTE (Data Terminal Equipment) doit être utilisé pour des équipements terminaux. Le brochage type DCE (Data Control Equipment) est normalement utilisé pour des équipements intermédiaires utilisés sur des liaisons (modems,...). Le brochage DTE normalisé est, celui décrit sur les connecteurs à 25 points des PC. Le brochage DCE est très simple, sur le connecteur DB25 de l'équipement on retrouve les mêmes signaux en les croisant.

5. Amélioration de l'ensemble avec un séquenceur

Pour remédier aux problèmes précédents, on va utiliser un séquenceur (ou machine à états). Trois octets seront envoyés à l'hyperterminal : le chiffre des dizaines, le chiffre des unités et un retour chariot.

- dès que "en" je prépare les trois octets à envoyer et remplit le buffer au fur et à mesure.

Le séquenceur agira sur une partie combinatoire que l'on présente maintenant.

Partie combinatoire

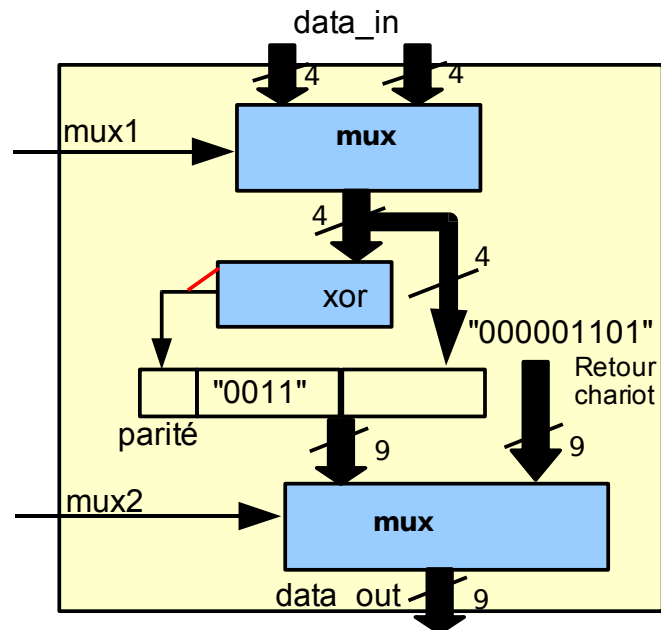
Le but de la partie combinatoire est de calculer la parité et de gérer ce que l'on envoie suivant deux entrées appelées mux1 et mux2.

Le caractère '0' a comme code ASCII "0110000" sur 7 bits soit "00110000" sur 8 bit. On peut ainsi construire simplement les codes ASCII des chiffres de '0' à '9' : "0011" suivi du chiffre sur 4 bits.

Puisque l'on a choisi une parité impaire (**odd en anglais**) elle est calculée en inversant un ou exclusif sur tous les 8 bits. Nous pouvons remarquer que le ou exclusif de "0011" est 0 et qu'ainsi seuls les 4 derniers bits sont nécessaires pour le calcul de cette parité.

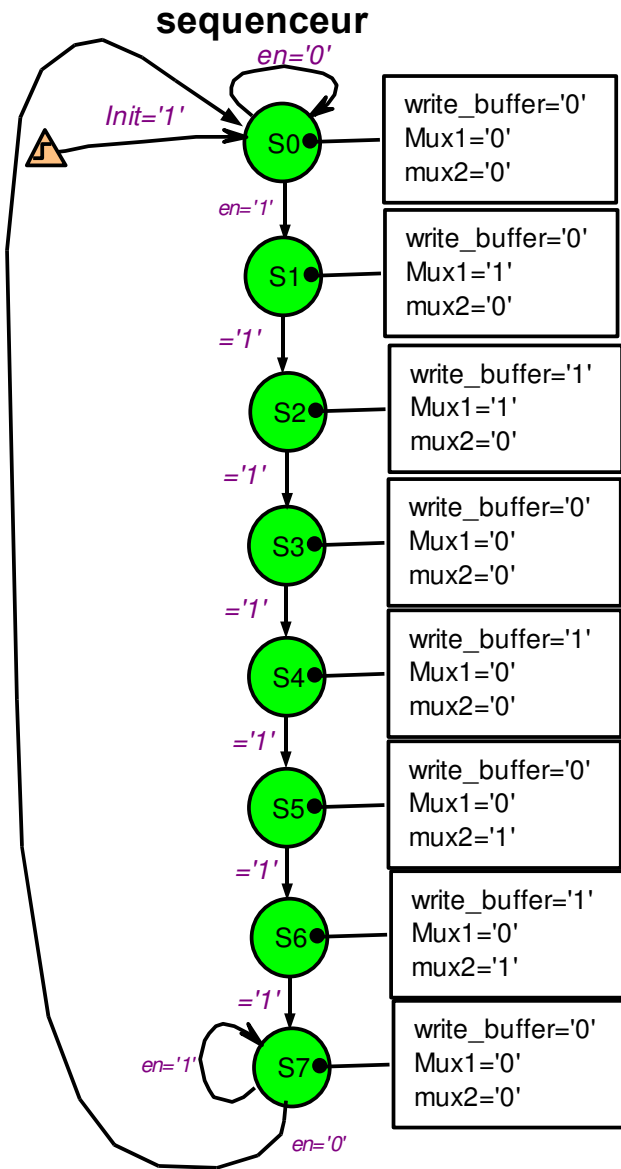
Voici schématisé le travail réalisé par ce composant combinatoire dans la figure ci-contre.

Il vous montre comment sont construits les neuf bits à envoyer, comment est calculée la parité et comment le dernier multiplexeur permet de choisir entre une valeur fixe (**ici un retour chariot qui nécessitera de choisir dans configuration, CR/LF auto pour le GTKTerm**) et une autre valeur.



Le séquenceur

Le séquenceur met en œuvre le composant combinatoire en positionnant correctement mux1 et mux2. Comme d'habitude, nous présentons le graphe d'évolution de ce séquenceur.



Comme nous l'avons vu, "en" est un signal très lent et le séquenceur attendra que ce signal repasse à un pour revenir dans son état initial S0.

En S1 Mux1 passe à 1 car on a l'intention d'envoyer les dizaines en premier.

En S2, puisque "write_buffer" passe à 1 cela veut dire que l'on écrit dans le buffer

En S3 Mux0 passe à 1 car on a l'intention d'envoyer les unités maintenant.

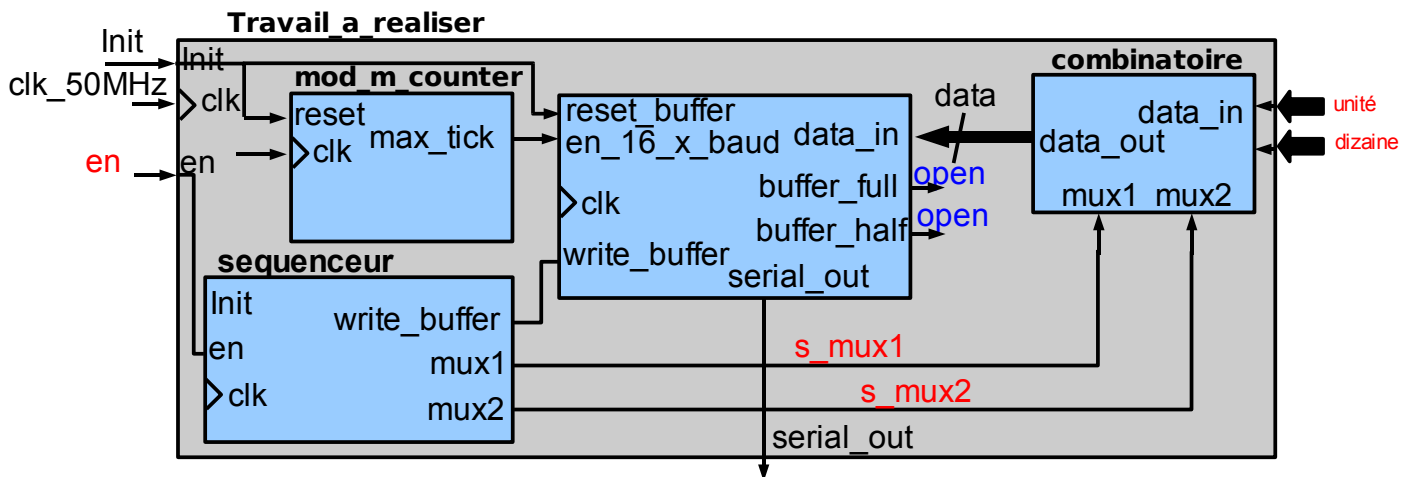
En S4, puisque "write_buffer" passe à 1 cela veut dire que l'on écrit dans le buffer

En S5 Mux2 passe à 1 pour envoyer le retour chariot (RC dans la suite)

En S6 c'est là que l'on envoie véritablement ce RC... puis on reste bloqué en S7 tant que en (signal lent) ne repasse pas à 0.

Travail à réaliser

On vous demande de réaliser pratiquement cet ensemble



dont les parties principales ont été décrites dans la préparation. Essayer de réaliser la partie combinatoire sans composants (c'est à dire avec des signaux, des équations des `with select when`)

Tester et donner le petit défaut constaté.

6. Corriger le dernier défaut

Le défaut constaté dans la section précédente est dû au fait que "en" arrive trop tôt. En effet, quand il est positionné à 1 le compteur sera incrémenté seulement sur le front suivant (de l'horloge très lente), c'est à dire quand "en" retombe à 0.

