TP Electronique Numérique

ENSICAEN 1ere Année Electronique
isabelle.lartigau@ensicaen.fr
arnaud.martin@ensicaen.fr

Table des matières

Chapitre 1

Fonctions combinatoires

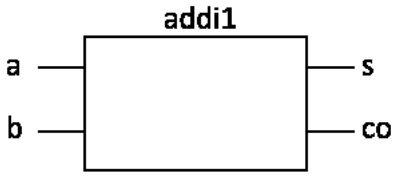
Les fonctions combinatoires sont indispensables au fonctionnement de tout système numérique. La table de vérité d'une fonction combinatoire constitue son ADN, elle fournit des informations claires sur son mode de fonctionnement.

En pratique, toute fonction combinatoire est représentée par des portes logiques simples ET, OU, INV. Pour utiliser le moins de portes possible, on passe par la table de KARNAUGH pour simplifier la fonction.

1.1 Additionneur 1 bit en logique cablée

On souhaite réaliser un additionneur 1 bit simple.

1. Compléter la table de vérité de l'additionneur :



a	b	co	s
0	0		
0	1		
1	0		
1	1		

FIGURE 1.1 – Entité d'un additionneur 1 bit + table de vérité

a	b	co	s
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

TABLE 1.1 – Table de vérité complétée de l'additionneur 1 bit

2. En déduire l'équation logique des sorties s et co.
   On en déduit que $s = \overline{a}.b + a.\overline{b}$ et que $co = a.b$
3. Réaliser le montage pratique en utilisant des portes logiques NAND (CD4011). Voir la datasheet en Annexe pour réaliser le schéma.

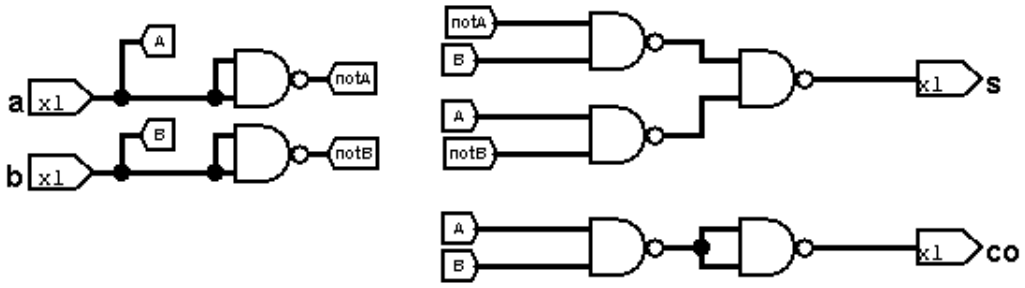


FIGURE 1.2 – Schéma de l'additionneur à base de portes NAND

4. Tester le fonctionnement pour les 4 combinaisons des entrées.

1.2 L'additionneur 1 bit

La réalisation pratique se fera maintenant sur un composant programmable de type FPGA. On utilisera le logiciel Logisim-Evolution pour décrire le circuit ou fournir la table de vérité. La table de vérité pourra être traduite en portes logiques par l'outil de synthèse. L'implémentation va générer un fichier de configuration qui, une fois chargé dans le composant, va générer les portes logiques nécessaires et les connecter comme il se doit pour réaliser l'additionneur.

1.2.1 Création d'un projet

— Démarrer le logiciel Logisim-Evolution. Le logiciel s'ouvre en créant un nouveau projet qu'il faut sauvegarder.
— *File* > *Enregistrer*

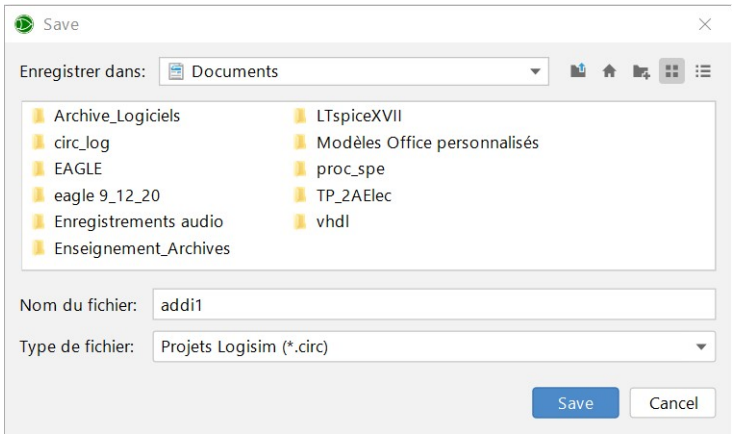


FIGURE 1.3 – Création du Projet

1.2.2 Création d'un nouveau circuit (composant)

— Renommer le circuit *main* en *addi1* : *Propriétés >> Nom du circuit >> addi1*

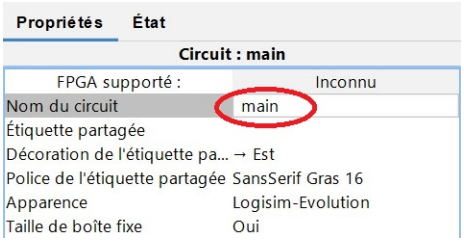


FIGURE 1.4 – Création du circuit

— Dessiner le schéma du circuit additionneur 1 bit. Le panneau de navigation permet d'accéder à tous les composants des bibliothèques pour dessiner un circuit logique. Par exemple, dans la bibliothèque *Portes logiques*, se trouvent les portes NAND nécessaires.

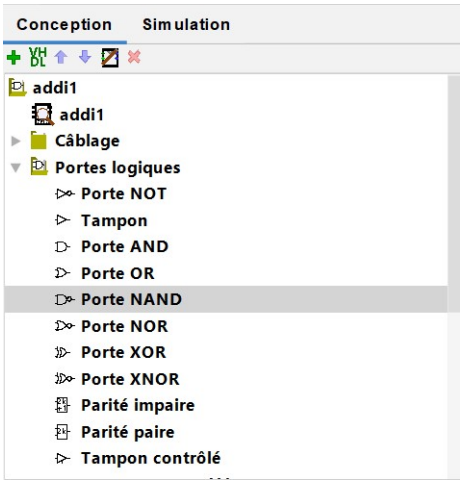


FIGURE 1.5 – Bibliothèques de Logisim

— Sélectionner et positionner les portes nécessaires.
— Puis sélectionner les broches d'entrée et de sortie dans la barre d'outils. Pour chaque entrée ou sortie, il faut indiquer son nom dans l'onglet *Etiquette*.

FIGURE 1.6 – Entrées/Sorties

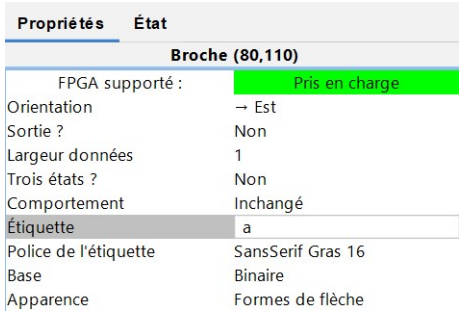


FIGURE 1.7 – Modification du nom des Entrées/Sorties

— Enfin, il faut réaliser les connectiques entre les différents éléments en sélectionnant l'outil *Câbles* dans la barre d'outils.

FIGURE 1.8 – Outil Cablage

Remarque : il est possible d'adapter le nombre d'entrées des portes logiques dans l'onglet *Nombre d'entrées*.

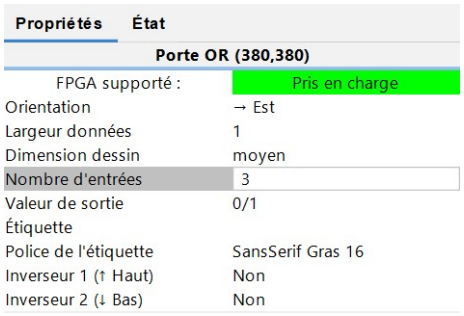


FIGURE 1.9 – Changement du nombre d'entrées d'une porte logique

1.2.3 Vérification du fonctionnement du circuit (Simulation sous Logisim-Evolution)

— Sélectionner l'onglet simulation, et vérifier que le simulation tourne en continu (premier bouton).

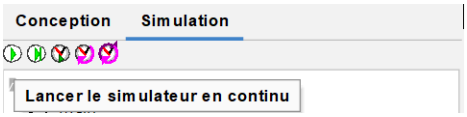


FIGURE 1.10 – Onglet Simulation

— Sélectionner dans la barre d'outils l'onglet qui permet de changer les valeurs des entrées dans le circuit. Vérifier la table de vérité en changeant les valeurs des entrées a et b.

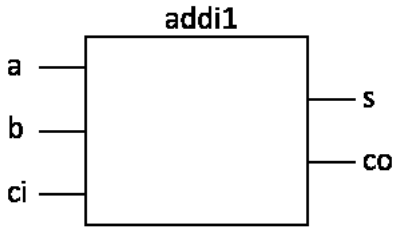
FIGURE 1.11 – Changement des valeurs d'entrée

— Fixer les valeurs des entrées pour tester toutes les combinaisons possibles des entrées a et b et valider le fonctionnement du circuit.

1.2.4 Création du circuit à partir de la table de vérité

On souhaite maintenant réaliser un additionneur complet à la place de l'additionneur simple.

— Supprimer le circuit précédent dessiné dans *addi1*.



ci	a	b	co	s
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

FIGURE 1.12 – Additionneur 1 bit complet

— Compléter la table de vérité de l'additionneur complet.

— Logisim propose la création de circuit logique à partir d'une table de vérité. On renseigne la table de vérité et on laisse à Logisim le soin de générer et dessiner le circuit correspondant. Pour cela : *Projet >> Analyser le circuit*

— Dans l'onglet *Entrées & Sorties* : Renseigner les entrées et les sorties.

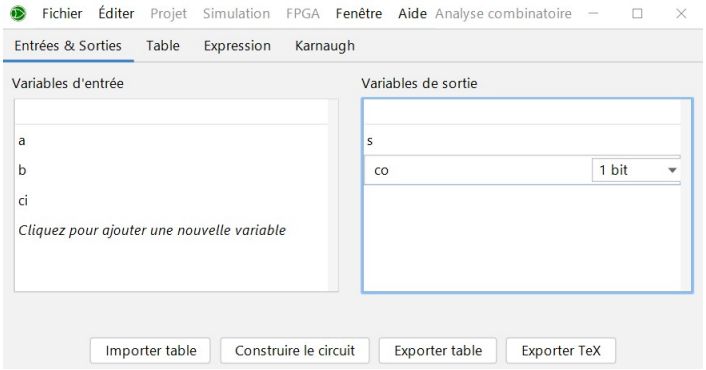


FIGURE 1.13 – Entrées et Sorties de la table de vérité

— Dans l'onglet Table : Ecrire la table de vérité. Les combinaisons des entrées sont déjà écrites.

FIGURE 1.14 – Ecriture de la table de vérité

— Vérifier l'équation logique dans l'onglet *Expression* de cette fenêtre ainsi que les tables de Karnaugh des deux sorties dans l'onglet *Karnaugh*.

— Générer le circuit : *Construire le circuit >> OK*

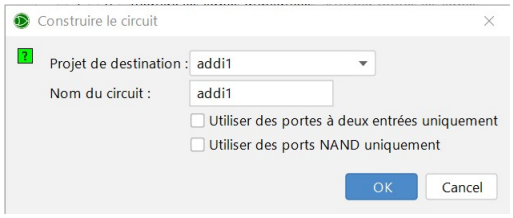


FIGURE 1.15 – Génération du circuit

— Simuler le circuit et vérifier son fonctionnement

1.2.5 Implémentation du circuit sur maquette Basys3

On souhaite implémenter ce circuit sur FPGA.

— Choisir un dossier dans lequel sera généré le projet : *Fichier >> Préférences*

— Se placer dans l'onglet *FPGA Commander parametres* et indiquer l'emplacement du projet.

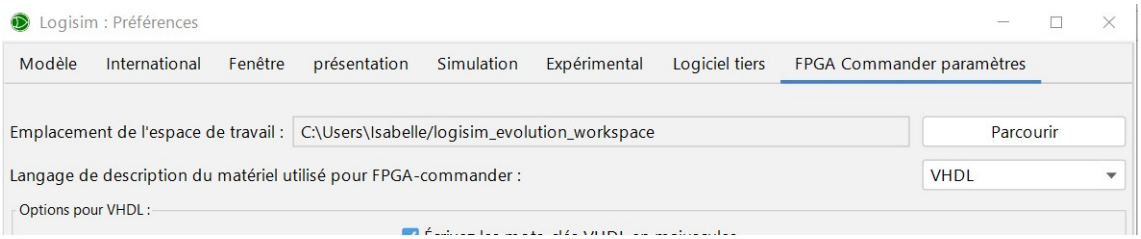


FIGURE 1.16 – Choix de l'emplacement de l'espace de travail

— Sélectionner *FPGA >> Synthétiser et télécharger.*

— Choisir la *carte Cible : BASYS3*.

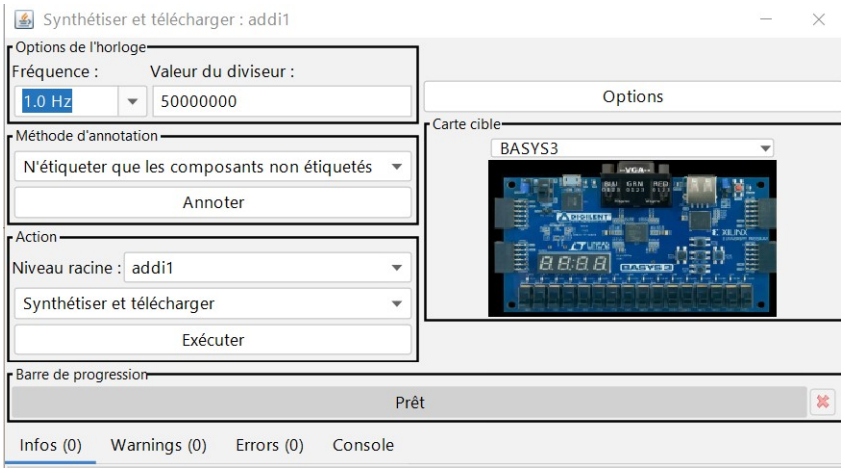


FIGURE 1.17 – Synthèse sur BASYS3

— Cliquer sur *Annoter*.
— Dans le cadre *Actions*, sélectionner
  — le niveau racine correct (*addi1*),
  — *Synthétiser et télécharger*
— Cliquer sur *Exécuter*.
— Réaliser le mappage : Associer les noms d'entrées et de sorties aux périphériques d'entrée et de sorties
  — Entrées -> Switchs
  — Sorties -> LEDs

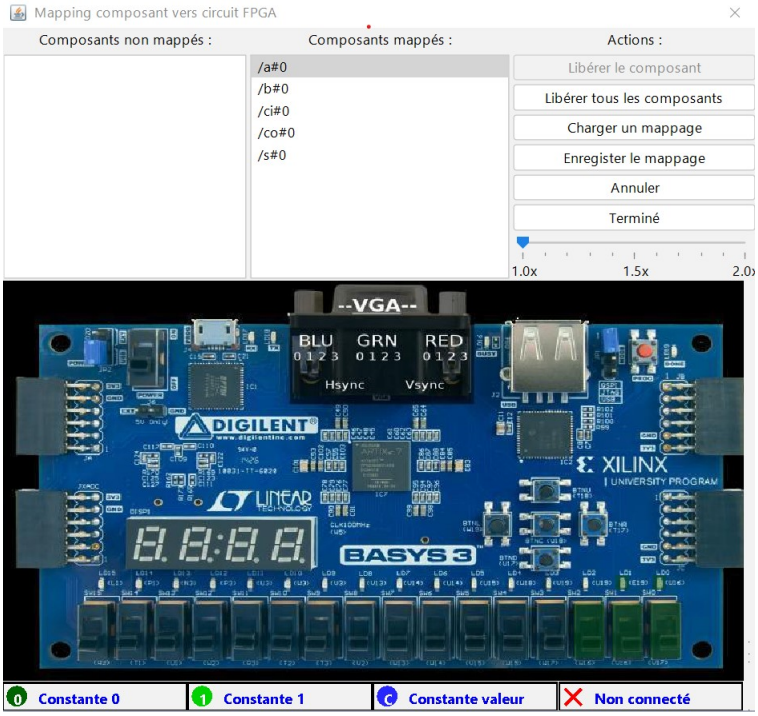


FIGURE 1.18 – Synthèse et mappage sur BASYS3

— Cliquer sur *Terminé.*
— Connecter la carte BASYS3 puis télécharger :

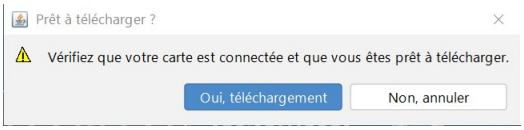


FIGURE 1.19 – Téléchargement sur BASYS3

— Tester sur la maquette.

1.2.6 Visualisation de l'implémentation avec Vivado.

— Ouvrir le projet *vp.xpr* généré par Logisim dans le dossier sélectionné précédemment. Le projet se trouve dans *Sandbox >> vp*. Le projet s'ouvre avec le logiciel Vivado.
— Dans Vivado, choisir *RTL Analysis >> Open Elaborated Design.*
— *Window >> I/O Ports* pour voir les broches d'entrées/sorties choisies sur la maquette.

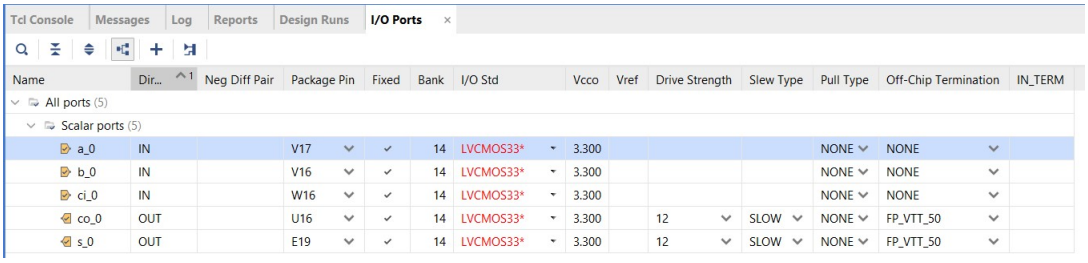


FIGURE 1.20 – Visualisation des broches sur Vivado

— Identifier dans la colonne *Package Pin* les noms des broches du FPGA choisies. Vérifier la correspondance sur la maquette BASYS3 ainsi que sur le schéma de la datasheet de la BASYS3 (voir Annexe).
— Cliquer sur *Window >> Package* pour visualiser la localisation des broches sur le boitier du FPGA.
— Cliquer sur *Implementation >> Open Implemented Design* pour visualiser l'implémentation réelle sur le composant FPGA.
— Dans l'onglet *Netlist*, sélectionner l'ensemble des Nets.

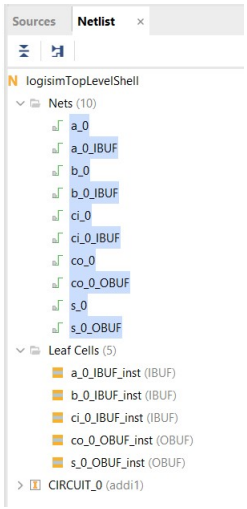


FIGURE 1.21 – Sélection des connections

— Analyser dans la fenêtre *Device* les différents éléments
  — Blocs *Inputs/Outputs (IOB)* : Pads, Buffers
  — Blocs *logiques configurables (CLB)* : Slices et LUTs : Blocs de mémoire dans lequel est implémentée la fonction logique
— Vérifier les tables de vérité implémentées dans les LUTs : Sélectionner la LUT puis *Propriétés >> Truth Table*.

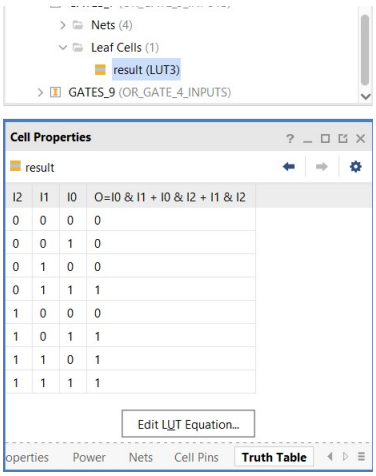


FIGURE 1.22 – Visaulisation de la table de vérité d'une LUT

Chapitre 2

L'additionneur 5 bits et décodeur 7 segments

2.1 Additionneur 5 bits

— L'additionneur 5 bits est construit avec 5 additionneurs 1 bit en cascade. Vu de l'extérieur, il ressemble à cela :

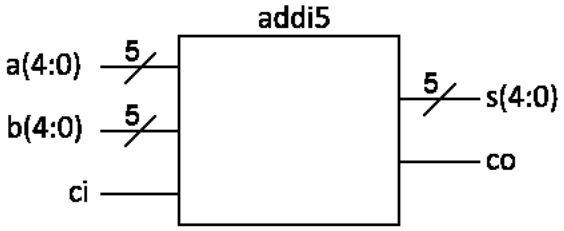


FIGURE 2.1 – Entité de l'Additionneur 5 bits

— Structure interne :

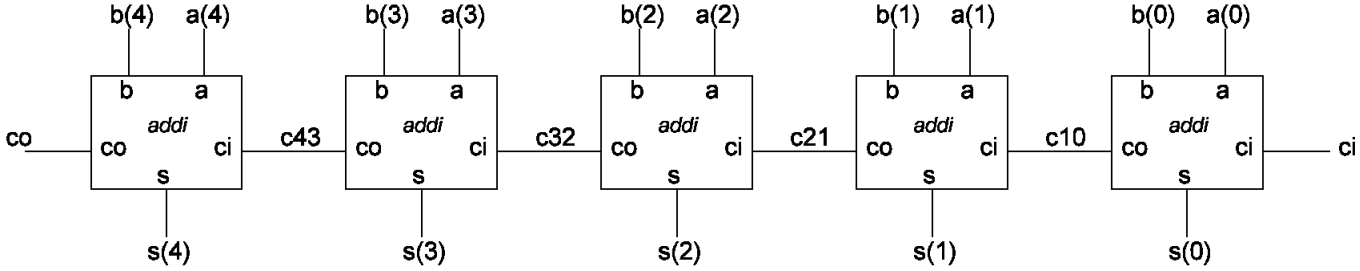


FIGURE 2.2 – Structure interne (architecture) de l'addionneur 5 bits

— Créer nouveau projet : *add_sous*.

— Récupérer le circuit *addi1* réalisé au TP précédent : *Projet >> Charger une librairie>> Librairie logisim-Evolution*

— Ajouter la librairie du premier TP : *addi1.circ*

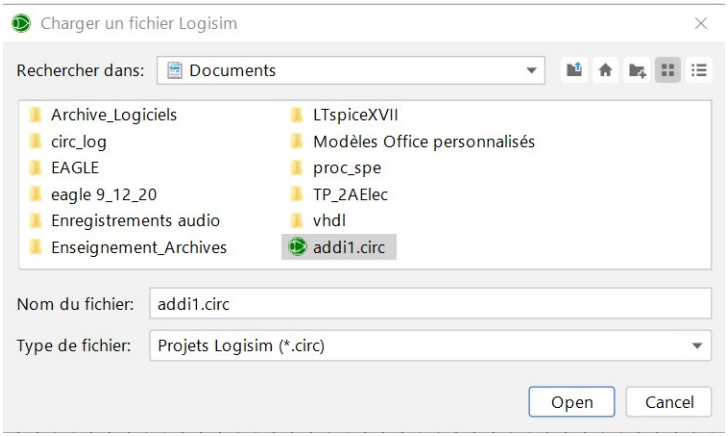


FIGURE 2.3 – Ajout d'une librairie

— La librairie *addi1* est maintenant ajoutée à la liste des bibliothèques disponibles dans Logisim.

— Renommer le main en *addi5*.

— Câbler l'additionneur 5 bits en utilisant le circuit *addi1*.

— Les entrées sont des bus de taille 5 bits, ce paramètre est changé dans l'onglet *Largeur données*.

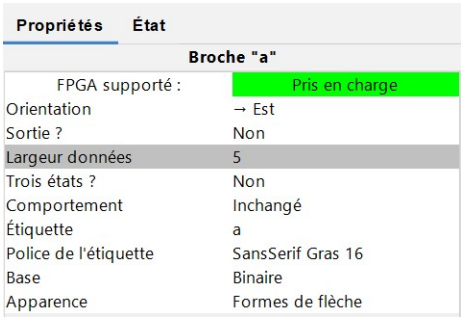


FIGURE 2.4 – Changement de la taille du bus d'entrée

— Les bus d'entrée et de sortie doivent être découpés bit à bit pour s'adapter à l'entrée du circuit *addi1*. Pour cela, on utilise un *répartiteur (Splitter)* à sélectionner dans la librairie *Câblage*.

— Pour ce répartiteur, indiquer son orientation, le nombre de terminaisons et la largeur du faisceau dans les propriétés.

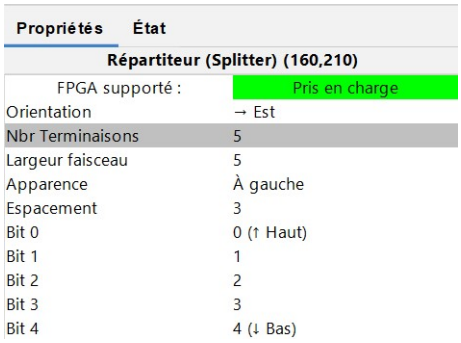


FIGURE 2.5 – Changement des propriétés du répartiteur

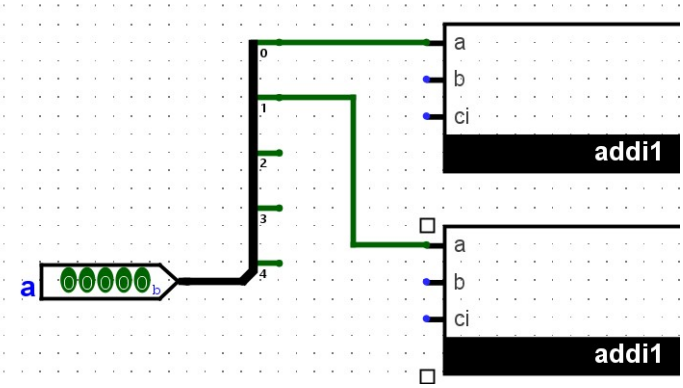


FIGURE 2.6 – Connexion du répartiteur avec le bus et le circuit addi1

— Il est possible de changer la base pour les valeurs d'entrées ou de sortie

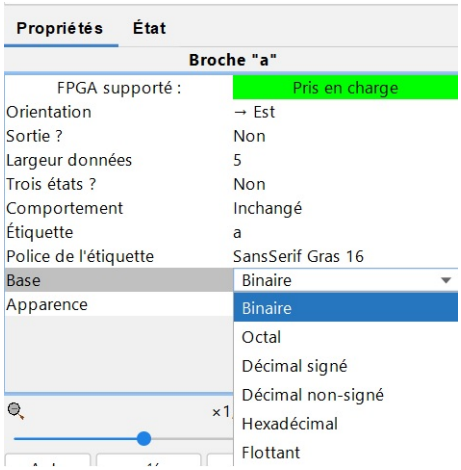


FIGURE 2.7 – Changement de base pour la représentation des nombres

— Simuler le circuit *addi5* et vérifier son fonctionnement pour quelques combinaisons des entrées.
— Implémenter et tester sur la maquette.

2.2 L'additionneur-soustracteur 5 bits

— Le soustracteur 5 bits est réalisé avec un additionneur 5 bits. Comme $A - B = A + (-B)$, on fait le compément à 2 de B, c'est à dire (le complément à 1) + 1. B est inversé pour complémenter à 1, ci=op est mise à 1 pour faire +1.

— La figure suivante donne la vue externe (l'entité) de l'additionneur soustracteur, l'entrée op permet de sélectionnet l'addition (op = 0) ou la soustraction (op = 1) :

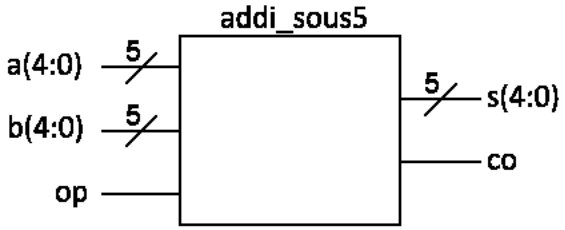


FIGURE 2.8 – Additionneur-Soustracteur 5 bits

— Structure interne (architecture) de l'additionneur soustracteur 5 bits :

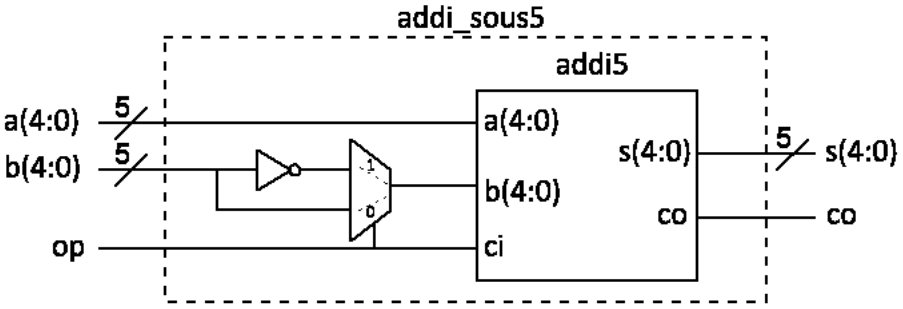


FIGURE 2.9 – Architecture de l'additionneur-soustracteur 5 bits

— Réaliser le circuit *addi_sous5*.

— Simuler le circuit et vérifier son fonctionnement pour quelques combinaisons des entrées.

— Implémenter et tester sur la maquette.

2.3 Transcodage et affichage

2.3.1 Introduction

Sur la maquette BASYS3, il y a 4 afficheurs à 7 segments qui reçoivent tous les mêmes données à afficher. Heureusement, chaque afficheur possède un signal de sélection qui permet de l'allumer ou non. En faisant tourner rapidement la sélection individuelle des afficheurs, l'œil humain a l'impression de les voir tous allumés en même temps.

— Pour commencer, il faut créer un décodeur hexadécimal vers 7 segments (circuit *decodeur_hex_7seg*) qui transforme un code binaire sur 4 bits en 7 signaux qui affichent le caractère correspondant sur l'afficheur.