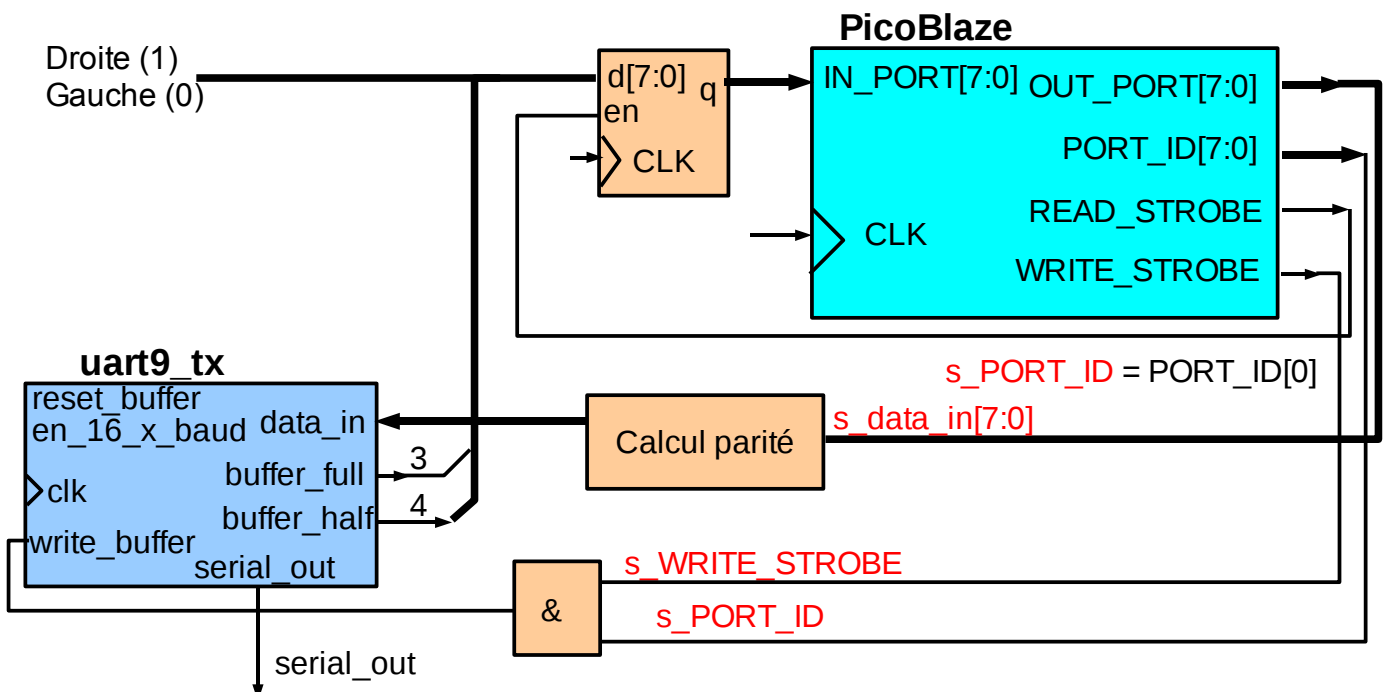
Le nouveau séquenceur partira encore avec "en", mais attendra qu'il repasse à 0 avant de commencer à envoyer dans la RS232.

Proposez et tester un séquenceur amélioré. Faire constater.

7. Aller plus loin : où pourrait-on ajouter un picoBlaze ?

Ce TP a été fait sans picoBlaze. Nous allons maintenant nous poser la question de savoir comment pourrait-on ajouter un picoBlaze. Ceci est présenté dans la figure ci-dessous : on voit que le séquenceur a disparu ainsi que les compteurs : tout est laissé au picoBlaze, du comptage à la transformation en code ASCII en passant par la lecture des entrées droite et gauche pour finir par l'écriture dans le buffer de la liaison série. Pour simplifier un éventuel programme nous avons laissé le calcul de la parité par un élément combinatoire comme cela a été fait dans les sections précédentes.

Ce montage sera utilisé en TP9 avec la liaison PS2.



TP8 - Une liaison série pour enregistrer des valeurs en RAM (sans picoBlaze)

L'envoi de données par la RS232 va maintenant servir à remplir une mémoire avec des valeurs prédéterminés. Ces valeurs seront ensuite envoyées sur des LEDs pour réaliser divers chenillars à l'aide d'un compteur.

1. Travail préliminaire

On désire réaliser un ensemble comprenant un diviseur de fréquence (le même que celui du TP7), un compteur 4 bits binaire possédant une entrée "en" de validation" et une entrée "init" (devinez pourquoi faire ?) et une mémoire.

Préparation

Pour éviter tout changement des options de compilation pour qu'ISE infère correctement une mémoire BRAM (Block RAM) nous allons utiliser un composant faisant partie des bibliothèques Xilinx. Ce composant s'appelle RAM16X8S. Le meilleur moyen d'en trouver la documentation est le suivant :

- ouvrir une page blanche de schématique (qui ne nous servira pas par la suite en fait)
- chercher ce composant dans la catégorie mémoire et le dessiner sur la page blanche
- click droit sur le composant dessiné, puis "Object properties", puis "Symbol Info"

Vous avez en bas de la page de la documentation un exemple d'instanciation de ce composant en VHDL (puis en verilog, ce qui ne nous intéresse pas). Ce que l'on vous demande de faire à ce stade est de **comprendre comment cette RAM est initialisée**.

- fermer la page schématique sans enregistrer mais garder la documentation ouverte.

Calculer les valeurs prédéfinies à mettre dans la RAM pour réaliser un chenillard aller et retour d'une LED.

Travail à réaliser

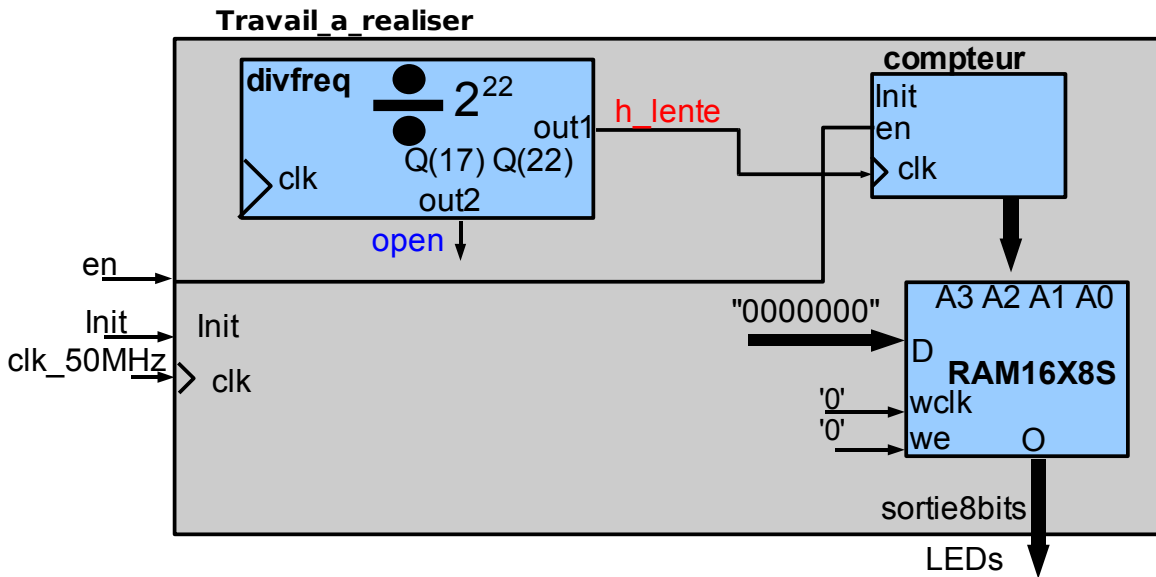
On vous demande d'importer le diviseur de fréquence du TP7 (disponible aussi dans le fichier "median.vhd" sur le site perso de S. Moutou), de réaliser un compteur sur 4 bits avec entrées "init" et "en".

Ajouter la mémoire RAM16X8S correctement initialisée, sans oublier les lignes :

```
1      Library UNISIM;
2      use UNISIM.vcomponents.all;
```

avant votre entité globale.

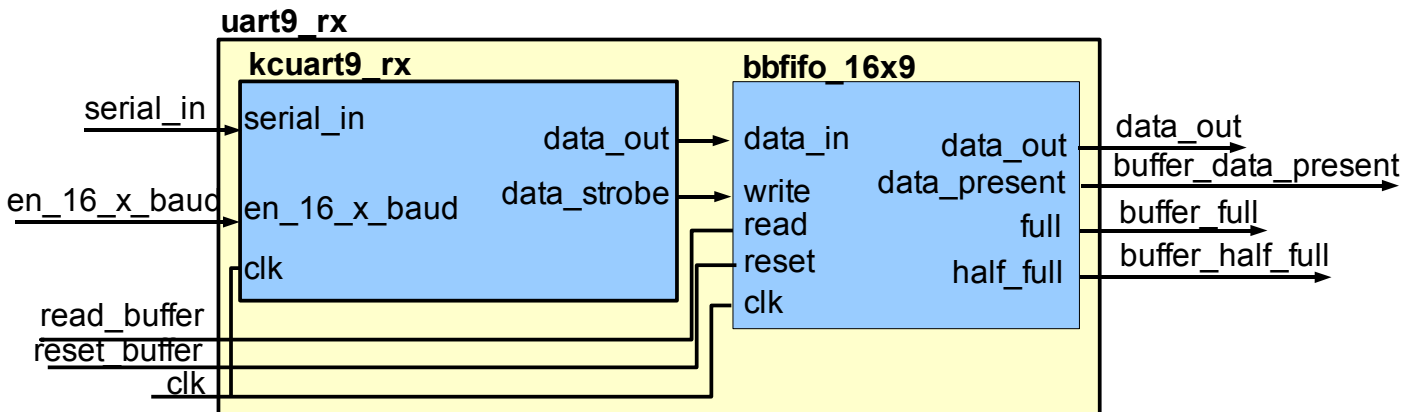
Le schéma de principe est présenté ci-dessous :



Tester et faire valider.

2. Modification du contenu de la RAM

On désire maintenant s'intéresser à la possibilité de modifier le contenu de cette mémoire à l'aide d'une liaison RS232. Par rapport au TP précédent la liaison devra être dans l'autre sens : c'est le PC qui maintenant va fournir des valeurs au FPGA. La réalisation de cette liaison se fait pratiquement comme dans le TP précédent (`_rx` au lieu de `_tx`) :

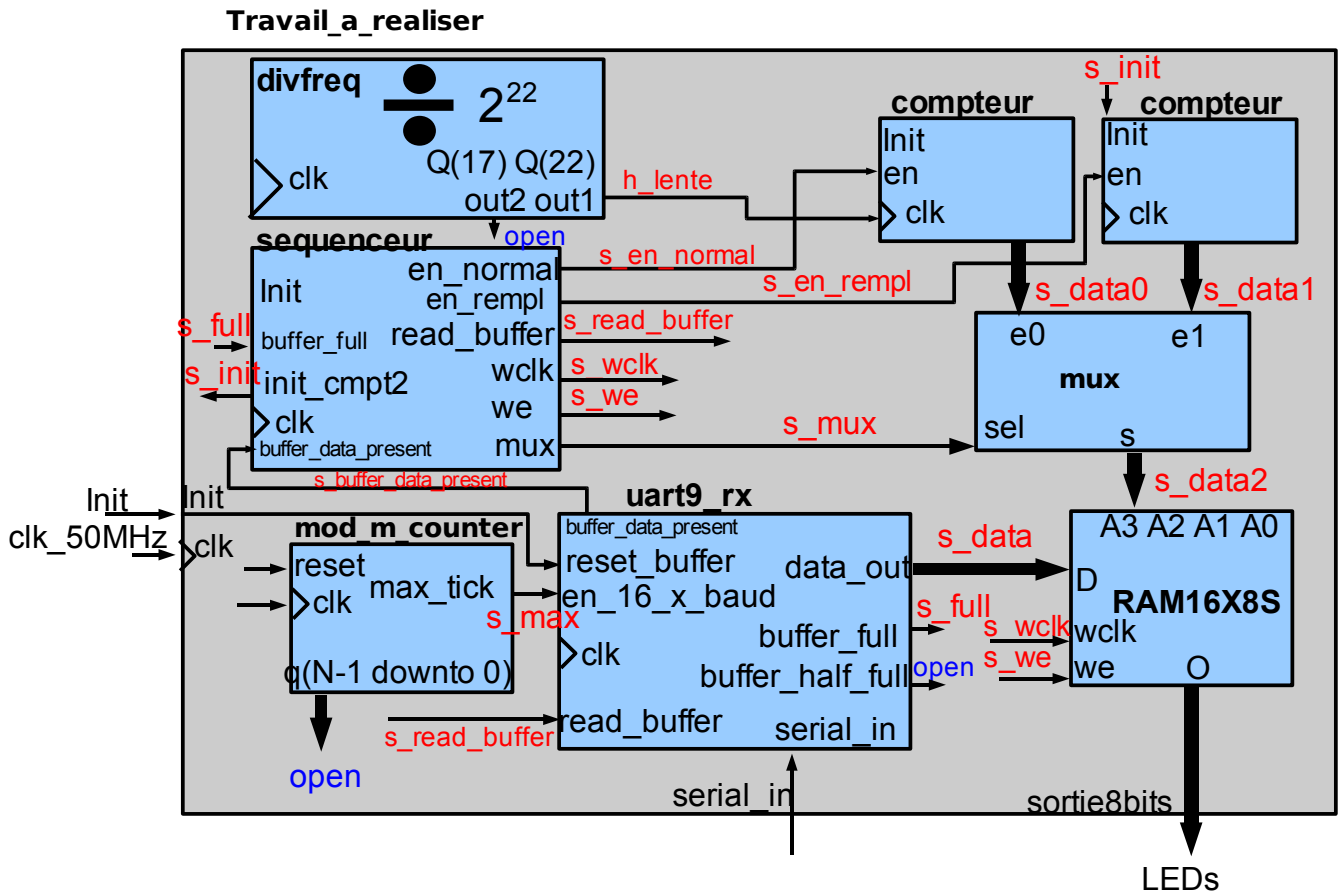


On rappelle que les trois fichiers `uart9_rx.vhd`, `kcuart9_rx.vhd` et `bbfifo_16x9.vhd` sont à chercher dans **UTTLO11.zip**

Un séquenceur sera responsable dès qu'il y a une donnée reçue ("`buffer_full`") de lire "`data_out`" et de l'écrire dans la mémoire puis d'incrémenter un compteur (d'adresses). Nous aurons en final deux compteurs : un pour lire le contenu de la mémoire et un pour écrire. Ils devront donc se partager le bus d'adresse, cela pourra se faire avec un multiplexeur dont l'entrée sera gérée par le séquenceur. Quand le séquenceur sera en train d'écrire dans la mémoire, il devra bloquer le compteur de lecture.

Travail à réaliser

Comme habituellement le travail à réaliser vous est donné sous forme schématique. Vous devez l'implanter en VHDL avec des **PORT MAP**.

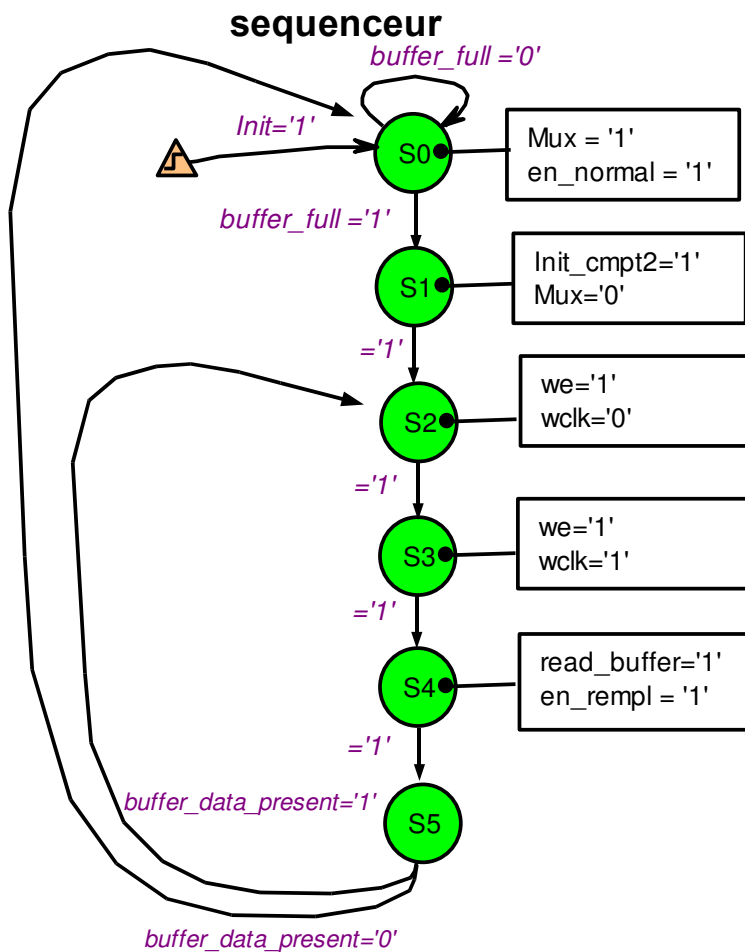


Toutes les horloges non connectées sur ce schéma sont reliées au 50MHz

Il y a beaucoup de travail, mais si vous respectez les noms des signaux (en rouge) et les noms de chacune des entrées et sorties de chaque bloc, le travail se fait rapidement. Les deux compteurs du haut sont identiques et sont simplement les compteurs de l'exercice 1 de ce TP.

Le mux est classique et déjà utilisé, mais vous avez une chance sur deux de le faire fonctionner comme il faut : comme le séquenceur ci-dessous vous l'indique, quand `sel = 1` vous sortez `e0` !

Le plus long à réaliser sera indiscutablement le séquenceur, mais ce n'est pas le premier que vous réalisez. Nous avons un petit peu hésité à le donner. Il aurait été intéressant de vous le faire chercher mais pour avoir plus de chance de faire fonctionner l'ensemble nous avons décidé de le donner (an page suivante). Son principe est de lancer le remplissage de la mémoire dès que le buffer est plein. Cette technique fonctionne correctement parce que nous avons justement 16 valeurs à envoyer à la RAM.



Une fois que toute la partie matérielle est en place, il vous faut réaliser un fichier binaire de 16 valeurs pour remplir la RAM. On vous propose le fichier C++ suivant :

```

1      /** tpmemo.cpp */
2      #include <stdio.h>
3      #include <string.h>
4      char valhex(char aconv){
5          static char hex[17]="0123456789ABCDEF";
6          char i;
7          i=0;
8          while(aconv!=hex[i]) i++;
9          return i;
10     }
11     main() {
12         FILE *sortie;
13         static char hexa[17]="0123456789ABCDEF";
14         char trame[128],nomfichier[34];
15         unsigned char nval,i,tab[255];
16         printf(" *****\n");
17         printf("\n ***** Conversion valeur -> fichier Intel HEX *****");
18         printf("\n ***** TP programmation de memoires *****");
19         printf("\n ***** Version 0.1 (S. Moutou & Cie) *****");
20         printf("\n *****\n");
21         printf("\n\nCombien de valeurs voulez-vous entrer ? ");
22         scanf("%d",&nval);
23         for (i=0;i<nval;i++) {

```

```

24         printf("%d° valeur (en hexadecimal) : ",i+1);
25         scanf("%x",&tab[i]);
26     }
27     printf ("\nQuel est votre nom de fichier de sauvegarde ? (extension .bin)
28     ");
29     scanf("%s",nomfichier);
30     sortie=fopen(nomfichier,"w");
31     fwrite(tab,1,16,sortie);
32     fclose(sortie);
33     }

```

que l'on compile avec **g++ tpmemo.cpp -o tpmemo**

On le lance alors avec `./tpmemo` :

```

[smoutou@localhost L011]$ ./tpmemo
*****
*****  Conversion valeur -> fichier Intel HEX  *****
*****  TP programmation de memoires  *****
*****  Version 0.1 (S. Moutou & Cie)  *****
*****

Combien de valeurs voulez-vous entrer ? 16
1° valeur (en hexadecimal) : 81
2° valeur (en hexadecimal) : 42
3° valeur (en hexadecimal) : 24
4° valeur (en hexadecimal) : 18
5° valeur (en hexadecimal) : FF
6° valeur (en hexadecimal) : 00
7° valeur (en hexadecimal) : FF
8° valeur (en hexadecimal) : 00
9° valeur (en hexadecimal) : 18
10° valeur (en hexadecimal) : 24
11° valeur (en hexadecimal) : 42
12° valeur (en hexadecimal) : 81
13° valeur (en hexadecimal) : 00
14° valeur (en hexadecimal) : FF
15° valeur (en hexadecimal) : 00
16° valeur (en hexadecimal) : FF

Quel est votre nom de fichier de sauvegarde ? (extension .bin) demo1.bin

```

Tester avec gkterm configuré comme ci-dessous :

PORT `:/dev/ttyS0`, vitesse : 19200, Parité : odd, Bits 8, Bit de stop : 1, contrôle de flux : none

Fichier -> envoi de fichier brut en cherchant votre fichier binaire créé.