— Ensuite, il faudra gérer la rotation des données sur les afficheurs (circuit *mux_4affi*), ainsi que la sélection individuelle des afficheurs (entrée *sel*)

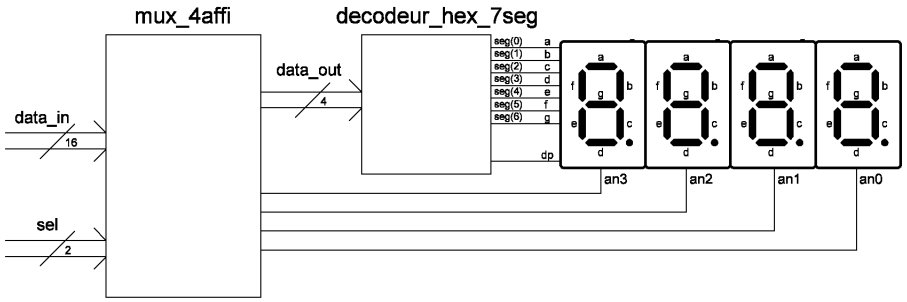


FIGURE 2.10 – Schéma de principe du transcodeur

2.3.2 Décodeur Hexa - 7 segments : circuit decodeur_hex_7seg

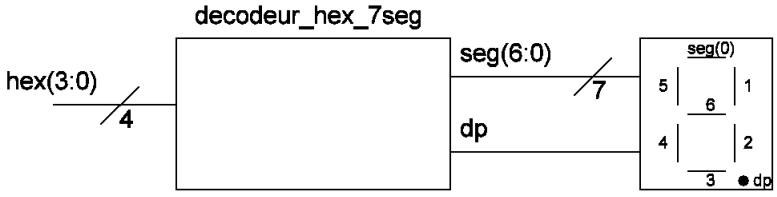


FIGURE 2.11 – Schéma de principe du décodeur héxadécimal 7 segments

— Le décodeur reçoit une entrée sur 4 bits (code hexadécimal) et fournit 8 sorties à l'afficheur (7 segments + le point décimal).

— Pour allumer une led (segment ou point décimal) de l'afficheur, il faut affecter son entrée à 1.

— Compléter la table de vérité du décodeur en considérant que le point décimal est toujours éteint.

hex(3 :0)	dp	seg(6 :0)

TABLE 2.1 – Table de vérité du décodeur héxadécimal 7 segments

— Créer un nouveau projet, nommé *affichage_multiplexe*.
— Créer un nouveau circuit nommé *decodeur_hex_7seg* dont la vue externe (entité) est représentée par la figure suivante :

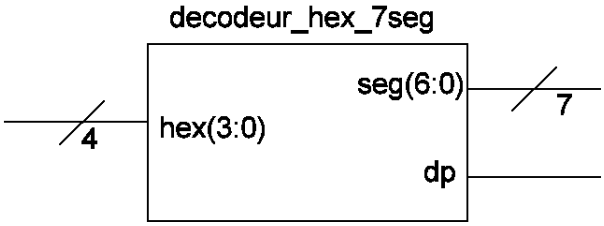


FIGURE 2.12 – Schéma de principe du décodeur hexadécimal 7 segments

— Réaliser le circuit *decodeur_hex_7seg* en renseignant sa table de vérité.
— Simuler le circuit et valider le bon fonctionnement du décodeur.
— Implémenter le circuit sur la maquette BASYS3 et tester pour toutes les combinaisons possibles. Comment fonctionnent les 4 afficheurs ?

2.3.3 Multiplexage des données : circuit mux_4affi

Pour cette application, on souhaite afficher la valeur hexadécimale des 16 switchs sur les 4 afficheurs de la maquette. Il est nécessaire de s'adresser séparément aux afficheurs par un bus de sélection. Il faut également sélectionner les 4bits de donnée que l'on souhaite afficher sur l'afficheur choisi.

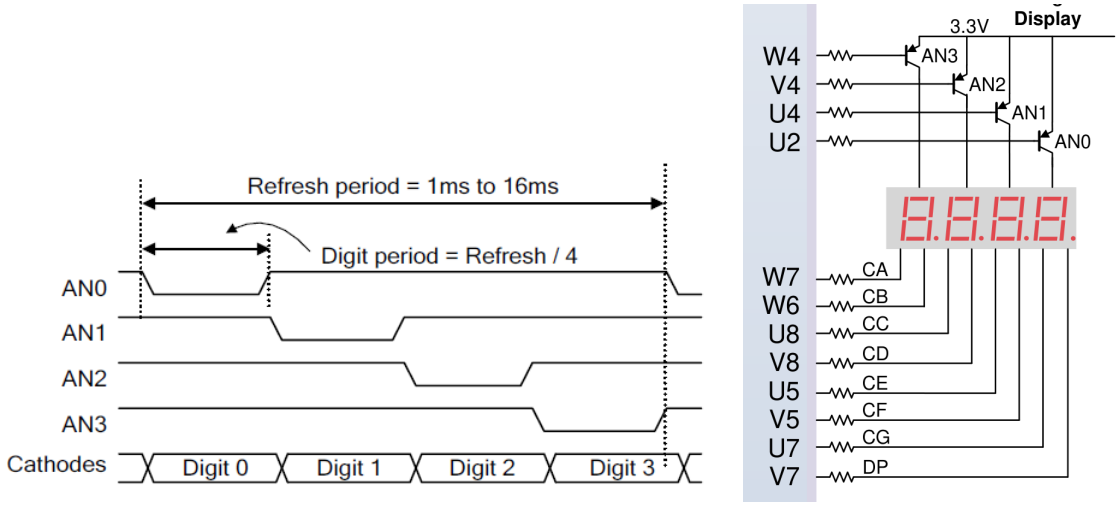


FIGURE 2.13 – Chronogramme présentant le multiplexage

— Cette étape consiste à réaliser le multiplexage des 16 signaux d'entrée nommés *data_ in(15 :0)* vers les 4 sorties *data_ out(3 :0)*. La sélection se fera par 2 fils nommés *sel(1 :0)* et le choix de l'afficheur se fait à l'aide du bus *an(3 :0)*. Le tableau suivant explique le mode de sélection.

sel(1 :0)	afficheur sélectionné	*an(3 :0)*	*data_ out(3 :0)*
00	1	1110	data_in(3 :0)
01	2	1101	data_in(7 :4)
10	3	1011	data_in(11 :8)
11	4	0111	data_in(15 :12)

TABLE 2.2 – Multiplexage de *data_ in* et sélection de *an* en fonction de l'entrée *sel*

— Créer un nouveau circuit nommé *mux_ 4affi* dont la vue externe (entité) est représentée par la figure suivante :

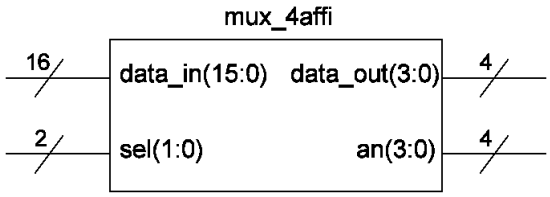


FIGURE 2.14 – Entité de mux_ 4affi

— En utilisant les multiplexeurs disponibles dans Logisim-Evolution, câbler l'architecture de l'entité *mux_ 4affi*.
— Simuler son fonctionnement.

2.3.4 Le circuit affichage complet

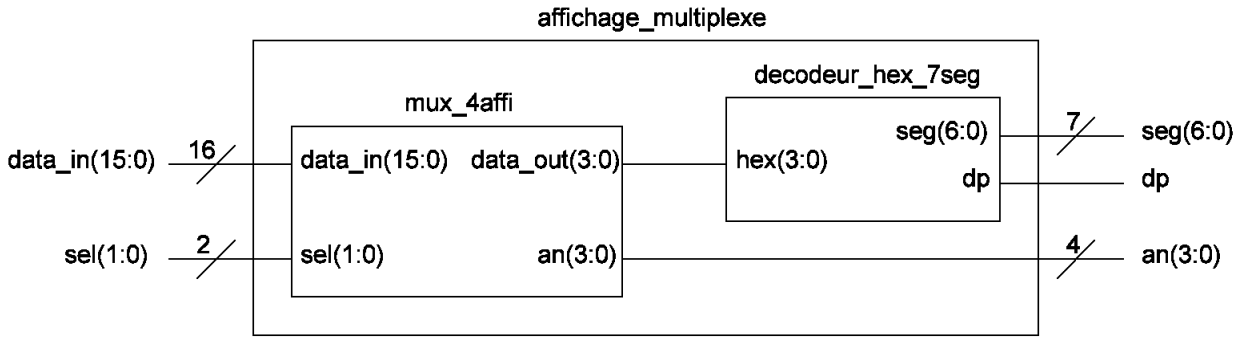


FIGURE 2.15 – Entité et architecture du circuit affichage

— Créer un nouveau circuit *affichage_multiplexe* dont l'entité et l'architecture sont représentées par la figure précédente. Câbler le circuit *affichage_multiplexe*.
— Simuler le projet.
— Programmer le composant cible et tester le fonctionnement sur la maquette. L'entrée *sel* sera réalisée par des boutons poussoirs.

Chapitre 3

Systèmes séquentiels

Les circuits logiques sont classés en deux catégories : combinatoires et séquentiels. Un circuit combinatoire se caractérise par le fait que l'état de ses sorties ne dépend que de l'état de ses entrées. C'est le cas du décodeur hexadécimal vers 7 segments étudié précédemment.

Par contre, l'état futur des sorties d'un système séquentiel dépendra de l'état des entrées mais aussi de l'état actuel des sorties. C'est le cas des feux tricolores : si le feu est actuellement vert il passera tout à l'heure à l'orange, s'il est rouge il passera au vert, ... Il passe de façon séquentielle, du vert à l'orange, de l'orange au rouge et du rouge au vert.

Pour réaliser un circuit séquentiel, il faut mémoriser l'état présent, et s'en servir pour le calcul de l'état futur avec une fonction combinatoire. Ce qui donne le schéma fonctionnel suivant, connu sous le nom de machine de Moore.

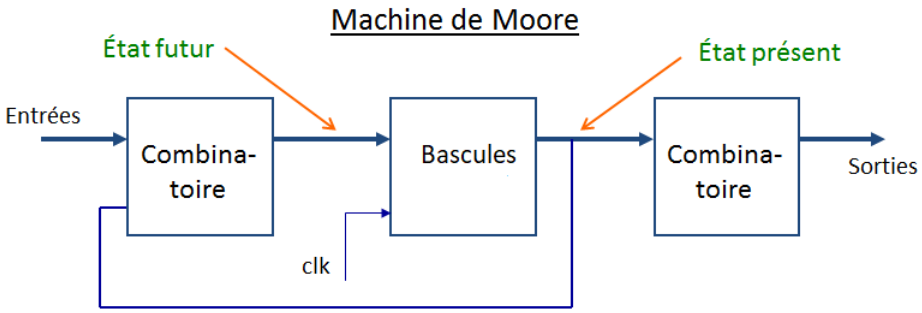


FIGURE 3.1 – Machine de Moore

3.1 Le compteur décompteur entre 1 et 6

On souhaite réaliser un circuit séquentiel qui compte et décompte de 1 à 6. Le système est compteur si $up/dn = 1$ et décompteur si $up/dn = 0$. Le schéma de principe du compteur-décompteur est représenté sur la figure suivante :

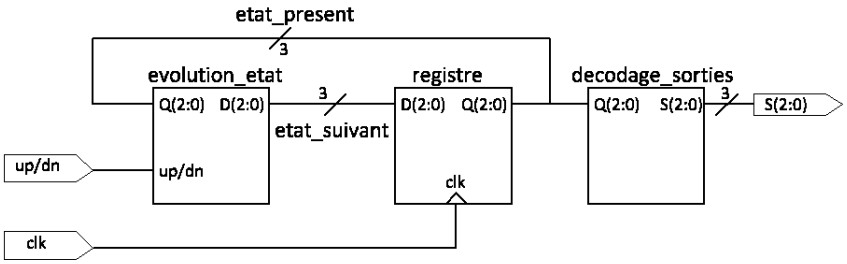


FIGURE 3.2 – Schéma de principe du compteur décompteur

— Créer un nouveau projet, nommé *compteur_decompteur*.

3.1.1 Circuit evolution_etat

— Créer un nouveau circuit nommé *evolution_etat*. Reprendre la table de vérité des états du compteur de 1 à 6 établie en TD, la compléter en ajoutant une entrée nommée *up/dn* au système. Réaliser le circuit *evolution_etat* à partir de sa table de vérité.
— Simuler son fonctionnement.

3.1.2 Circuit registre

— Créer un nouveau circuit nommé *registre*.
— En utilisant les bascules D disponibles dans Logisim-Evolution, câbler l'architecture de l'entité *basculesD*.

3.1.3 Simulation du ciruit registre

On souhaite pour ce circuit séquentiel visualiser l'évolution des signaux et afficher les chronogrammes.
— Passer dans l'onglet *Simulation*. La barre d'outils propose 3 possibilités d'évolution de l'horloge :
  — le premier bouton permet de démarrer ou de stopper la simulation,
  — le troisième bouton permet d'activer le cadencement de l'horloge
  — le quatrième bouton permet de faire avancer l'horloge d'une demi-période
  — le cinquième bouton permet de faire avancer l'horloge sur une période.

FIGURE 3.3 – Possibilité d'évolution de l'horloge

— Activer le cadencement de l'horloge. Pour cette activation, il est demandé de sélectionner le signal qui servira d'horloge : Sélectionner le signal *clk*.

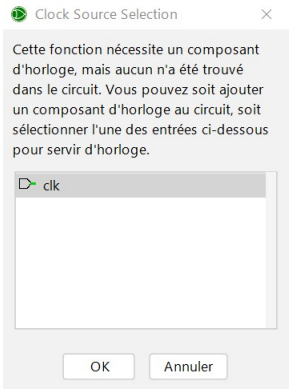


FIGURE 3.4 – Choix de l'horloge pour le circuit

— Il faut également choisir la fréquence de ce signal d'horloge : *Simulation >> Fréquence des tics >> ....*

— Tester les différentes possibilités de cadencement de l'horloge et visualiser sur le schéma l'évolution des signaux.

— On peut également visualiser le résultat de la simulation sur un chronogramme : *Simulation >> Chronogramme*

— Mettre la simulation en Pause. Dans l'onglet *Options* : Choisir *Mode continu en temps réel*

— Dans l'onglet *Chronogramme* : Choisir les signaux à afficher avec *Ajouter ou Supprimer des signaux*

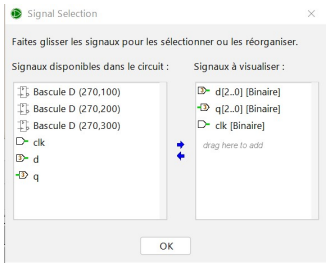


FIGURE 3.5 – Choix des signaux à afficher

— Lancer la simulation, visualiser et valider la simulation sur le chronogramme.

3.1.4 Circuit decodage_sorties

— Créer un nouveau circuit nommé *decodage_sorties.* Réaliser le circuit *decodage_sorties* à partir de sa table de vérité.

— Simuler son fonctionnement.

3.1.5 Compteur décompteur complet

— Créer un nouveau circuit nommé *compteur_decompteur.*
— Réaliser le circuit *compteur_decompteur* à partir des entités *evolution_etat, registre* et *decodage_sorties* selon le schéma précédent.
— Simuler le projet.
— Programmer le composant cible et tester le fonctionnement sur les LEDs de la maquette.
— Que constatez-vous ? Comment remédier à ce problème ?

3.1.6 Diviseur d'horloge

— Un compteur binaire à N bits peut être utilisé comme diviseur de fréquence.