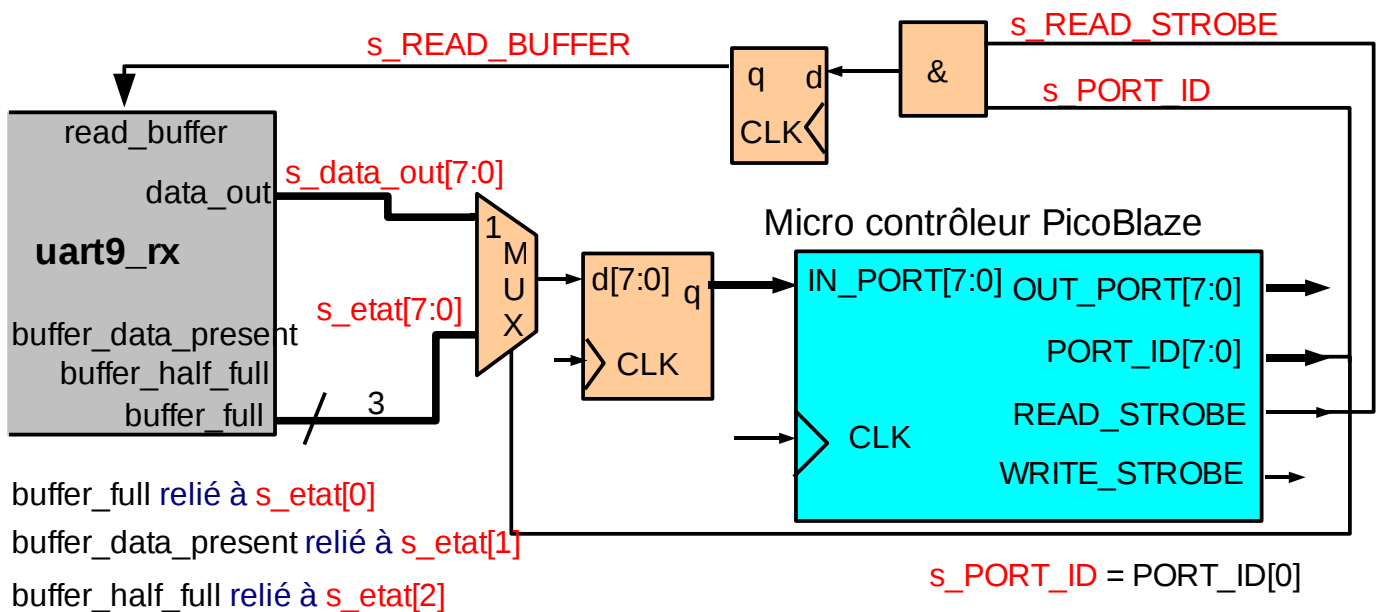
Si tout fonctionne, vous obtiendrez un nouveau chenillard.

Arrivé à ce point vous pouvez lire la suite mais aussi reprendre la partie non réalisée du TP précédent.

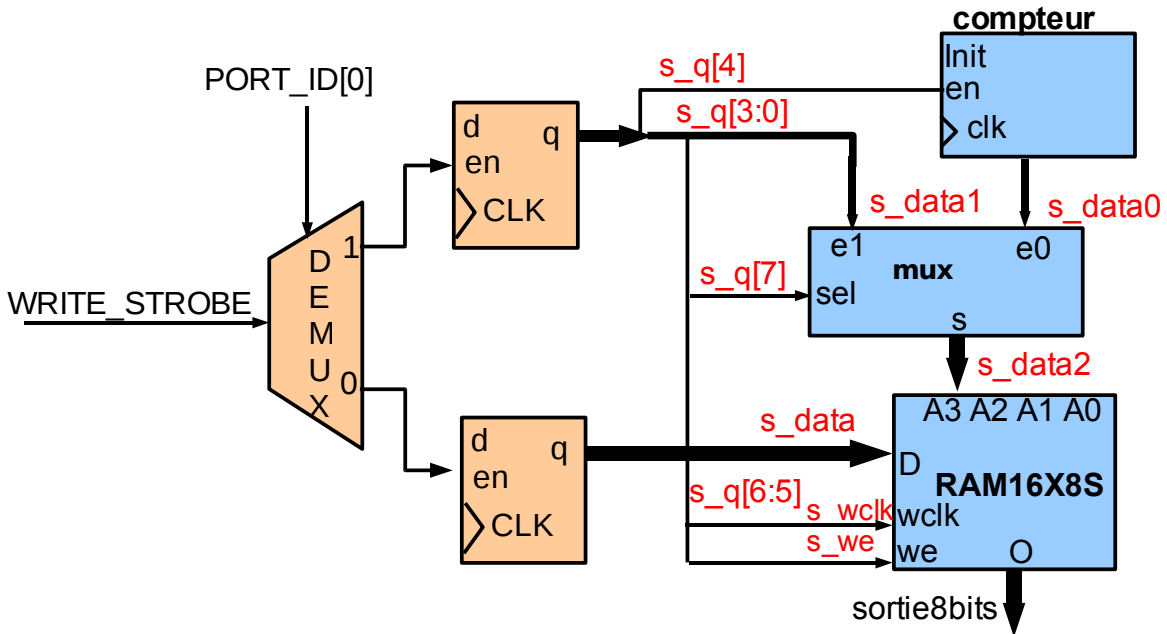
3. Aller plus loin : interfacer un picoBlaze.

Notre séquenceur fonctionne correctement parce que nous avons tout juste 16 valeurs à envoyer et qu'ainsi il était possible de démarrer avec le signal "buffer_full". Si vous avez plus de 16 valeurs à envoyer, il faudrait mieux démarrer sur "buffer_data_present" et quand la valeur est écrite attendre un certain temps (à déterminer). Si de nouveau "buffer_data_present" passe à un avant l'écoulement du temps, on incrémente le compteur pour envoyer une nouvelle valeur, autrement on revient à l'état initial.

Utiliser un picoBlaze pour gérer tous ces problèmes est une bonne idée. Voici comment interfacer une liaison série entrante avec le picoBlaze :



Ceci sera mis en œuvre un peu plus loin. Il nous faut encore interfacer la mémoire pour pouvoir la remplir à partir du picoBlaze. Ceci peut être réalisé avec le schéma de principe suivant :



Vous voyez apparaître deux PORTs en sortie : Le port 1 remplace le séquenceur et le compteur qui permettait de remplir la mémoire tandis que le port 0 présente les nouvelles données à écrire dans la RAM.

Dans le TP10 on fera encore mieux dans l'enfouissement : la RAM sera la RAM interne du picoBlaze, le compteur sera une variable interne du picoBlaze ... enfin bref il ne restera rien de la partie en bleu de ce schéma : un seul PORT de sortie réalisera "sortie8bits". Bon d'accord comme il n'y a plus rien, on vous laissera trouver le programme picoBlaze pour remplacer tout cela... il sera grand temps de mettre en application vos connaissances en assembleur picoBlaze.

TP9 Liaison PS2

La liaison PS2 est (ou devrait-on dire était) utilisée pour la communication entre claviers et souris et ordinateurs PC. L'objectif de ce TP est de réaliser un décodage des informations provenant d'un clavier.

1. Travail simple

On vous demande de lire très rapidement

http://fr.wikiversity.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language/Exercices/TP_3

jusqu'à la section "Mise en conformité des horloges".

On vous demande de prendre la correction de P. P. Chu ou celle à six états proposée par l'un d'entre nous, et de l'adapter à votre carte.

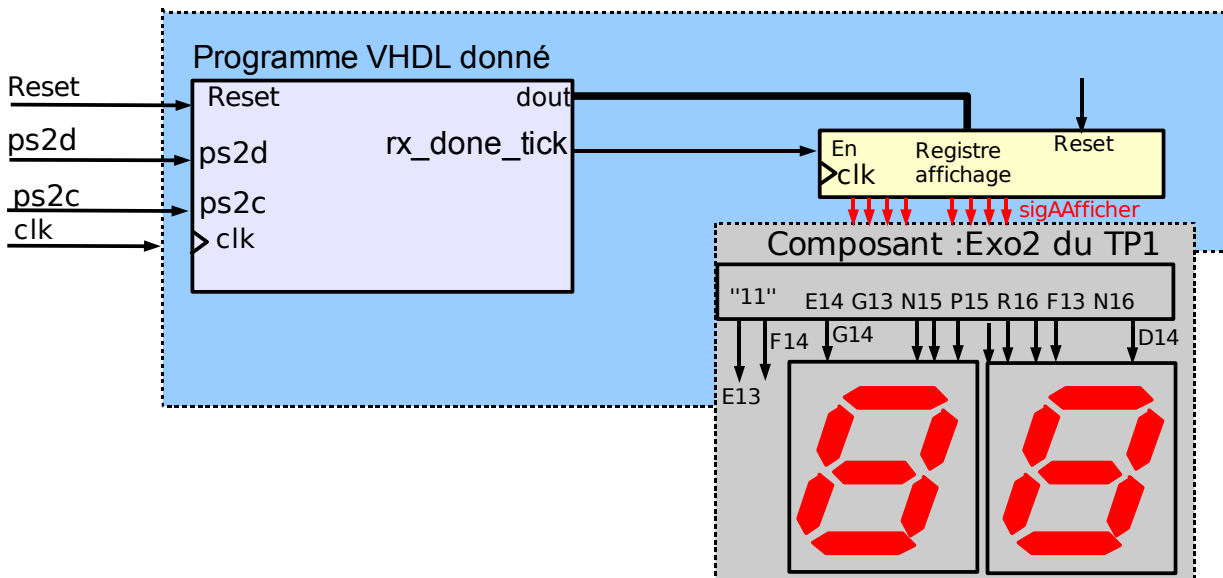
Ce qu'il y a à adapter c'est :

Retrouver la correction du TP1 du WIKI pour le rectangle gris ou votre propre transcodeur du TP2 de LO11.

Si WIKI, retoucher l'affichage pour deux afficheurs seulement avec segment actif quand entrée correspondante est à 0 alors que pour vous il faut un 1 !!!