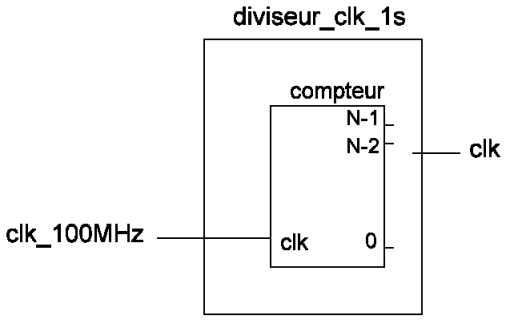


FIGURE 3.6 – Schéma de principe du diviseur

— Créer un circuit *diviseur_clk_1s.*
— Pour réaliser ce diviseur, utiliser le composant *Compteur* dans la librairie Memoire/Sequentiel de Logisim.
— Configurer et cabler le composant pour réaliser une division d'horloge par 8 en utilisant l'aide de Logisim : *Aide >> Guide Utilisateur >> Référence de la bibliothèque >> Memory Library >> Compteurs*
— Simuler et valider le fonctionnement grace aux chronogrammes.
— La fréquence de la maquette BASYS3 est 100MHz. Configurer le compteur pour avoir une fréquence de sortie de 1Hz. Utiliser dans ce cas la possibilité de fixer la valeur maximum du comptage.

3.1.7 Compteur décompteur complet avec horloge à 1Hz

— Compléter le circuit *compteur_decompteur* avec le circuit *diviseur_clk_1s.*

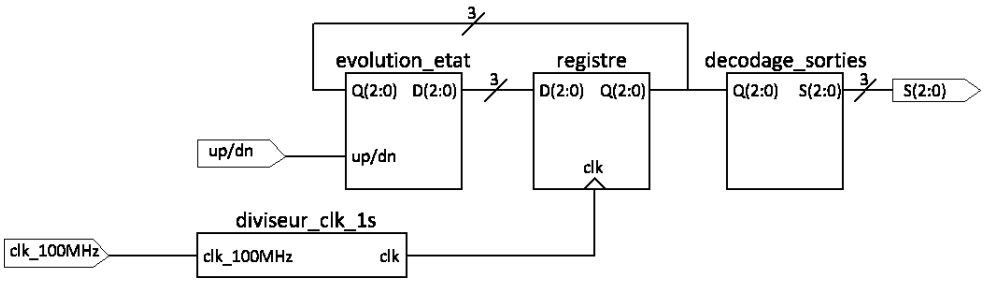


FIGURE 3.7 – Schéma de principe du compteur décompteur avec le diviseur d'horloge

— Programmer le composant cible et tester le fonctionnement sur les LEDs de la maquette.

3.2 Compteur - décompteur et Affichage

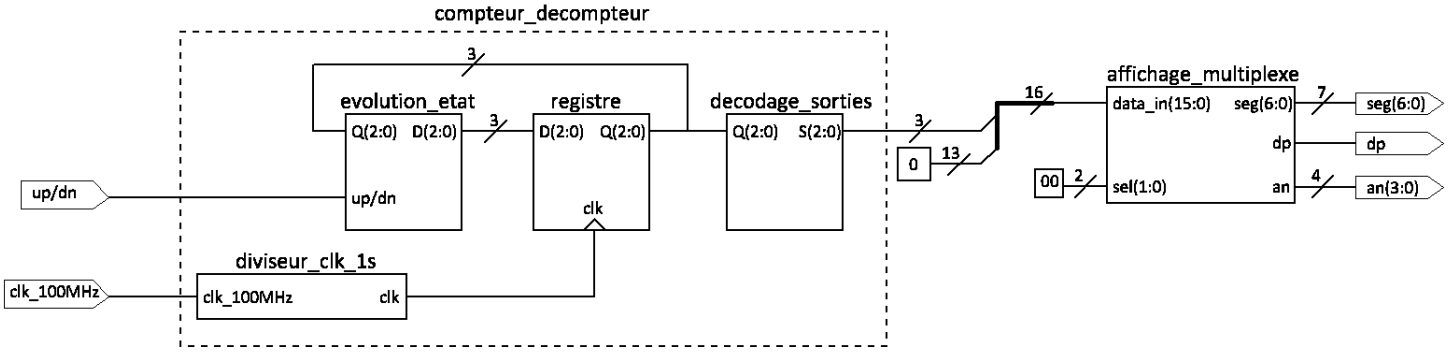


FIGURE 3.8 – Schéma de principe du compteur décompteur avec affichage

— Créer un nouveau projet *compteur_decompteur_afficheur.*

— Ajouter la librairie du TP précédent : *affichage_multiplexe.circ* dans laquelle se trouve le circuit *affichage_multiplexe.*

— Ajouter la librairie du TP précédent : *compteur_decompteur.circ* dans laquelle se trouve le circuit *compteur_decompteur.*

— Réaliser le circuit *compteur_decodeur_afficheur* à partir des entités *compteur_decompteur* et *affichage_multiplexe* selon le schéma précédent.

— Programmer le composant cible et tester le fonctionnement sur la maquette.

3.3 Retour sur le projet décodeur

On souhaite reprendre et améliorer le décodeur hexadécimal - 7 segments réalisé au TP précédent. Pour cela, on voudrait que l'entrée sel (qui permet de sélectionner l'afficheur) évolue régulièrement sans avoir à la commander par un bouton poussoir.

3.3.1 Génération des signaux de sélection : sel(1 :0)

L'évolution de l'entrée sel permet de choisir l'afficheur et les données qui le concernent. Il faut faire évoluer la valeur sel de façon suffisamment rapide pour balayer les 4 afficheurs sans que l'œil humain ne s'aperçoive du clignotement. On choisit une période de rafraichissement de l'ordre de 2.6 ms. Un diviseur de fréquence permet d'obtenir cette fréquence à partir de l'horloge de la maquette $f_{clk-100MHz} = 100MHz$.

— Sachant que la période de clk vaut $T_{clk} = 10ns$, une division de fréquence 2^{18} permet de fournir un signal de période $T_{sel} = 2^{18} \cdot 10ns = 2,6ms$.

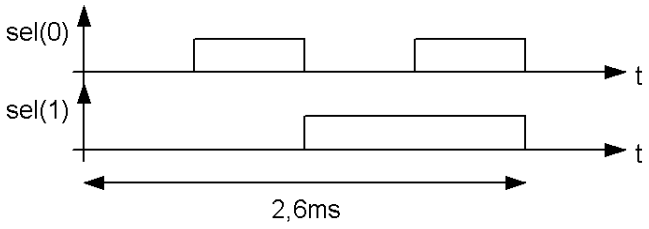


FIGURE 3.9 – Chronogramme des signaux sel

— Un compteur binaire à 18 bits (17 : 0) est utilisé comme diviseur de fréquence. La sortie sel(1) est connectée au bit 17 du compteur, la sortie sel(0) au bit 16.

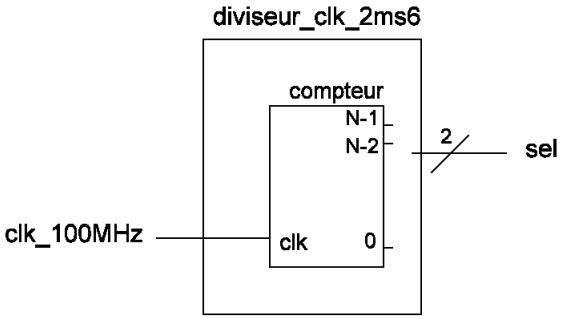


FIGURE 3.10 – Schéma de principe du diviseur

— Copier le projet *affichage_multiplexee* en *affichage_sequentiel_multiplexe* .

— Créer un nouveau circuit nommé *diviseur_clk_2ms6*.
— En utilisant un compteur disponible dans Logisim-Evolution, câbler l'architecture de l'entité *diviseur_clk_2ms6*.

3.3.2 Projet decodeur complet

— Créer un nouveau circuit *affichage_sequentiel_multiplexe* dont l'entité et l'architecture sont représentées par la figure suivante :

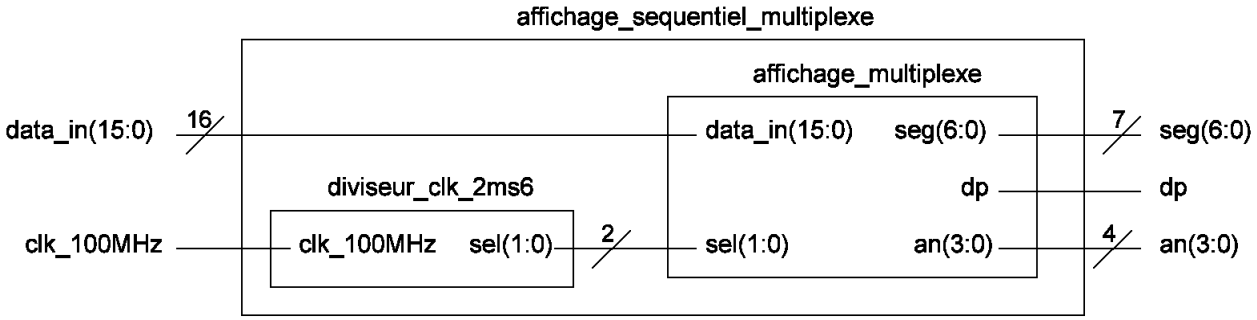


FIGURE 3.11 – Schéma de principe du décodeur complet

— Simuler le projet.
— Programmer le composant cible et tester le fonctionnement sur la maquette.

Chapitre 4

La division

4.1 Présentation

4.1.1 Principe de la division

L'opération de division ne conduit pas à un opérateur arithmétique structuré cascadable ou arborescent comme l'addition, la soustraction ou la multiplication. Il s'agit d'une opération plus complexe reposant sur les autres opérateurs. Toutefois, comme pour les autres opérateurs, le principe d'implémentation de la division s'inspire de la manière de diviser 2 nombres à la main. Un exemple est illustré en figure 4.1. En base 10, le diviseur multiplié par des puissances de 10 est soustrait au dividende autant de fois que possible. En base 2, les multiplications sont avantageusement remplacées par des décalages à gauche. L'implémentation de la division reprend cette démarche. Elle peut être réalisée de façon logicielle (soft) sur une UAL donnée ou matérielle (hard). Une réalisation hard présente l'avantage d'être optimisée et donc plus rapide. Ce TD/TP a pour objectif ce type de réalisation hard de diviseur.

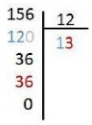


FIGURE 4.1 – Exemple de division en base 10. 156=12*10+3*12

4.1.2 Algorithme de la division

De nombreux algorithmes existent pour implémenter une division de nombres binaires. L'algorithme choisi dans ce TD/TP est l'algorithme sans restauration (Non Restoring Division Algorithm).

Les étapes de cet algorithme sont décrites ci-dessous et un exemple suit dans le tableau 4.2.

Algorithme de division sans restauration :

— Décalage des registres P et A de 1 bit vers la gauche.

— Si le contenu du registre P est :

- négatif : on additionne le contenu du registre B à P
- positif : on soustrait le contenu du registre B à P

— Si le résultat chargé dans P est négatif on met le bit de poids faible de A (A0) à « 0 » sinon à « 1 ».

— Ces étapes sont répétées N fois, N étant le format des données, ici N=4.

Opération élémentaire	P $P_4P_3P_2P_1P_0$	A $A_3A_2A_1A_0$	Commentaires
Chargement des registres	00000	**0111**	Initialisation des registres. Chargement du dividende (A) et du diviseur (B)
décalage	00000	**111**-	décalage P,A de 1 bit à gauche
-B	11110	**111**-	P≥0, on soustrait le diviseur B (P+(-B)).
A0<=0	11110	**1110**	résultat <0, A0 mis à « 0 ».
décalage	11101	**110**-	décalage P,A de 1 bit à gauche
+B	00010	**110**-	P<0, on additionne le diviseur B (P+B).
A0<=0	11111	**1100**	résultat <0, A0 mis à « 0 ».
décalage	11111	**100**-	décalage P,A de 1 bit à gauche
+B	00010	**100**-	P<0, on additionne le diviseur B (P+B).
A0<=1	00001	**1001**	résultat >0, A0 mis à « 1 ».
décalage	00011	**001**-	décalage P,A de 1 bit à gauche
-B	11110	**001**-	P≥0, on soustrait le diviseur B (P+(-B)).
A0<=1	00001	**0011**	résultat >0, A0 mis à « 1 ».
		quotient	

FIGURE 4.2 – Algorithme de la division sans restauration. Exemple de la division de 7 (0111) dans le registre A par 2 (0010) dans le registre B. Remarque : la soustraction de 2 est représentée dans le tableau par l'addition de -2 (11110). Le quotient obtenu 0011 (3) correspond au résultat de la division de 7 par 2.

4.2 Réalisation matérielle

4.2.1 Shéma de principe

L'implémentation matérielle de l'algorithme de division nécessite une structure logique avec une unité de traitement (UT) dédiée aux opérations élémentaires et une unité de contrôle (UC) chargée de la piloter. Une structure de ce type, celle utilisée dans ce TD/TP, est illustrée en figure 4.3. La demande de division, DIVstart, provient d'une structure supérieure de type coeur de CPU. L'unité de contrôle (UC) locale de la division hardware génère les signaux de contrôle pour effectuer l'algorithme de division sur l'unité de traitement (UT) dédiée. Les données sont présentées stables en entrée par le coeur de CPU qui vient chercher les résultats de la division quand le signal DIVend est actif.

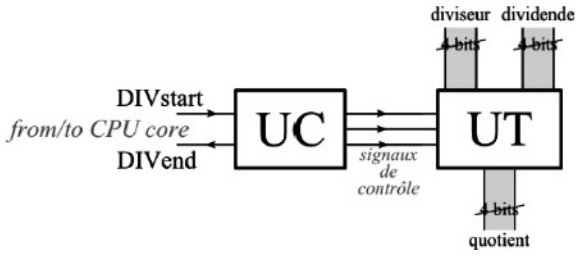


FIGURE 4.3 – Structure logique de l'algorithme de division

L'unité de traitement comprend 3 registres de données et un additionneur/soustracteur. Le registre A contient le dividende, B stocke le diviseur, P est un registre de travail.

- Le registre A possède trois commandes : **LOADA** charge le dividende, **SHIFTA** décale à gauche A d'un bit et **LOADA0** charge le bit de poids faible.
- Le registre B possède une seule commande : **LOADB** charge le diviviseur.
- Le registre de travail P possède trois commandes : **RESETP** remet à zéro P, **SHIFTP** décale à gauche P d'un bit et **LOADP** charge le résultat de l'addition-soustraction.

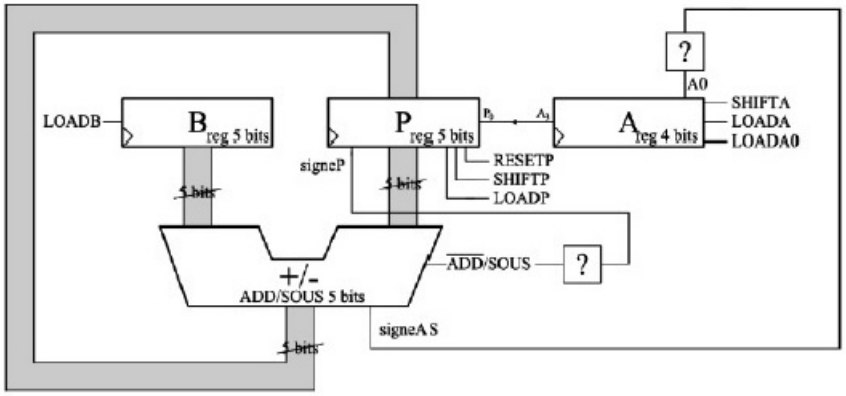


FIGURE 4.4 – Structure logique de l'unité de traitement

4.3 Réalisation de l'unité de contrôle

L'unité de contrôle doit piloter l'unité de traitement pour y effectuer l'algorithme de division. Pour l'unité de contrôle une structure de machine à états finis de type Moore est choisie.

4.3.1 Conception de l'unité de contrôle

L'opération de division sera contrôlée par une machine d'états selon le modèle de MOORE.

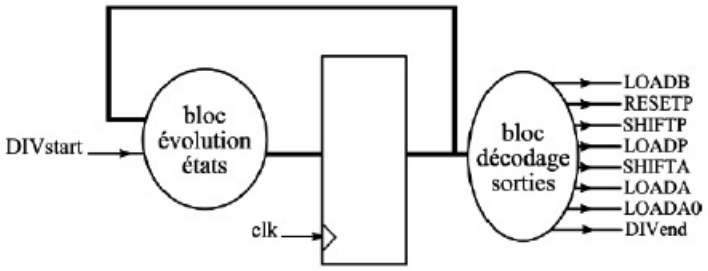


FIGURE 4.5 – Schéma de principe de l'unité de contrôle

— Le signal **DIVstart** démarre l'algorithme. Le registre conserve la valeur du quotient à la fin de l'algorithme.
— Une sortie **DIVend** indique quand le résultat est disponible.
— La période de l'horloge de cadencement, clk, de la machine d'état doit être supérieure au temps de calcul de l'additionneur/soustracteur.

Compétez le tableau suivant en vous inspirant de l'algorithme décrit en section 4.1.2.

Description	N°	loadB	resetP	shiftP	loadP	shiftA	loadA	loadA0	divend
début, attente divstart	0	0	0	0	0	0	0	0	0
charger A et B, reset P	1								
décalage	2								
charger A0 et P	3								
	4								
	5								
	6								
	7								
	8								
	9								
fin	10	0	0	0	0	0	0	0	1
début, attente divstart	0	0	0	0	0	0	0	0	0

Après analyse, des simplifications sont possibles et finalement, seuls 4 signaux de sortie pour l'unité de contrôle sont nécessaires pour piloter l'unité de traitement (figure 4.6).

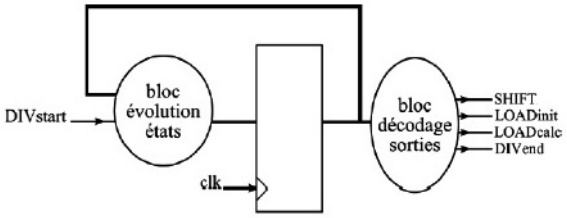


FIGURE 4.6 – Schéma de principe de l'unité de contrôle simplifiée

1. Que représentent les commandes **LOADcalc**, **LOADinit** et **SHIFT** ?
2. Dessiner le diagramme d'états et coder ses états.

4.3.2 Réalisation de la machine d'états

La réalisation proposée s'inspire directement du modèle de MOORE : 1 bloc mémoire et 2 blocs logiques combinatoires.

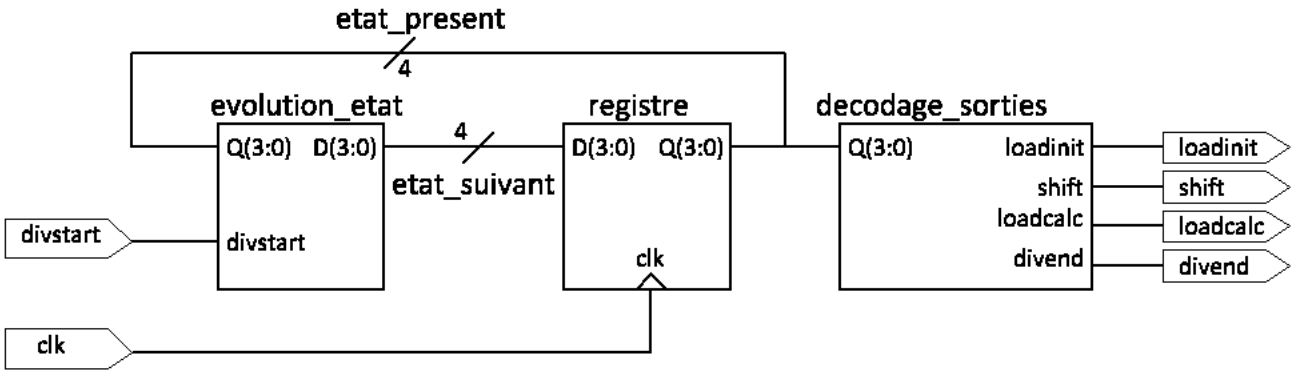


FIGURE 4.7 – Unité de contrôle

1. Créer un nouveau projet, nommé *division_arithmetique*.
2. Créer un nouveau circuit nommé *evolution_etat*. Réaliser le circuit *evolution_etat* à partir de sa table de vérité. Simuler son fonctionnement.
3. Créer un nouveau circuit nommé *registre*. En utilisant les bascules disponibles dans Logisim-Evolution, câbler l'architecture de l'entité *registre*. Simuler son fonctionnement.
4. Créer un nouveau circuit nommé *decodage_sorties*. Réaliser le circuit *decodage_sorties* à partir de sa table de vérité. Simuler son fonctionnement.
5. Créer un nouveau circuit nommé *UC*. Réaliser le circuit *UC* à partir des entités *evolution_etat*, *registre* et *decodage_sorties* selon le schéma précédent.
6. Simuler son fonctionnement.

4.4 L'unité de traitement

L'unité de traitement comporte un bloc combinatoire (l'additionneur) et trois registres A sur 4bits, B sur 5bits et P sur 5bits.

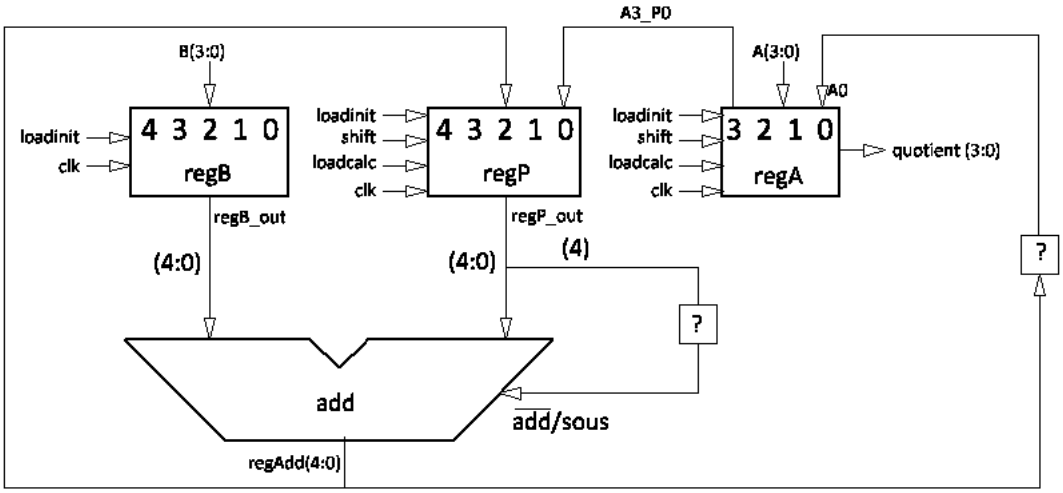


FIGURE 4.8 – Schéma de principe de l'unité de traitement

1. Créer un nouveau circuit nommé $regB$. Le bit de poid fort de $regB\_out$ sera toujours à 0.
2. Créer un nouveau circuit nommé $regP$.
3. Créer un nouveau circuit nommé $regA$.
4. Créer un nouveau circuit nommé UT.
   (a) Quelle porte logique se trouve dans les carrés contenant un point d'interrogation ?
   (b) Réaliser le circuit UT à partir des entités $regA$, $regB$, $regP$ et add_sous5 selon le schéma précédent.
   (c) Simuler son fonctionnement (pensez à piloter la clk manuellement).

4.5 La division

La division, représentée par la figure suivante, consiste à assembler les 2 unitées créées précédemment.

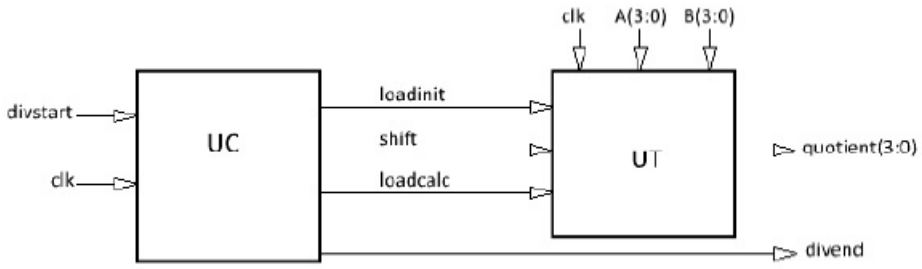


FIGURE 4.9 – Schéma de principe du montage division

1. Créer un nouveau circuit nommé *division_arithmetique* dont l'entité est représentée par le schéma précédent.
2. Réaliser le circuit *division_arithmetique* à partir des entités UC et UT selon le schéma précédent.
3. Simuler son fonctionnement.
4. Programmer le composant cible et tester le fonctionnement sur la maquette.

4.6 La division avec affichage

— Créer un nouveau projet *division_arithmetique_afficheur.*

— Ajouter la librairie du TP précédent : *affichage_sequentiel_multiplexe.circ* dans laquelle se trouve le circuit *affichage_sequentiel_multiplexe.*

— Ajouter la librairie du TP : *division_arithmetique.circ* dans laquelle se trouve le circuit *division_arithmetique.*

— Réaliser le circuit *division_arithmetique_afficheur* à partir des entités *division_arithmetique* et *affichage_sequentiel_multiplexe* qui permet d'afficher A, *1* et B sur les trois afficheurs de gauche et *quotient* sur l'afficheur de droite.

— Programmer le composant cible et tester le fonctionnement sur la maquette.

Chapitre 5

Le contrôleur VGA

Dans ce dernier TP, il vous est demandé d'implémenter un circuit qui génère des signaux VGA pour dessiner un écran de couleur unie (4096 choix de couleurs) sur votre moniteur dans une résolution de 800×600 (72Hz). Les documents suivants fournissent des descriptions détaillées du fonctionnement des signaux VGA et quelques conseils sur la conception du contrôleur VGA.

5.1 Présentation du VGA

VGA signifie Video Graphics Array. Initialement, il se réfère spécifiquement au matériel d'affichage introduit pour la première fois avec l'ordinateur IBM® PS/2 en 1987. Avec son utilisation généralisée, il fait désormais généralement référence aux normes d'affichage analogiques des ordinateurs (définies par VESA®), le connecteur DB-15 (généralement connu sous le nom de connecteur VGA).

Les normes d'affichage d'ordinateur analogique sont spécifiées, publiées, protégées par copyright et vendues par l'organisation VESA (www.vesa.org). Les informations de synchronisation utilisées dans ce TP sont un exemple de la façon dont un moniteur VGA peut être piloté dans une résolution de 800 × 600.

Un connecteur DB-15 est un connecteur subminiature D à 15 broches à trois rangées (du nom de leur blindage métallique en forme de D). Le nom de chaque broche est illustré à la figure 5.1 . Nous nous concentrerons uniquement sur les 5 signaux suivant : Red, Grn, Blue, HS et VS. Red, Grn et Blue sont trois signaux analogiques qui spécifient la couleur d'un point sur l'écran, tandis que HS et VS fournissent une référence de position de l'endroit où le point doit être affiché sur l'écran. En pilotant correctement ces cinq signaux conformément à la spécification de synchronisation VGA, nous pouvons afficher tout ce que nous voulons sur n'importe quel moniteur. Pour comprendre comment ces signaux doivent être pilotés, nous devons examiner le fonctionnement réel de nos moniteurs.

Créer un nouveau circuit nommé $regA$.

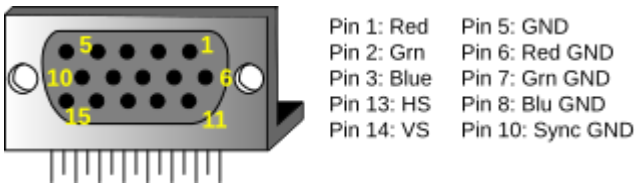


FIGURE 5.1 – Brochage d'un connecteur DB-15 (VGA)

5.2 Comment fonctionnent nos moniteurs ?

Les écrans VGA basés sur CRT (cathode-ray tube) utilisent des faisceaux d'électrons mobiles modulés en amplitude (ou rayons cathodiques) pour afficher des informations sur un écran recouvert de phosphore. Les écrans LCD utilisent un réseau de commutateurs qui peuvent imposer une tension sur une petite quantité de cristaux liquides, modifiant ainsi la permittivité de la lumière à travers le cristal pixel par pixel. Bien que la description suivante soit limitée aux écrans CRT, les écrans LCD ont évolué pour utiliser les mêmes synchronisations de signal que les écrans CRT (la discussion sur les « signaux » ci-dessous concerne donc à la fois les écrans CRT et les écrans LCD). Les écrans CRT couleur utilisent trois faisceaux d'électrons (un pour le rouge, un pour le bleu et un pour le vert) pour dynamiser le luminophore qui recouvre la face interne de l'extrémité d'affichage d'un tube à rayons cathodiques (voir Figure ci-dessous).

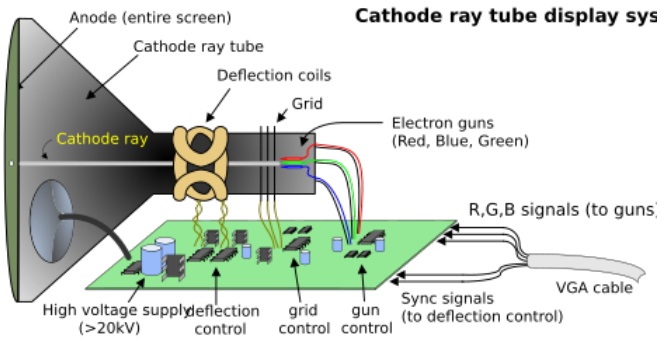


FIGURE 5.2 – Fonctionnement d'un moniteur CRT

Les faisceaux d'électrons émanent de « canons à électrons » qui sont des cathodes chauffées à pointe fine placées à proximité d'une plaque annulaire chargée positivement appelée « grille ». La force électrostatique imposée par la grille tire les rayons d'électrons sous tension des cathodes, et ces rayons sont alimentés par le courant qui circule dans les cathodes. Ces rayons de particules sont initialement accélérés vers la grille, mais ils tombent rapidement sous l'influence de la force électrostatique beaucoup plus grande qui résulte de la charge de toute la surface d'affichage recouverte de phosphore du CRT à 20 kV (ou plus). Les rayons sont focalisés en un faisceau fin lorsqu'ils traversent le centre des grilles, puis ils accélèrent pour impacter la surface d'affichage recouverte de phosphore. La surface du phosphore brille

vivement au point d'impact, et il continue à briller pendant plusieurs centaines de microsecondes après le retrait du faisceau. Plus le courant introduit dans la cathode est important, plus le phosphore brillera.

Entre la grille et la surface d'affichage, le faisceau traverse le col du CRT où deux bobines de fil produisent des champs électromagnétiques orthogonaux. Comme les rayons cathodiques sont composés de particules chargées (électrons), ils peuvent être déviés par ces champs magnétiques. Les formes d'onde de courant traversent les bobines pour produire des champs magnétiques qui interagissent avec les rayons cathodiques et les amènent à traverser la surface d'affichage selon un motif «trame», horizontalement de gauche à droite et verticalement de haut en bas, comme illustré sur la figure ci-dessous Lorsque le rayon cathodique se déplace sur la surface de l'affichage, le courant envoyé aux canons à électrons peut être augmenté ou diminué pour modifier la luminosité de l'affichage au point d'impact du rayon cathodique.