Ajouter le registre d'affichage (rectangle jaune de la figure ci-dessous)

Réaliser complètement le grand rectangle bleu

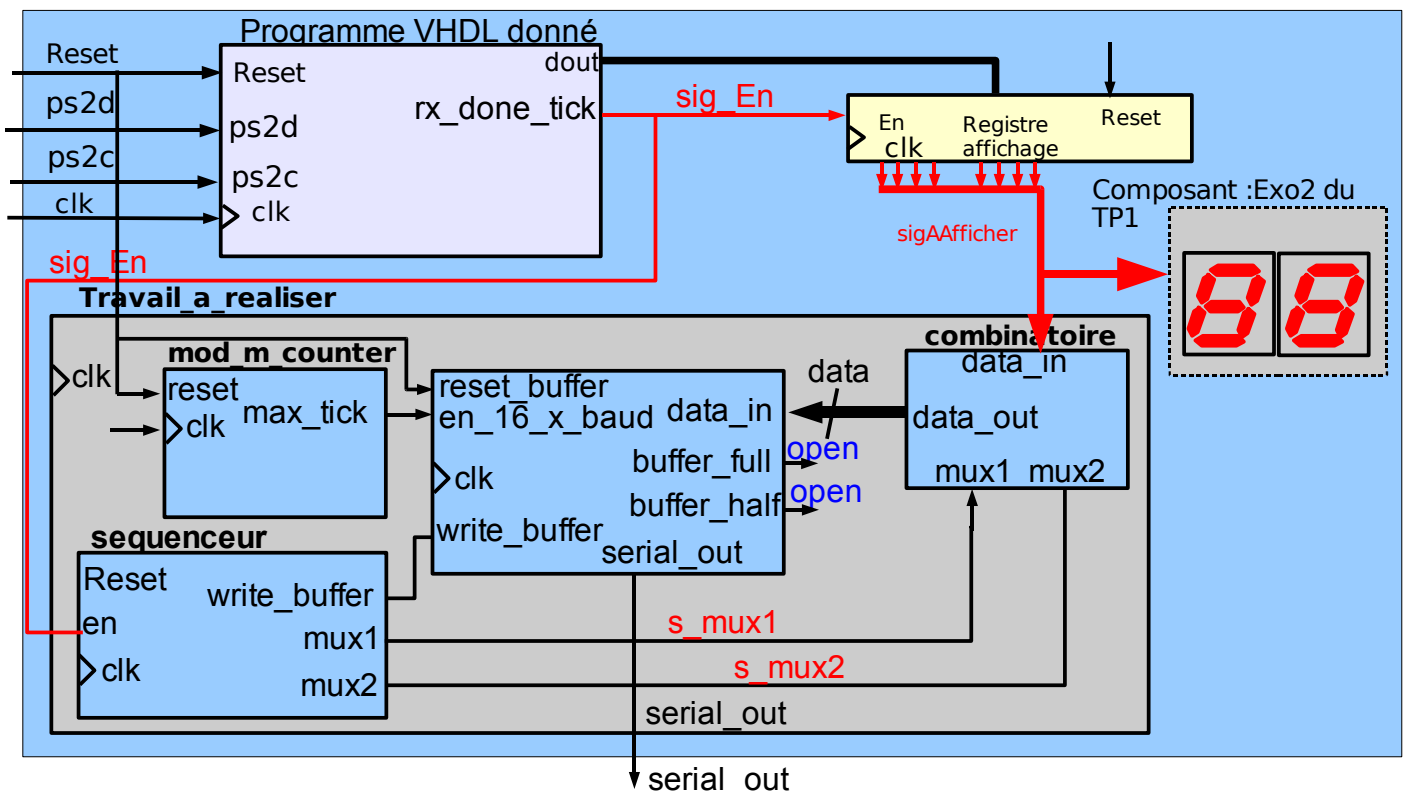


Faire constater quand cela fonctionne.

2. Trouver ce qui sort réellement du clavier

Votre affichage ne permet d'afficher que deux codes hexadécimaux alors que plusieurs sont envoyés à la suite. Pour remédier au problème, on vous demande d'adapter une transmission série (comme en TP 7 pour que chaque fois qu'une valeur est affichée dans le registre d'affichage cette même

valeur est envoyée sur un octet dans le FIFO (qui enverra le tout sur la RS232).



Nous allons dans un premier temps reprendre la partie "**Travail_a_realiser**" du compteur de passages du TP7. Il n'est pas difficile de s'apercevoir que cela ne fonctionne pas complètement correctement : pouvez-vous expliquer pourquoi les octets sont affichés bizarrement avec des caractères '?' ou '>' et autres ?

Vous pouvez garder les deux afficheurs en parallèle pour vérification (comme présenté dans le schéma).

Jusqu'à présent nous n'avons pas énormément travaillé : uniquement utilisé de la récup de composants. Il nous faut maintenant mettre les mains dans le cambouis pour avancer un peu. La partie "**combinatoire**" du compteur de passage est un peu trop restrictive : elle permet seulement de transformer des chiffres BCD. Nous voulons maintenant envoyer de l'hexadécimal. Il nous faut donc améliorer cette partie combinatoire. C'est ce que l'on se propose de faire dans la prochaine section.

3. Améliorer l'affichage dans gtkterm

Utiliser des LUTs en VHDL (rappel)

Un exemple est donné maintenant :

```

1      library IEEE;
2      use IEEE.STD_LOGIC_1164.ALL;
3      library unisim;
4      use unisim.vcomponents.all;
5      ENTITY transcodeur IS PORT(
6          e : in STD_LOGIC_VECTOR(3 DOWNTO 0);    -- 4 entrées
7          s : out STD_LOGIC_VECTOR(6 DOWNTO 0));  -- 7 sorties
8      END transcodeur;
```

```

9      ARCHITECTURE atranscodeur OF transcodeur IS BEGIN
10     i1 : LUT4
11         generic map (INIT => X"EAAA")
12     port map( I0 => e(0),
13             I1 => e(1),
14             I2 => e(2),
15             I3 => e(3),
16             O => s(0) );
17     .....
```

Cet exemple vous montre comment on câble une **LUT4** en VHDL (**port map**) et comment on l'initialise (**generic map**). Le câblage de ce composant est correct mais pas son initialisation puisqu'on vous demande de la calculer plus loin.

Les deux lignes **library ...** et **use ...** sont à ajouter avant toute entité qui utilise une LUT en plus bien sûr de "**library ieee;**".

Préparation de la table de vérité

On rappelle qu'une parité impaire veut dire que nos neufs bits seront en nombre impair. Compléter la partie parité de la table de vérité en essayant par la même occasion de comprendre la partie du contenu donné (c'est à dire la notion de code ASCII).

e(3)	e(2)	e(1)	e(0)	parité	b7	b6	b5	b4	b3	b2	b1	b0	caractère
0	0	0	0	1	0	0	1	1	0	0	0	0	'0'
0	0	0	1		0	0	1	1	0	0	0	1	'1'
0	0	1	0		0	0	1	1	0	0	1	0	'2'
0	0	1	1		0	0	1	1	0	0	1	1	'3'
0	1	0	0		0	0	1	1	0	1	0	0	'4'
0	1	0	1		0	0	1	1	0	1	0	1	'5'
0	1	1	0		0	0	1	1	0	1	1	0	'6'
0	1	1	1		0	0	1	1	0	1	1	1	'7'
1	0	0	0		0	0	1	1	1	0	0	0	'8'
1	0	0	1		0	0	1	1	1	0	0	1	'9'
1	0	1	0		0	1	0	0	0	0	0	1	'A'
1	0	1	1		0	1	0	0	0	0	1	0	'B'
1	1	0	0		0	1	0	0	0	0	1	1	'C'
1	1	0	1		0	1	0	0	0	1	0	0	'D'
1	1	1	0		0	1	0	0	0	1	0	1	'E'
1	1	1	1	0	0	1	0	0	0	1	1	0	'F'

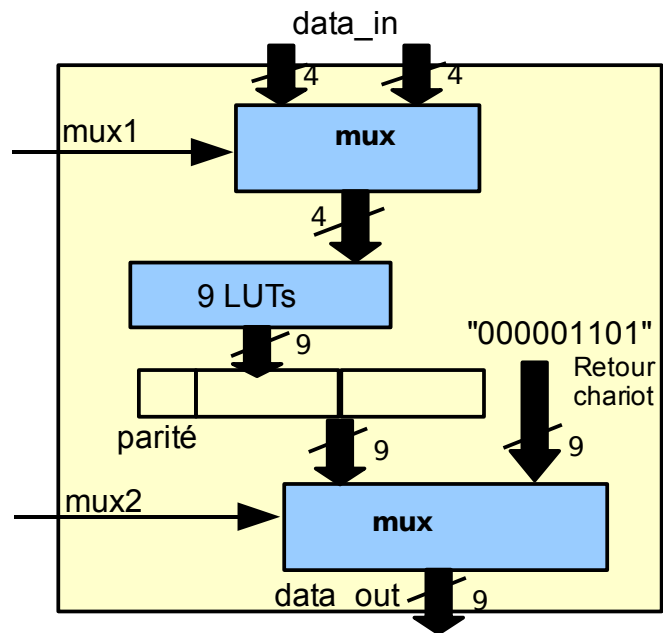
Travail à réaliser

Nous allons corriger l'affichage pour finir et avoir un affichage hexadécimal. Pour cela nous allons modifier le composant combinatoire du TP7 comme la figure le montre avec 9 LUTs.

Montrer que nous avons besoin en fait de moins de 9 LUTs, car deux LUTs sont identiques et une est à 0.

Nous n'avons plus besoin de calcul de parité car ce sera réalisé par une LUT4.

Sachant que l'on peut mélanger dans une architecture des équations avec des **port map** on vous demande de repérer et de modifier ce composant.



Indications :

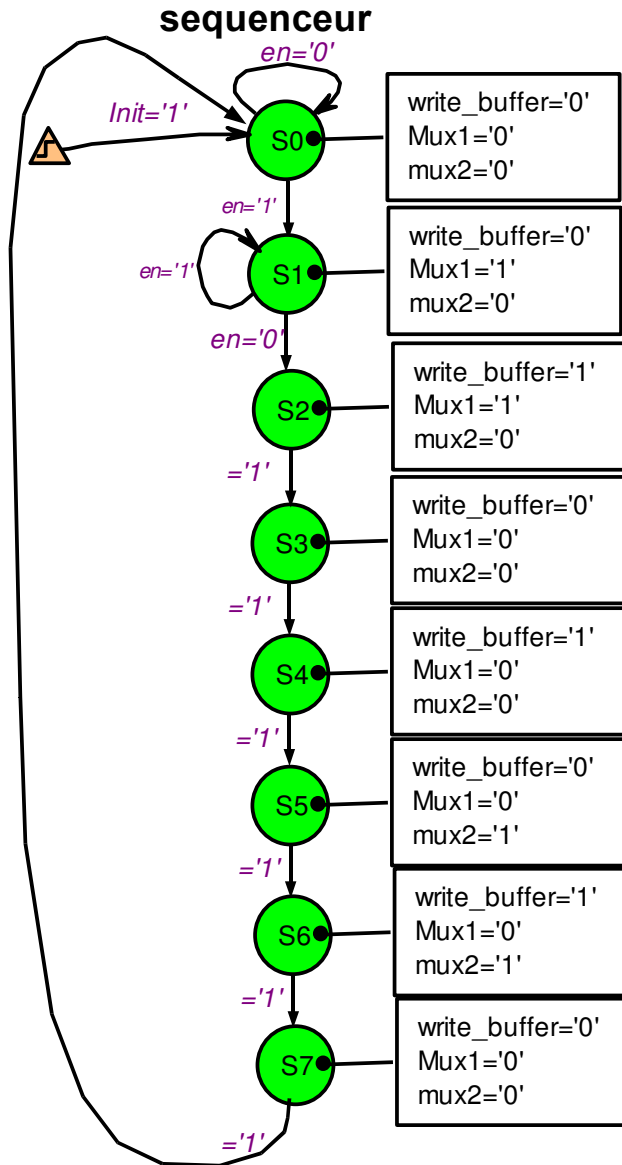
```

1      library ieee;
2      use ieee.std_logic_1164.all;
3      library unisim; -- ajouté pour TP9
4      use unisim.vcomponents.all; -- ajouté pour TP9
5      entity combinatoire is
6          port (data_in : in std_logic_vector(7 downto 0);
7                mux1, mux2 : in std_logic;
8                data_out : out std_logic_vector(8 downto 0)
9          );
10     end combinatoire;
11     architecture acombinatoire of combinatoire is
12         signal s_data, s_data_1 : std_logic_vector(8 downto 0);
13     begin
14         i1 : LUT4
15             generic map (INIT => X"56AA")
16             port map( I0 => s_data_1(0),
17                     I1 => s_data_1(1),
18                     I2 => s_data_1(2),
19                     I3 => s_data_1(3),
20                     O => s_data(0) );
21         ....
22         with mux1 select
23             s_data_1(3 downto 0) <= data_in(3 downto 0) when '0',
24                                     data_in(7 downto 4) when others;
25         with mux2 select
26             data_out <= s_data when '0',
27                         "000001101" when others; --RC ici (ou LF ?)
28                                     --choisir dans configuration CR/LF auto
29     end acombinatoire;
30

```

Le séquenceur est très similaire à celui du compteur de passages.

Il est donné ci-contre.



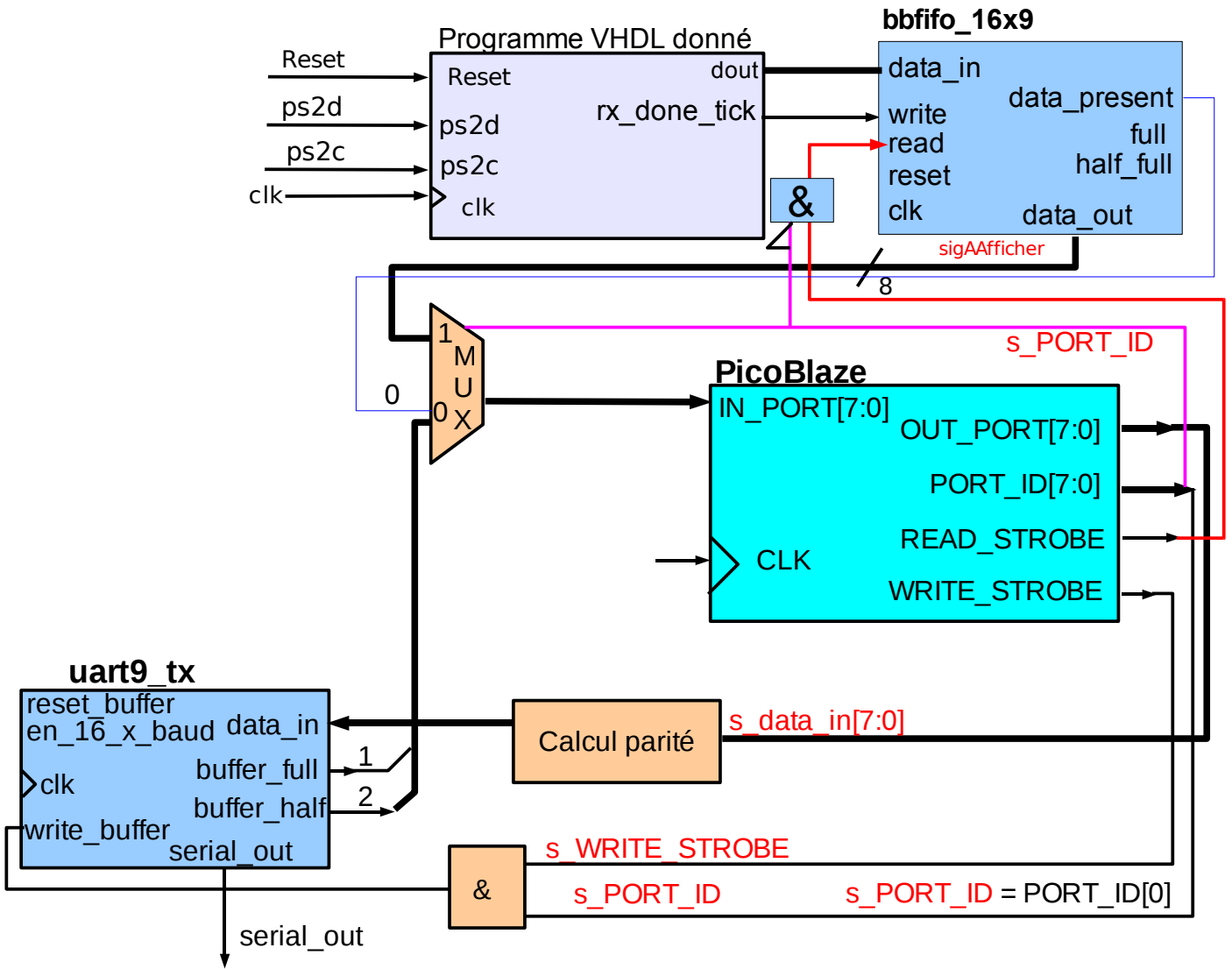
4.Plus loin encore : interfacier un picoBlaze

Interfacier un picoBlaze à un clavier PS2 peut se faire à plusieurs niveaux :

- entre le module combinatoire (que l'on garde) et le module série géré alors par le picoBlaze ; il lui faudra alors jouer le rôle du séquenceur qui doit gérer le reset (trop court pour le picoBlaze).
- en lieu et place du registre d'affichage et du séquenceur GRAFCET et du reste.

Pour la deuxième solution on a choisi en fait d'utiliser un bbfifo_16x9 déjà utilisé dans le TP8.le programme attendra la présence de données (en PORT 0), lira son entrée correspondant en PORT 1, transformera tout cela en ASCII avant de le renvoyer sur la liaison série.

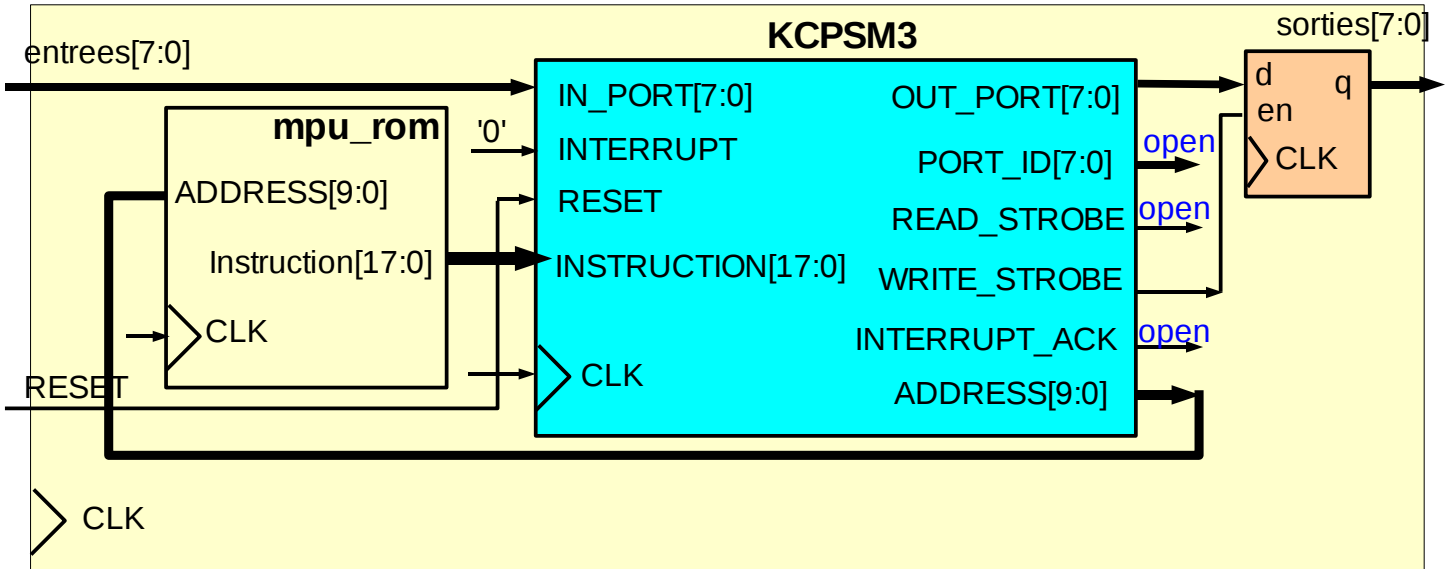
Le schéma complet est proposé ci-dessous.



TP10 Programmer le PicoBlaze

1. Introduction

Nous redonnons pour l'expliquer un peu mieux le schéma de principe d'utilisation du picoBlaze.



Face à ce schéma vous êtes susceptible de vous posez deux questions essentielles :

- comment fais-je pour entrer et sortir des informations comme allumer des LEDs, lire des interrupteurs enfin toutes les choses classiques que l'on a vu jusqu'ici ? On a déjà partiellement répondu à cette question mais il y a encore à faire.

- comment fais-je pour mettre un programme dans la mémoire ? C'est l'objectif de ce TP.