Les informations ne sont affichées que lorsque le faisceau se déplace dans le sens « avant » (de gauche à droite et de haut en bas), et non pendant le temps où le faisceau est réinitialisé sur le bord gauche ou supérieur de l'écran. Une grande partie du temps d'affichage potentiel est donc perdue dans des périodes de « suppression » lorsque le faisceau est réinitialisé et stabilisé pour commencer un nouveau passage d'affichage horizontal ou vertical. La taille des faisceaux, la fréquence à laquelle le faisceau peut être tracé sur l'écran et la fréquence à laquelle le faisceau d'électrons peut être modulé déterminent la résolution de l'affichage.

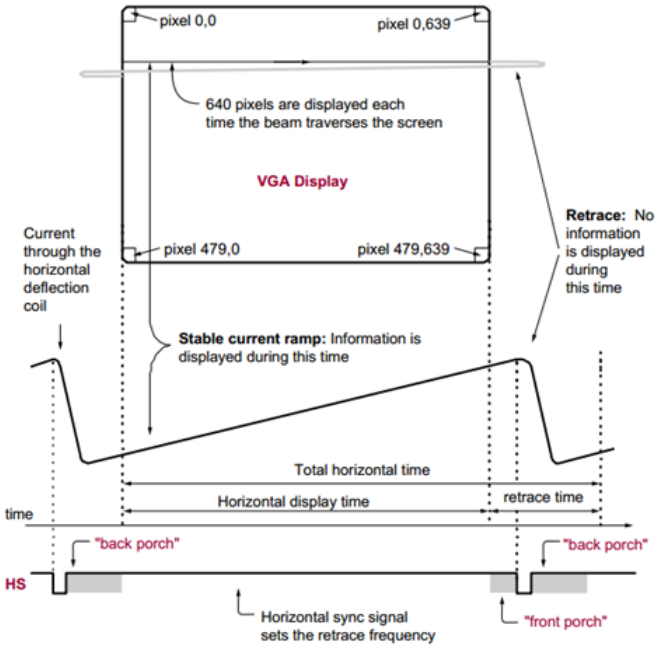


FIGURE 5.3 – Fonctionnement de l'affichage sur un moniteur CRT

5.3 Spécification de synchronisation VGA

Les écrans VGA modernes peuvent prendre en charge différentes résolutions, et un circuit de contrôleur VGA dicte la résolution en produisant des signaux de synchronisation pour contrôler les modèles de trame. Le contrôleur doit produire des impulsions de synchronisation à 3,3 V (ou 5 V) pour régler la fréquence à laquelle le courant circule dans les bobines de déviation, et il doit garantir que les données vidéo sont appliquées aux canons à électrons au bon moment. Les affichages vidéo raster (Image matricielle) définissent un certain nombre de "lignes" qui correspondent au nombre de passages horizontaux que la cathode effectue sur la zone d'affichage, et un certain nombre de "colonnes" qui correspondent à une zone sur chaque ligne qui est affectée à un "élément d'image" , ou pixel. Les affichages typiques utilisent de 240 à 1200 lignes et de 320 à 1600 colonnes. La taille globale d'un affichage et le nombre de lignes et de colonnes déterminent la taille de chaque pixel.

Les données vidéo proviennent généralement d'une mémoire vidéo (VRAM) ; avec un ou plusieurs octets attribués à chaque emplacement de pixel (la carte BASYS3 utilise 12 bits (3*4bits) par pixel). Le contrôleur doit parcourir le contenu mémoire lorsque les faisceaux se déplacent sur l'écran. Il récupére et applique les données vidéo à l'écran précisément au moment où le faisceau d'électrons se déplace sur un pixel donné.

Un circuit contrôleur VGA doit générer les signaux de synchronisation HS et VS et coordonner la livraison des données vidéo sur la base de l'horloge pixel. L'horloge pixel définit le temps disponible pour afficher un pixel d'information. Le signal VS définit la fréquence de "rafraîchissement" de l'affichage, ou la fréquence à laquelle toutes les informations sur l'affichage sont redessinées. La fréquence de rafraîchissement minimale est fonction de l'intensité du phosphore et du faisceau d'électrons de l'écran, avec des fréquences de rafraîchissement pratiques comprises entre 50 Hz et 120 Hz. Le nombre de lignes à afficher à une fréquence de rafraîchissement donnée définit la fréquence de « retracement » horizontal.

5.4 Spécification de synchronisation pour 800x600@72Hz

La figure ci-dessous et le tableau ci-dessous fournissent les spécifications de synchronisation pour une résolution de 800 × 600 à une fréquence d'images de 72 Hz :

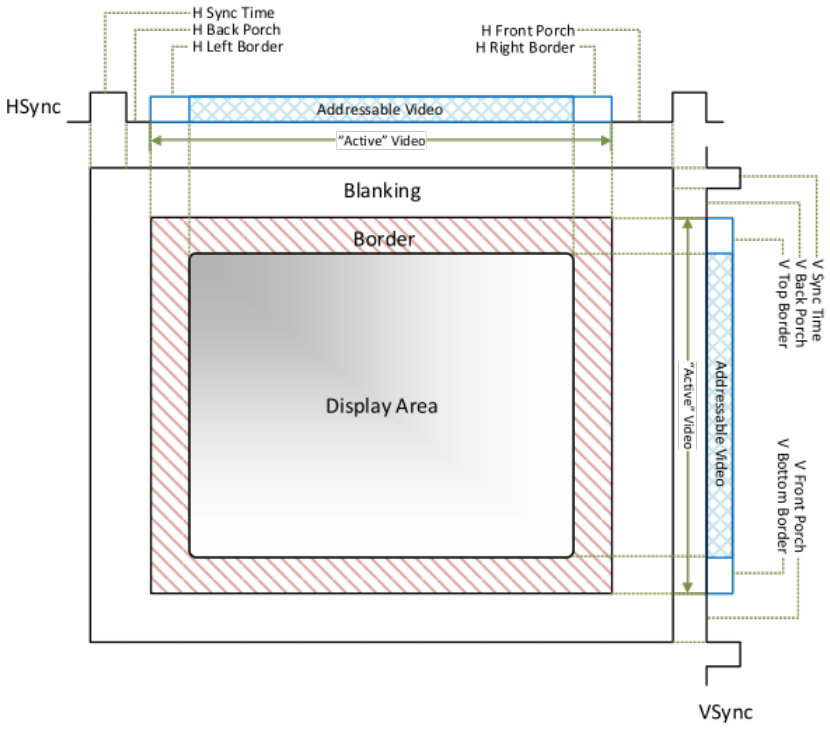


FIGURE 5.4 – Spécification des signaux HS et VS

Description	Notation	Temps	Largeur/Freq
Pixel Clk	Tpix	20 ns	50MHz
Horizontal Sync Time	Ths	2.4µs	120 Pixels
Horizontal Back Porch	Thbp	1.28µs	64 Pixels
Horizontal Front Porch	Thfp	1.12µs	56 PIxels
Horizontal Addressable Video Time	Thaddr	16µs	800 Pixels
Horizontal Adressable L/R Border	Thbd	0µs	0 Pixels
Whole Line Time	Tl	20.8µs	1040 Pixels
Vertical Sync Time	Tvs	124,8µs	6 Lines
Vertical Back Porch	Tvbp	478.4 µs	23 Lines
Vertical Front Porch	Tvfp	769.6 µs	37 Lines
Vertical Addressable Video Time	Tvaddr	12.48 ms	600 Lines
Vertical Adressable L/R Border	Tvbd	0µs	0 Lines
Whole Frame Time	Tf	13.8528 ms	666 Lines

TABLE 5.1 – Spécifications des timings

5.5 Astuces

Tout d'abord, vous avez besoin d'un diviseur d'horloge pour générer l'horloge pixel, qui fournit une référence temporelle aux signaux HS et VS. La fréquence d'horloge des pixels est de 50 MHz dans la spécification.

Deuxièmement, vous avez besoin de deux compteurs, un compteur (compteur horizontal) pour compter les pixels dans chaque ligne et un autre compteur (compteur vertical) pour compter les lignes dans une trame. Le compteur horizontal doit se réinitialiser lorsqu'il atteint la fin de la ligne (1039 dans ce cas), et lorsqu'il se réinitialise, il doit fournir un signal à l'entrée Enable du compteur vertical afin que le compteur vertical puisse ajouter 1 lorsqu'une nouvelle ligne commence. De même, le compteur vertical doit se réinitialiser lorsqu'il atteint la fin de la trame.

Les valeurs des compteurs sont ensuite comparées aux constantes définies dans la spécification pour générer les signaux HS, VS et le signal de commande du multiplexeur.

La figure ci-dessous montre la génération HS et VS basée sur les valeurs de compteur :

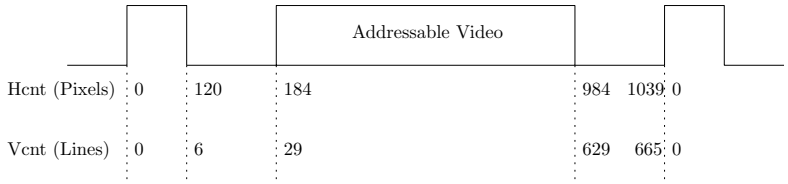


FIGURE 5.5 – Spécification des signaux HS et VS

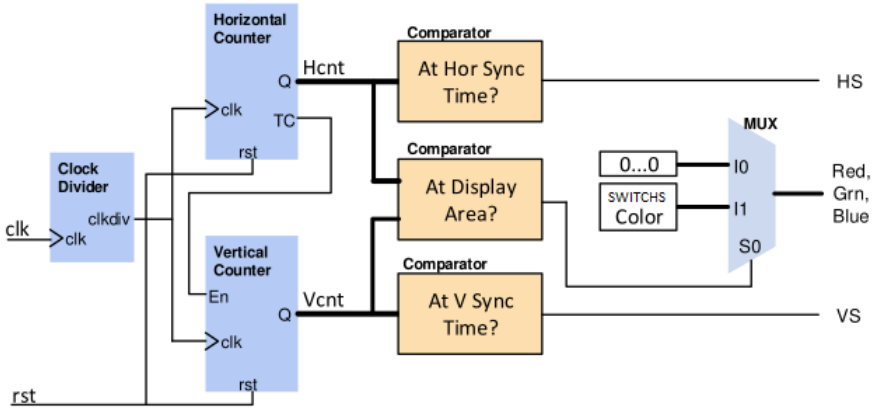


FIGURE 5.6 – Schéma fonctionnel du controleur VGA

5.6 Travail à réaliser

1. Réaliser le controleur VGA 800x600@72Hz.
2. En utilisant 12 switchs d'entrées (4 pour chaque couleur), tester sur la carte Basys3 le changement de couleur sur le moniteur (image unie).
3. Créer un nouveau circuit qui :
   (a) sera constitué d'une mémoire dans laquel vous rajouterez le fichier "image.txt" disponible sur Moodle. Utiliser le composant ROM de Logisim.

(b) viendra lire les pixels contenus dans la mémoire et les transfèrera correctement au controleur VGA .

4. Quelle image s'affiche ?

Chapitre 6

Annexes

MC14001UB, MC14011UB

UB-Suffix Series CMOS Gates

The UB Series logic gates are constructed with P and N channel enhancement mode devices in a single monolithic structure (Complementary MOS). Their primary use is where low power dissipation and/or high noise immunity is desired. The UB set of CMOS gates are inverting non–buffered functions.

- Supply Voltage Range = 3.0 Vdc to 18 Vdc
- Linear and Oscillator Applications
- Capable of Driving Two Low–power TTL Loads or One Low–power Schottky TTL Load Over the Rated Temperature Range
- Double Diode Protection on All Inputs
- Pin–for–Pin Replacements for Corresponding CD4000 Series UB Suffix Devices

MAXIMUM RATINGS (Voltages Referenced to V_{SS}) (Note 1.)

Symbol	Parameter	Value	Unit
V_{DD}	DC Supply Voltage Range	–0.5 to +18.0	V
V_{in}, V_{out}	Input or Output Voltage Range (DC or Transient)	–0.5 to V_{DD} + 0.5	V
I_{in}, I_{out}	Input or Output Current (DC or Transient) per Pin	±10	mA
P_D	Power Dissipation, per Package (Note 2.)	500	mW
T_A	Ambient Temperature Range	–55 to +125	°C
T_{stg}	Storage Temperature Range	–65 to +150	°C
T_L	Lead Temperature (8–Second Soldering)	260	°C

1. Maximum Ratings are those values beyond which damage to the device may occur.
2. Temperature Derating:
   Plastic “P and D/DW” Packages: – 7.0 mW/°C From 65°C To 125°C

This device contains protection circuitry to guard against damage due to high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltage higher than maximum rated voltages to this high–impedance circuit. For proper operation, V_{in} and V_{out} should be constrained to the range $V_{SS} \le (V_{in}$ or $V_{out}) \le V_{DD}$.

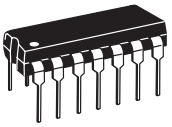
Unused inputs must always be tied to an appropriate logic voltage level (e.g., either V_{SS} or V_{DD}). Unused outputs must be left open.

ON Semiconductor

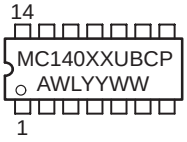
http://onsemi.com

MC14001UB
Quad 2–Input NOR Gate
MC14011UB
Quad 2–Input NAND Gate

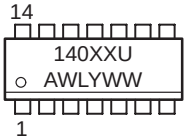
MARKING DIAGRAMS



PDIP–14
P SUFFIX
CASE 646



SOIC–14
D SUFFIX
CASE 751A

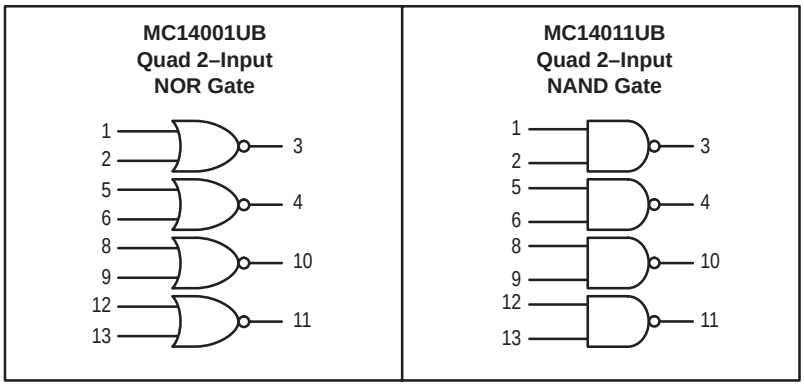


XX = Specific Device Code
A = Assembly Location
WL or L = Wafer Lot
YY or Y = Year
WW or W = Work Week

ORDERING INFORMATION

Device	Package	Shipping
MC14001UBCP	PDIP–14	2000/Box
MC14001UBD	SOIC–14	55/Rail
MC14001UBDR2	SOIC–14	2500/Tape & Reel
MC14011UBCP	PDIP–14	2000/Box
MC14011UBD	SOIC–14	55/Rail
MC14011UBDR2	SOIC–14	2500/Tape & Reel

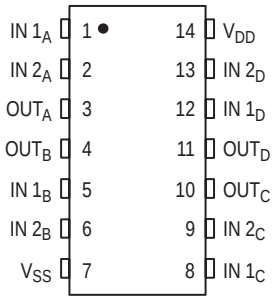
LOGIC DIAGRAMS



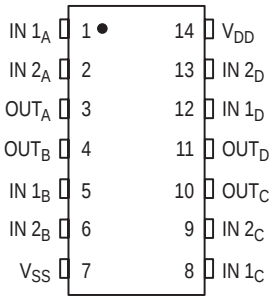
V_{DD} = PIN 14
V_{SS} = PIN 7
FOR ALL DEVICES