Utiliser "kpicosim" pour programmer

kpicosim est un logiciel qui permet d'écrire des programmes en assembleur et de les assembler pour obtenir une description VHDL de la mémoire ROM contenant le programme.

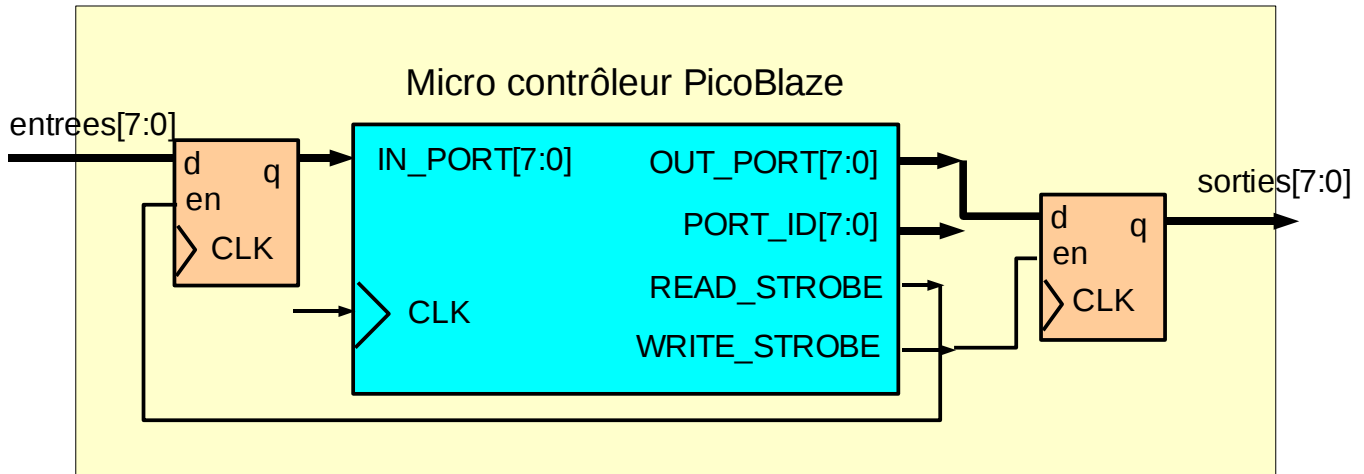
Pour ce faire il a besoin d'un fichier modèle qui se trouve dans "KCPSM3.zip" dans le répertoire **/Assembler** et qui s'appelle "ROM_form.vhd". Prenez ce fichier et mettez-le à un endroit précis pour être capable de le retrouver.

Pendant que vous y êtes prenez le fichier kcpsm3.vhd qui est le **picoBlaze** ainsi que tous les fichiers pour la liaison série (c'est toujours dans le répertoire VHDL).

Après compilation, pour générer le fichier VHDL :

File -> Export -> VHDL où template doit pointer sur votre fameux ROM_form.vhd,
output directory serait bien dans votre projet VHDL

Entity name, prenez ce que vous voulez mais vous en aurez besoin pour connecter (et c'est aussi votre nom de fichier vhd pour la mémoire.



où l'on a omis la mémoire programme qui est toujours présente mais ajouté un registre pour le PORT (unique) de sorties.

On gardera ce registre de sortie dans toute la suite du TP (avec ou sans celui d'entrée).

2. Travail introductif

Il est possible que la partie matérielle correspondant à la figure ci-dessus ait été déjà complètement ou partiellement réalisée. Cherchez dans vos archives. En tout cas il est donné dans le TP6.

Travail à réaliser (Exercice 1)

On vous demande de réaliser le montage ci-dessus pour qu'il réalise correctement l'affichage sur deux digits de la valeur hexadécimale entrée sur les interrupteurs. Vous vous efforcerez à réaliser le programme correspondant.

Indications : on pourra encore se référer au document de la wikiversité :

http://fr.wikiversity.org/wiki/Very_High_Speed_Integrated_Circuit_Hardware_Description_Language

Dans le chapitre TP4 (de ce WIKI), vous trouverez un programme qui fait la même chose mais sur deux ports différents : un port pour sortir la valeur sept segments et un port pour sélectionner un des quatre afficheurs présents sur la carte (Digilent spartan3) qui n'est pas la notre.

Remarquez l'adressage indexé dans le programme pour l'initialisation :

```
LOAD s0,01 ; met 1 dans registre s0
STORE s0,00 ; met s0 dans la mémoire RAM à l'adresse 00
LOAD s0,4F ; met 4F dans registre s0
STORE s0,01 ; met s0 dans la mémoire RAM à l'adresse 01
```

3. Retour sur les chenillards et la RS232 (comme en TP8)

On désire maintenant réaliser un chenillard avec le PicoBlaze.

Travail à réaliser (exercice 2)

1°) Réaliser le programme faisant un aller et retour (section d'introduction précédente pour lequel on revient à la sortie sur LEDs - modification du fichier ucf - modification du programme).

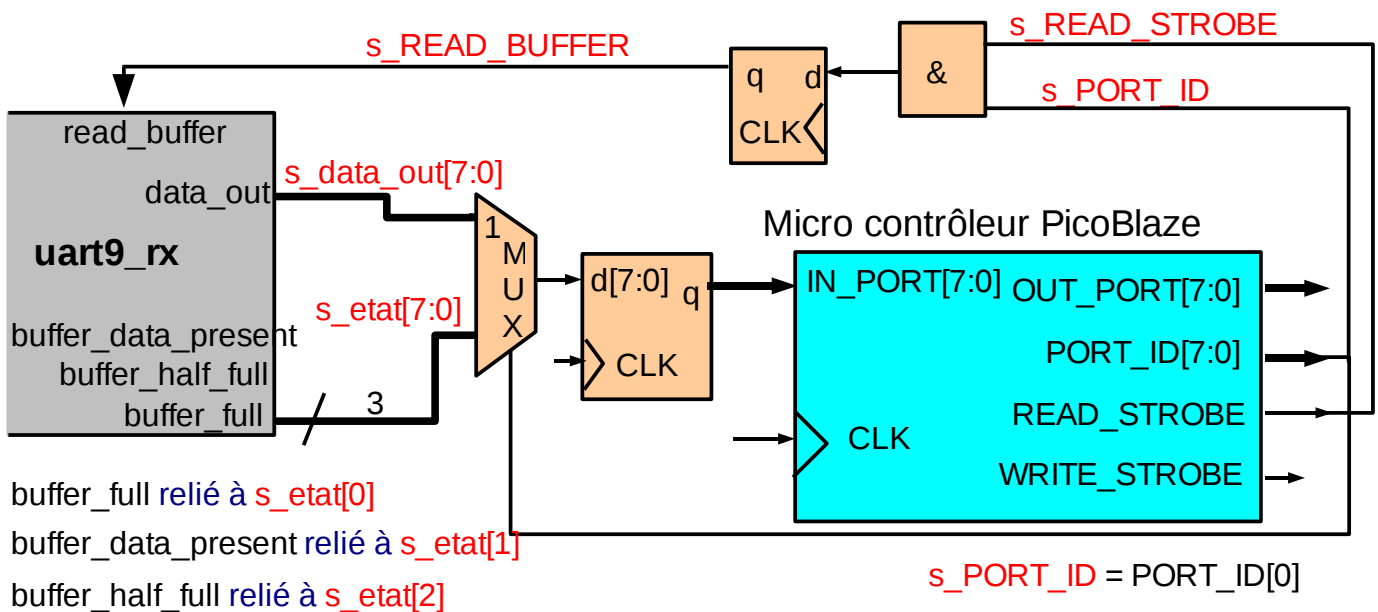
2°) Modifier le programme précédent pour qu'il utilise l'adressage indexé comme dans l'exercice la section précédente, sur 16 valeurs entrées en mémoire dans la partie initialisation.

Indications :

Le sous-programme d'attente est présenté dans les indications de l'exercice 3.

Travail à réaliser (exercice 3)

On cherche à connecter la réception RS232 (uart9_rx voir TP8) au picoBlaze. La façon de connecter est un peu spéciale et présentée ci-dessous :



On a omis dans ce schéma la sortie dans un registre complètement à droite, présentée dans l'exercice précédent.

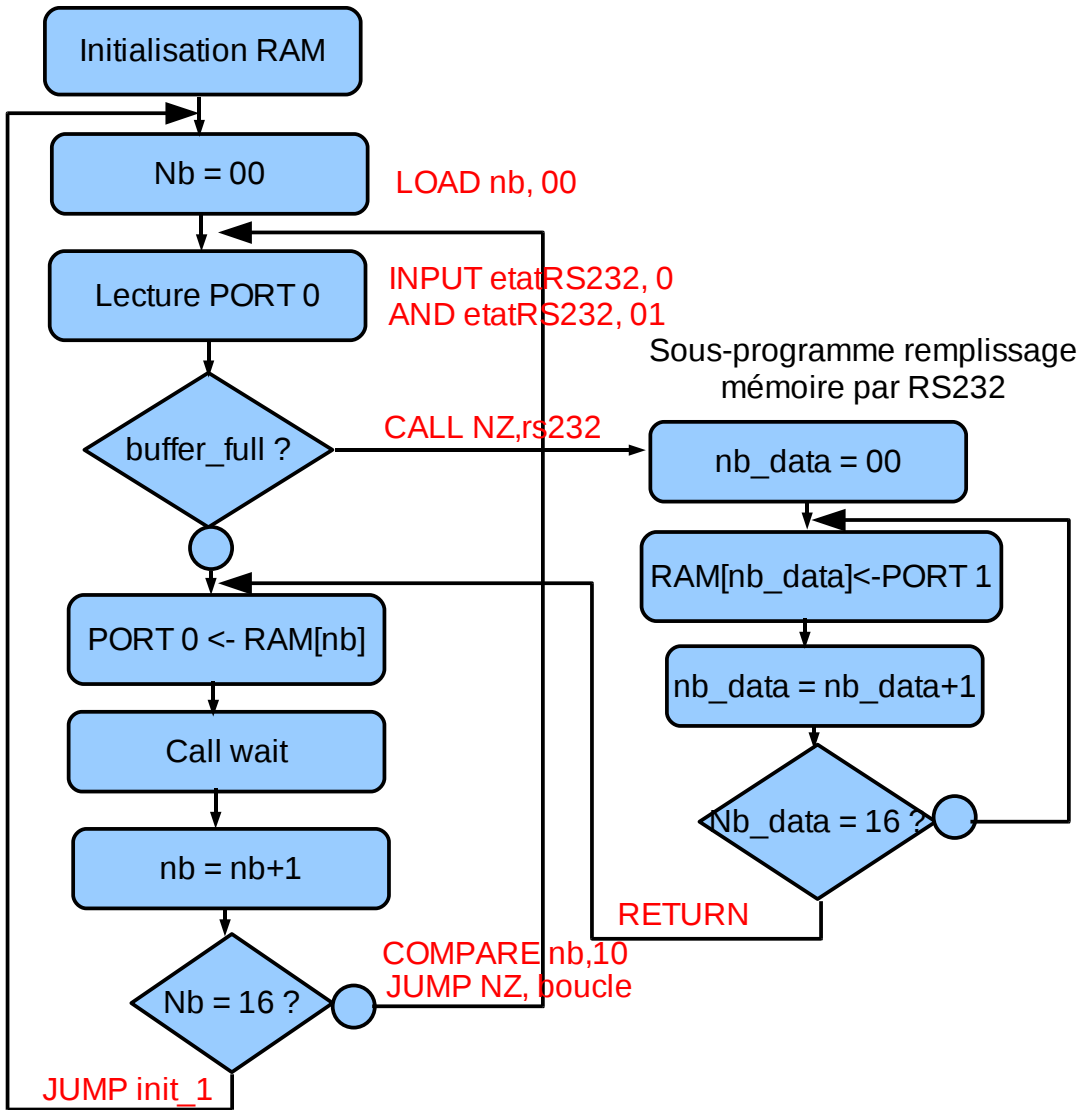
Le schéma propose de relier les trois signaux "buffer_data_present", "buffer_full" et "buffer_half_full" au deuxième PORT. En fait, seul le signal "buffer_full" nous importe, il faut donc bien noter où il est connecté. Remarquez aussi que quand la sélection du multiplexeur est à 1, c'est data_out qui arrive au picoBlaze (autrement dit les données seront disponibles lors d'une lecture du PORT 1 mais pour voir s'il y a des données, il nous faut lire le PORT 0).

1°) Réaliser l'ensemble matériel en VHDL sans oublier le générateur de bauds (voir TP7)

2°) Réaliser le programme complet du PicoBlaze : ce programme teste régulièrement si des données sont disponibles dans le buffer. Si oui, il les lit et les met dans la mémoire à la place des autres. Si non il réalise la boucle de décalage des LEDs.

Indications

L'organigramme du programme à réaliser est présenté maintenant :



Vous pouvez faire le remplissage de la mémoire par la RS232. On remarque que les compteurs de boucle vont décroissants car les tests de fin de boucle sont alors plus faciles à réaliser. On vous donne le sous-programme wait :

```

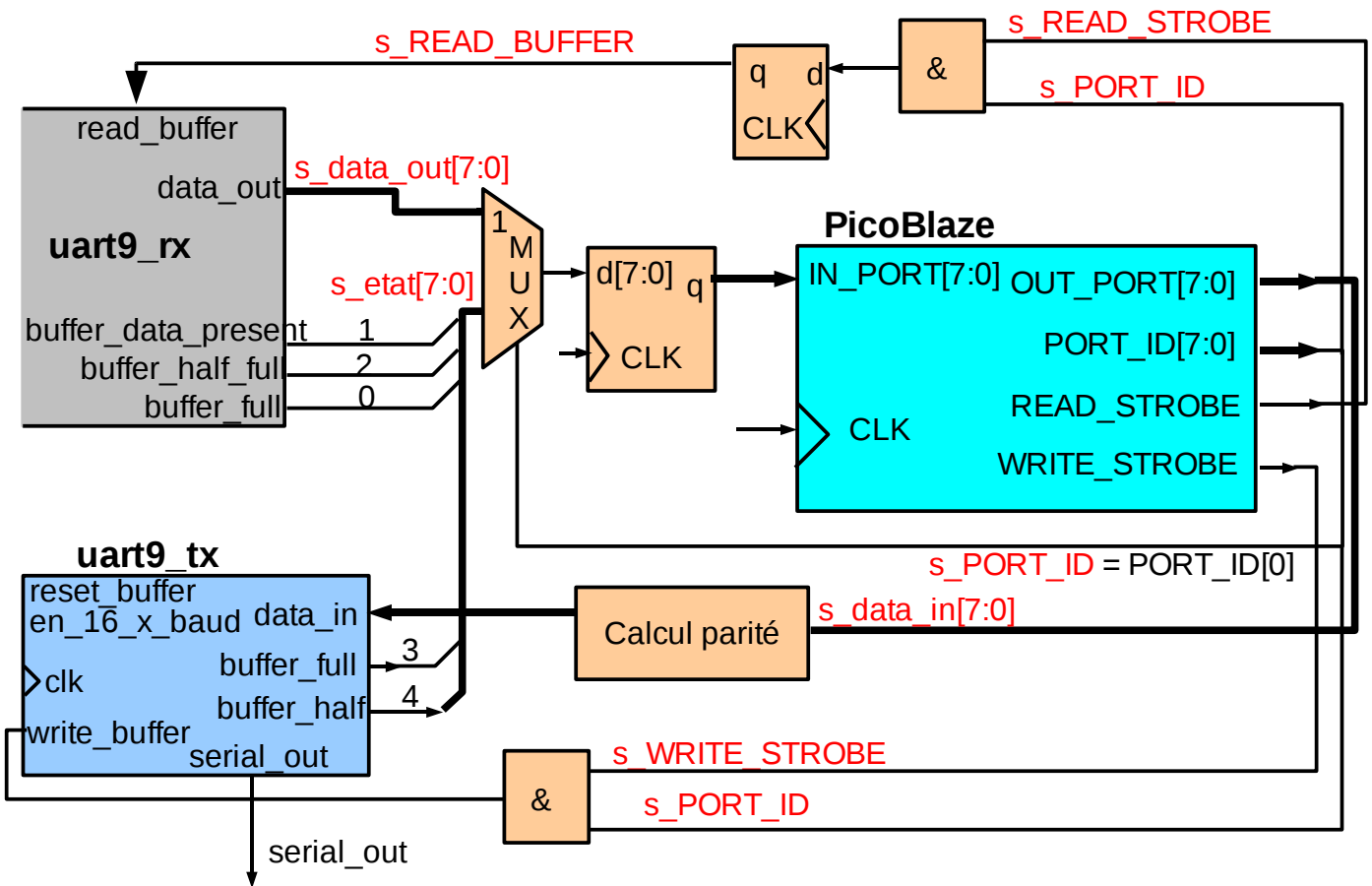
1      constant MAX, 80
2      namereg s0,i
3      NAMEREG s1, nb
4      NAMEREG s3, leds ; rename register s3 as "leds"
5      NAMEREG s4, etatRS232
6      NAMEREG s5, nb_data
7      NAMEREG s6, data_rs232
8      namereg s7,j
9      namereg s8,k
10     debut:
11     ....
12     ;=== triple boucle d'attente===
13     wait:
14         LOAD i,MAX
15     loop:
16         LOAD j,MAX
17     loop_1:
18         LOAD k,MAX
19     loop_2:
    
```

```

20             SUB k,01
21             JUMP NZ, loop_2
22             SUB j,01
23             JUMP NZ, loop_1
24
25             SUB i,01
26             JUMP NZ, loop
27             RETURN
    
```

4.RS232 bidirectionnelle

On va partir du schéma de principe de l'exercice 2 et lui ajouter toute la partie matérielle nécessaire à l'émission de la RS232.



Travail de préparation

Quel est le PORT utilisé par uart9_tx ? Rappelez-vous qu'à ce stade tous les PORTs sont utilisés par une sortie non représentée ici. Comment faire pour mettre cette sortie sur LEDs au port 0 ?

Travail à réaliser (exercice 4)

Réaliser la partie matérielle en prenant soin de mettre les LEDs en PORT 0.

1°) La partie logicielle sera destinée à réaliser un simple echo. On devra cependant gérer les retours chariots comme fin de ligne, Encore une fois le nombre de caractères reçus sera supposé ne pas dépasser 16. Notre programme devra donc stocker la chaîne de caractères jusqu'à l'arrivée d'un

retour chariot puis envoyer cette chaîne à GTKTerm. Chaque fois qu'une chaîne est remplie on allumera les LEDs que l'on éteindra lorsque le renvoi de la chaîne sera terminé.

Pour faciliter la relecture des programmes, on pourra utiliser des constantes comme :

```

1          CONSTANT UART_status_port, 00 ;UART status input
2          CONSTANT tx_half_full, 10    ; Transmitter      half full - bit4
3          CONSTANT tx_full, 08         ; FIFO              full - bit3
4          CONSTANT rx_half_full, 04    ; Receiver         half full - bit2
5          CONSTANT rx_full, 01         ; FIFO              full - bit0
6          CONSTANT rx_data_present, 02 ;                  data present - bit1

```

Ces constantes sont très liées à la partie matérielle.

2°) Améliorer la partie logicielle pour gérer les backspace comme caractère d'effacement.

Indications

Gestion des caractères normaux, retour chariot et backspace :

```

1          COMPARE UART_data, character_CR ;test for end of string
2          RETURN Z
3          COMPARE UART_data, character_BS ;test for back space
4          JUMP Z, BS_edit
5          ADD s1, 01                      ;increment memory pointer
6          COMPARE s1, s2                   ;test for pointer exceeding 16 characters
7          JUMP NZ, receive_full_test      ;next character
8          CALL send_backspace             ;hold end of string position on display
9          BS_edit:
10         SUB s1, 01                      ;memory pointer back one
11         COMPARE s1, string_start        ;test for under flow
12         JUMP C, string_start_again
13         CALL send_space                  ;clear character at current position
14         CALL send_backspace              ;position cursor
15         JUMP receive_full_test           ;next character
16         string_start_again:
17         CALL send_greater_than          ;restore '>' at prompt
18         JUMP receive_string              ;begin again

```

Travail à réaliser (Exercice 5)

On désire réaliser un programme dans le picoBlaze qui soit capable de réaliser les opérations suivantes sur les LEDs, ordres envoyés par la liaison RS232 à l'aide de GTKTerm.

clearall : éteint toutes les leds

setall : allume toutes les leds

set led0 : allume la led 0

clear led1 éteint la led 1

Après chacune de ces commandes l'état des leds est systématiquement envoyé dans GTKTerm avec un format de votre choix.