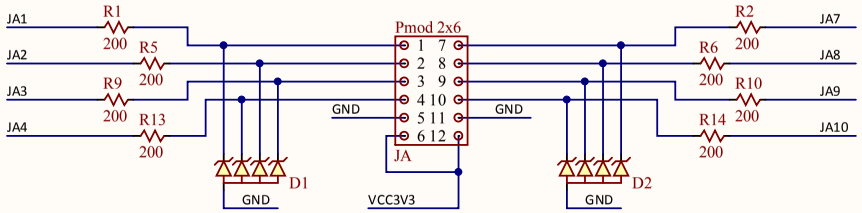
PIN ASSIGNMENTS

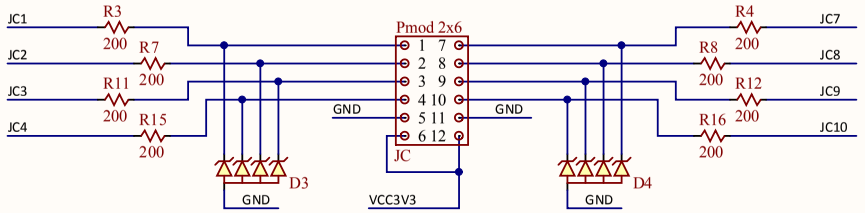
MC14001UB
Quad 2–Input NOR Gate

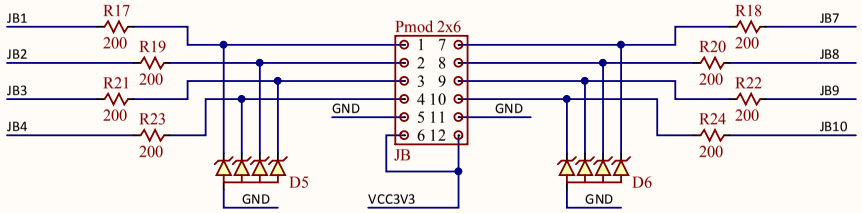


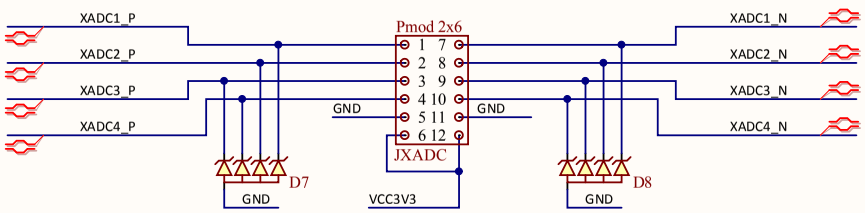
MC14011UB
Quad 2–Input NAND Gate

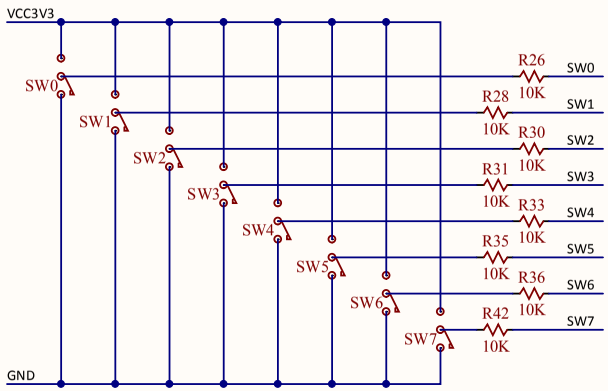


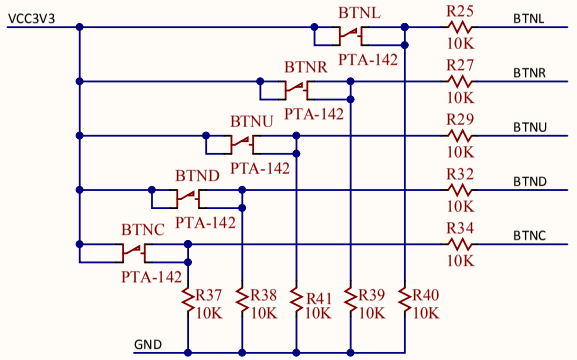


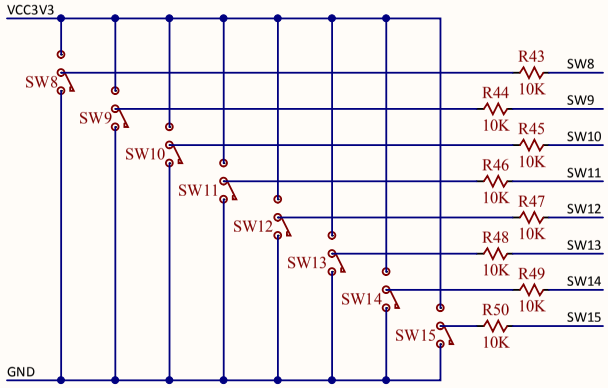










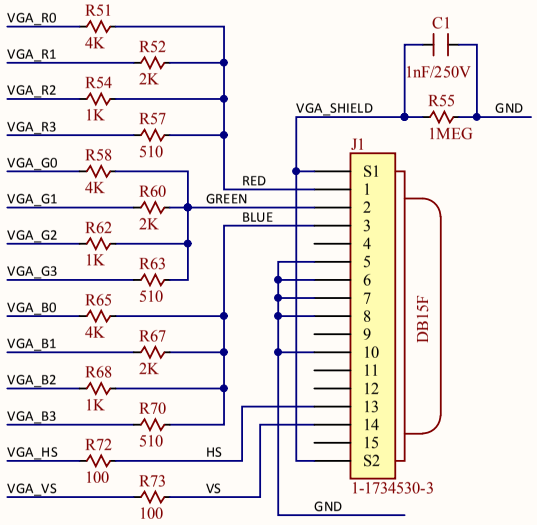


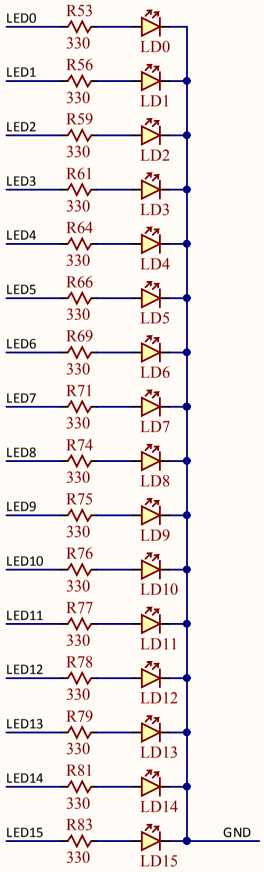
F1 Foot
F2 Foot
F3 Foot
F4 Foot

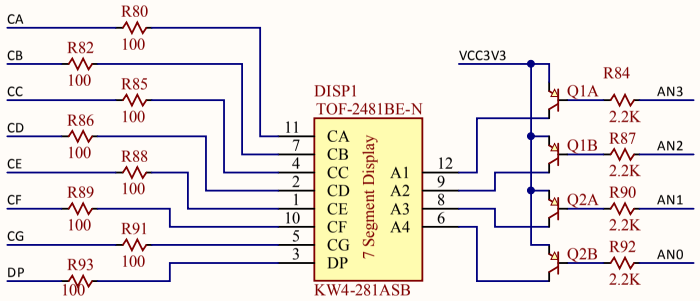
Digilent Inc. | CE | ROHS | Chinese ROHS | Basys 3 | XUP | Linear Tech

Title	Basys 3
Rev	C.0
Copyright	2014
Circuit	PMOD, I/O
Doc#	500-183
Engineer	MTA
Author	GMA
Date	5/21/2014
Sheet#	1 out of 8

DIGILENT® BEYOND THEORY

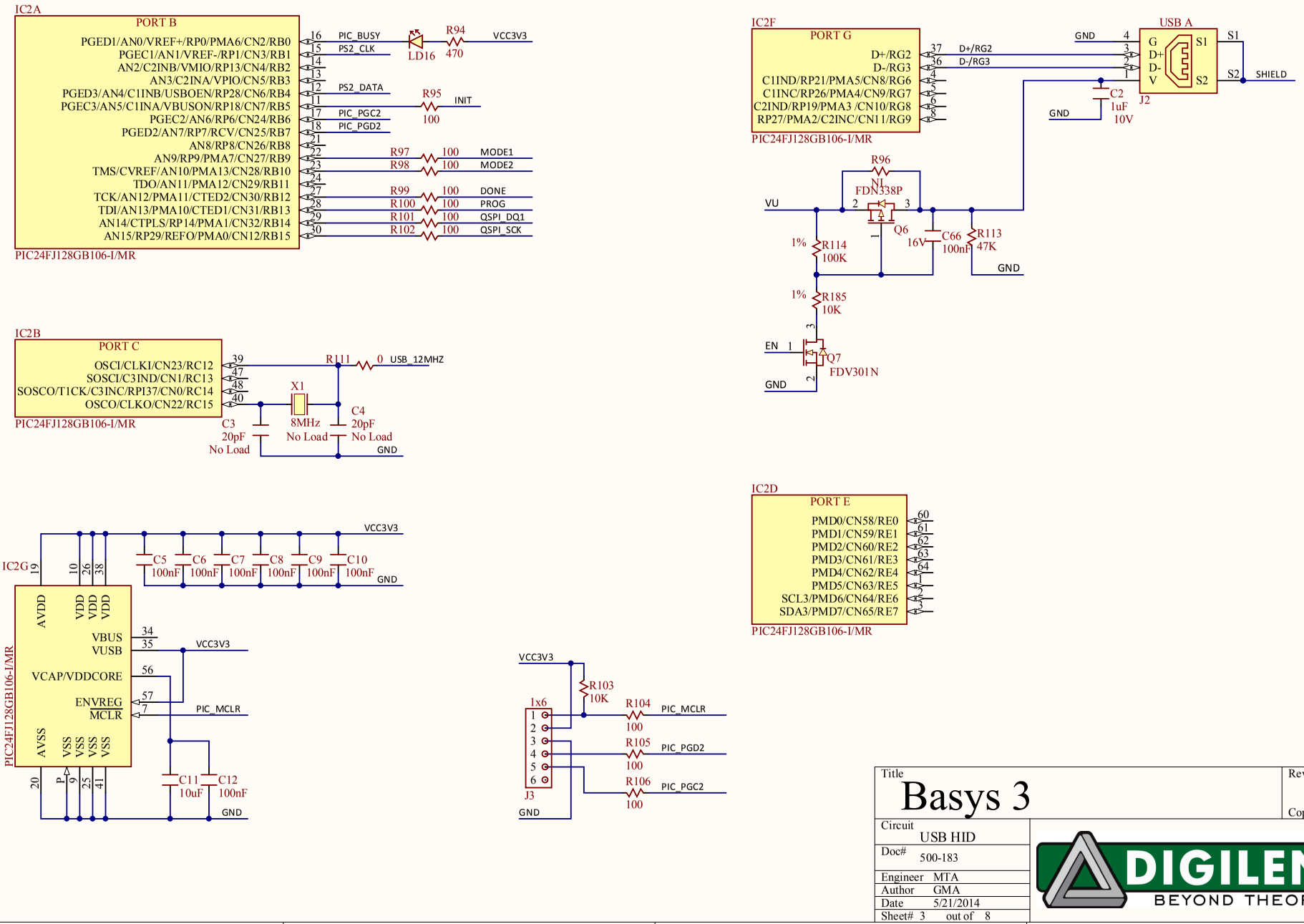


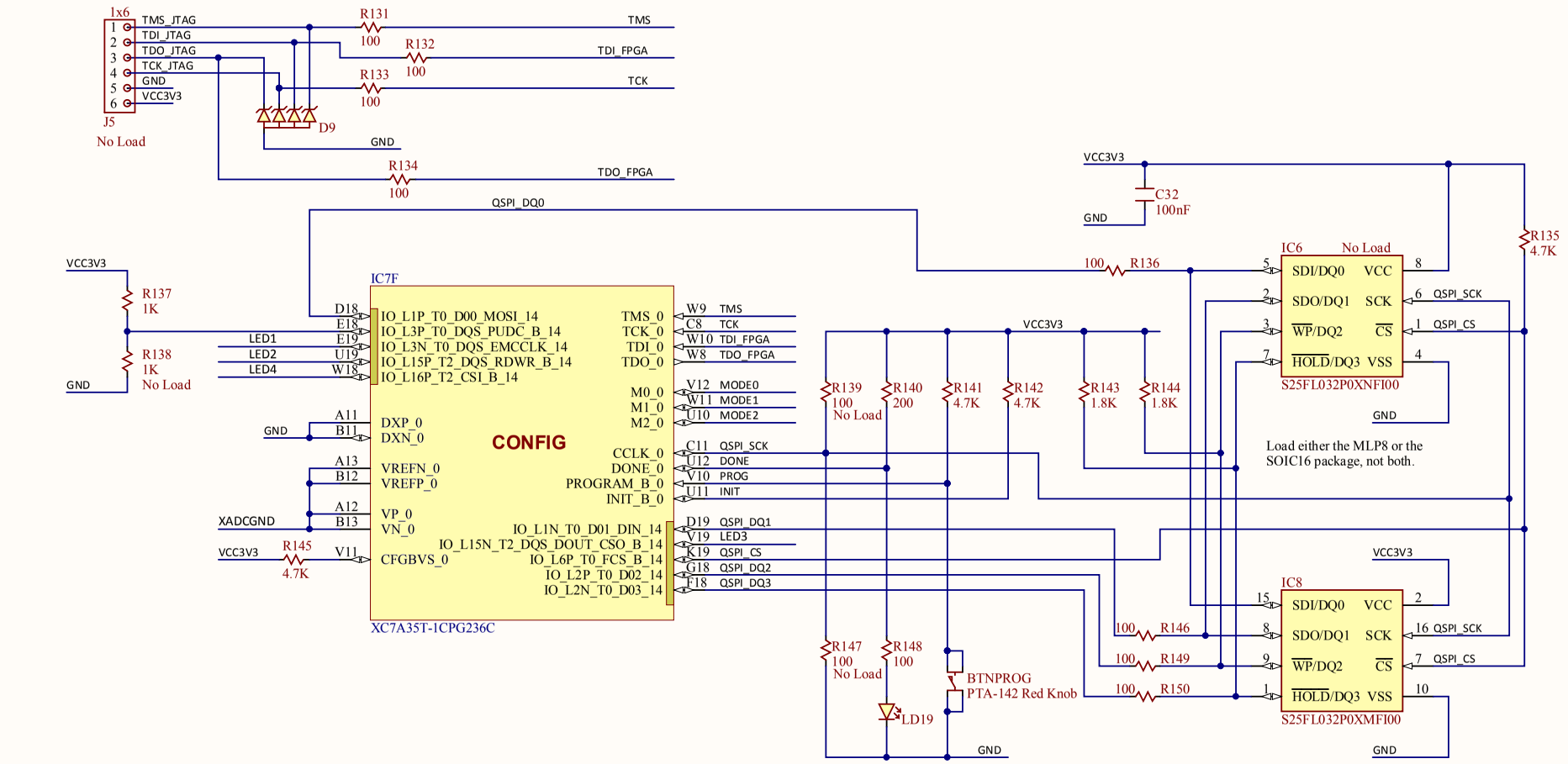




Title	Basys 3
Rev	C.0
Copyright	2014
Circuit	7 SEG, VGA, LEDs
Doc#	500-183
Engineer	MTA
Author	GMA
Date	5/21/2014
Sheet#	2 out of 8

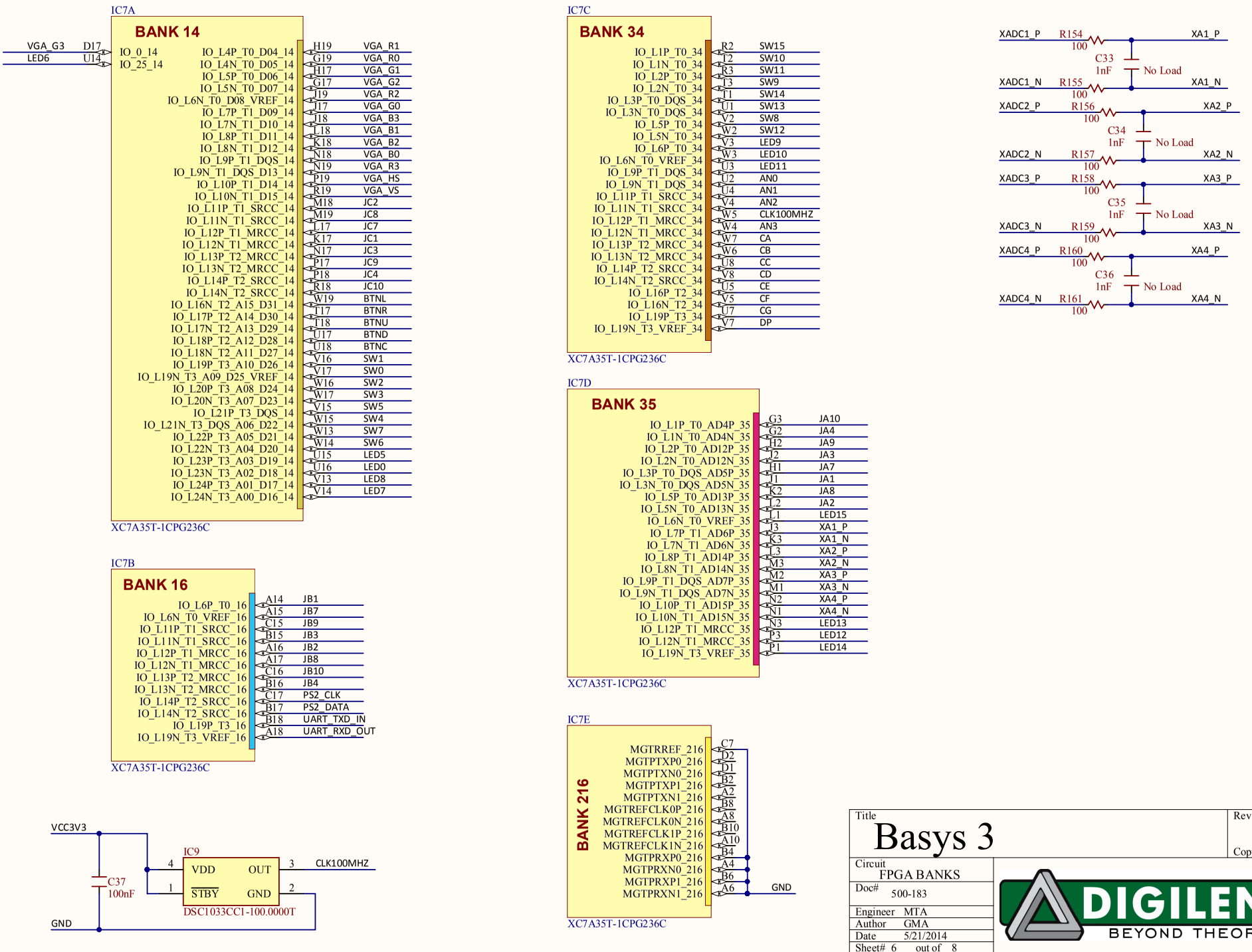
DIGILENT® BEYOND THEORY

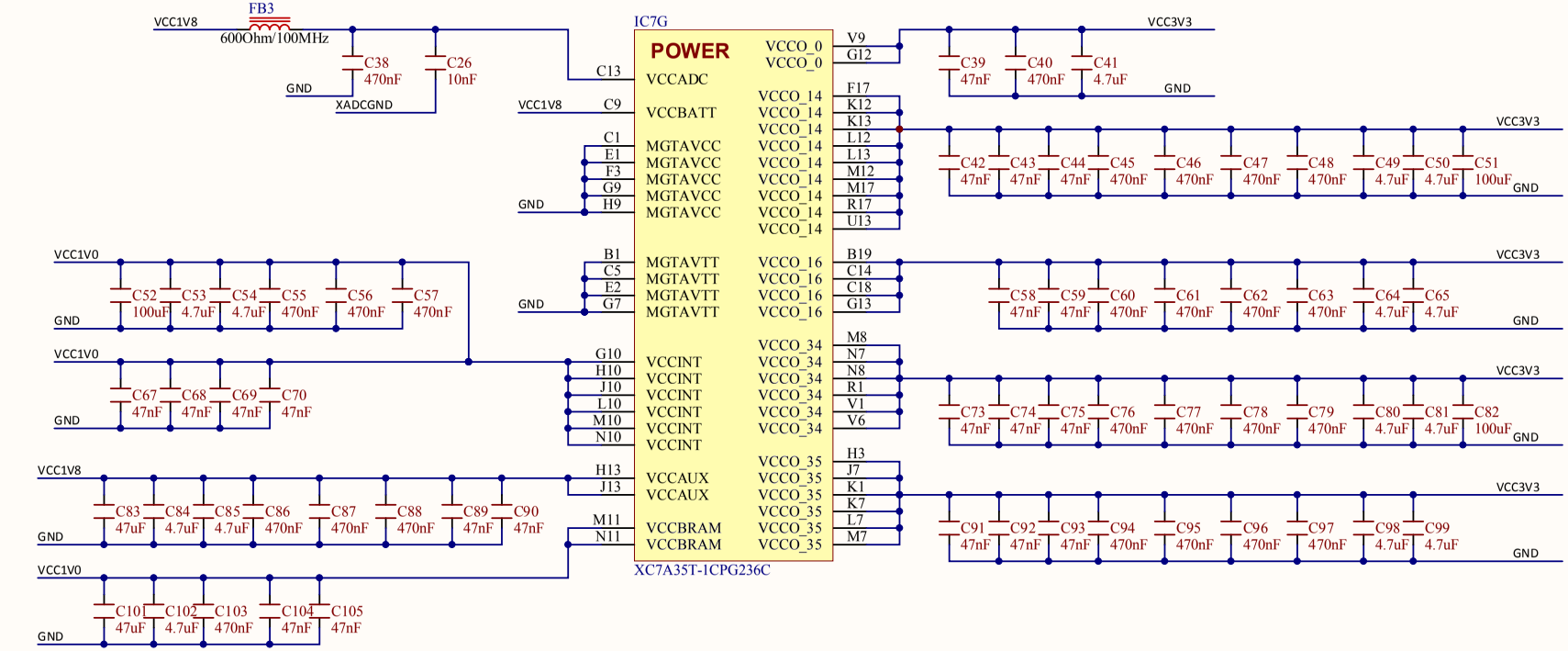

IC2A
PORT B
PGED1/AN0/VREF+/RP0/PMA6/CN2/RB0
PGEC1/AN1/VREF-/RP1/CN3/RB1
AN2/C2INB/VMIO/RP13/CN4/RB2
AN3/C2INA/VPIO/CN5/RB3
PGED3/AN4/C1INB/USBOEN/RP28/CN6/RB4
PGEC3/AN5/C1INA/VBUSON/RP18/CN7/RB5
PGEC2/AN6/RP6/CN24/RB6
PGED2/AN7/RP7/RCV/CN25/RB7
AN8/RP8/CN26/RB8
AN9/RP9/PMA7/CN27/RB9
TMS/CVREF/AN10/PMA13/CN28/RB10
TDO/AN11/PMA12/CN29/RB11
TCK/AN12/PMA11/CTED2/CN30/RB12
TDI/AN13/PMA10/CTED1/CN31/RB13
AN14/CTPLS/RP14/PMA1/CN32/RB14
AN15/RP29/REFO/PMA0/CN12/RB15
PIC_BUSY
PS2_CLK
PS2_DATA
PIC_PGC2
PIC_PGD2
LD16
R94
470
VCC3V3
R95
100
INIT
R97
R98
R99
R100
R101
R102
MODE1
MODE2
DONE
PROG
QSPI_DQ1
QSPI_SCK
PIC24FJ128GB106-I/MR
IC2F
PORT G
D+/RG2
D-/RG3
C1IND/RP21/PMA5/CN8/RG6
C1INC/RP26/PMA4/CN9/RG7
C2IND/RP19/PMA3 /CN10/RG8
RP27/PMA2/C2INC/CN11/RG9
USB A
GND
S1
S2
SHIELD
C2
1uF
10V
J2
R96
N1
FDN338P
VU
Q6
C66
100nF
16V
R113
47K
R114
100K
1%
R185
10K
EN
Q7
FDV301N
IC2B
PORT C
OSCI/CLKI/CN23/RC12
SOSCI/C3IND/CN1/RC13
SOSCO/T1CK/C3INC/RPI37/CN0/RC14
OSCO/CLKO/CN22/RC15
R111
0
USB_12MHZ
X1
8MHz
C3
20pF
No Load
C4
20pF
No Load
IC2D
PORT E
PMD0/CN58/RE0
PMD1/CN59/RE1
PMD2/CN60/RE2
PMD3/CN61/RE3
PMD4/CN62/RE4
PMD5/CN63/RE5
SCL3/PMD6/CN64/RE6
SDA3/PMD7/CN65/RE7
IC2G
PIC24FJ128GB106-I/MR
AVDD
VDD
VDD
VDD
VBUS
VUSB
VCAP/VDDCORE
ENVREG
MCLR
AVSS
VSS
VSS
VSS
VSS
C5
C6
C7
C8
C9
C10
100nF
C11
10uF
C12
100nF
PIC_MCLR
J3
1x6
R103
10K
R104
R105
R106
100
Title
Basys 3
Rev
C.0
Copyright 2014
Circuit
USB HID
Doc#
500-183
Engineer MTA
Author GMA
Date 5/21/2014
Sheet# 3 out of 8
DIGILENT
BEYOND THEORY

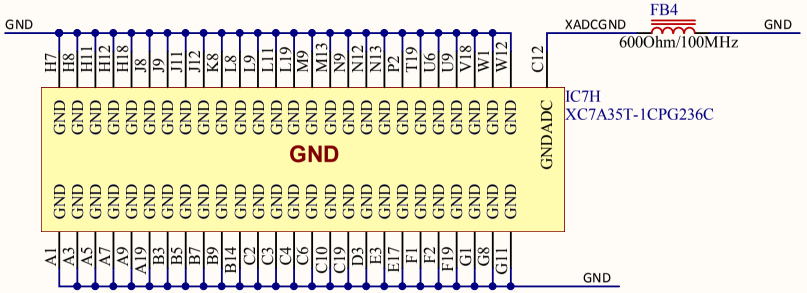


Title	Basys 3
Rev	C.0
Copyright	2014
Circuit	CONFIG, SPI FLASH
Doc#	500-183
Engineer	MTA
Author	GMA
Date	5/21/2014
Sheet#	5 out of 8

DIGILENT
BEYOND THEORY

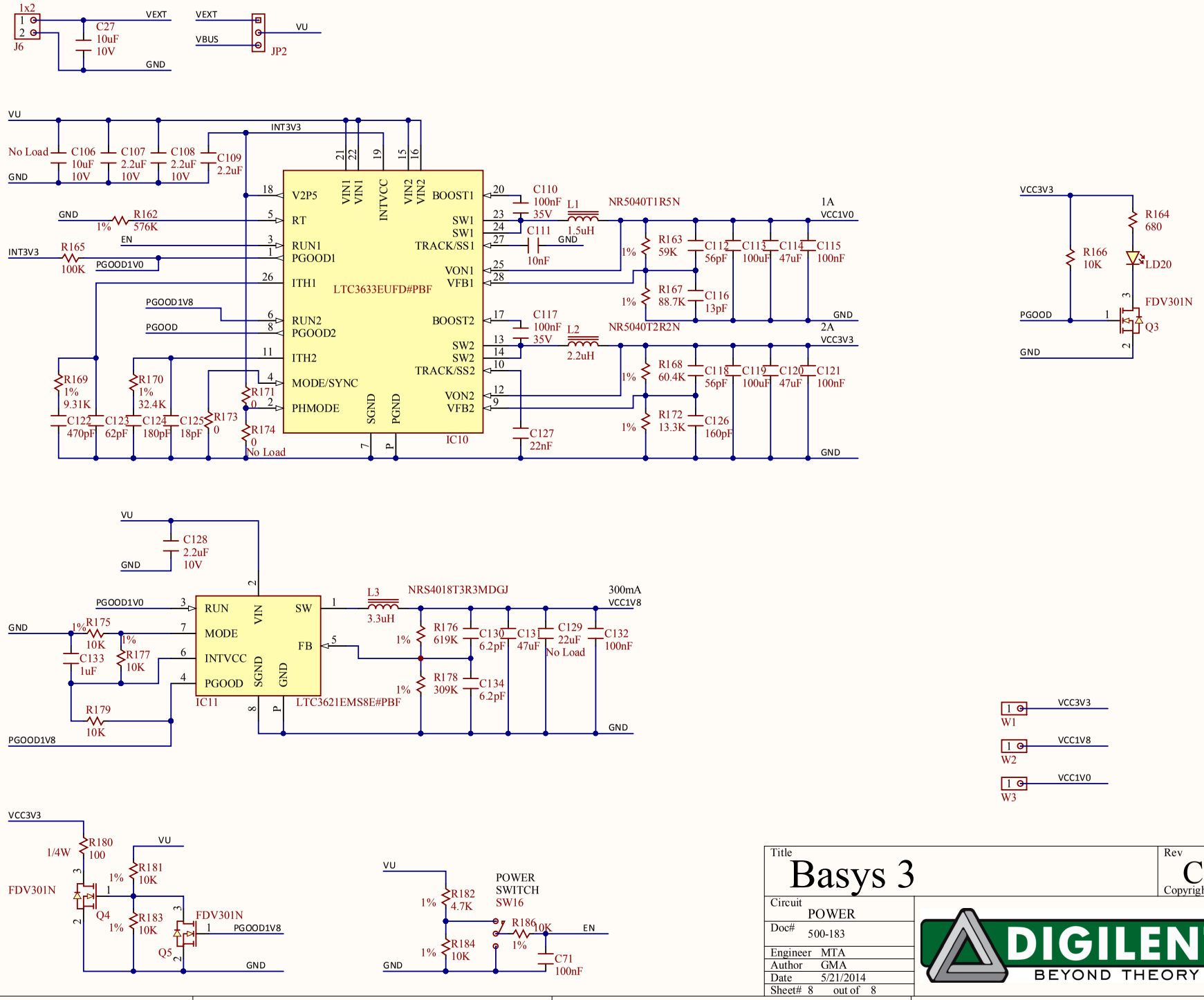

IC7A
BANK 14
VGA_G3 D17
LED6 U14
IO_0_14
IO_25_14
IO_L4P_T0_D04_14 H19 VGA_R1
IO_L4N_T0_D05_14 G19 VGA_R0
IO_L5P_T0_D06_14 H17 VGA_G1
IO_L5N_T0_D07_14 G17 VGA_G2
IO_L6N_T0_D08_VREF_14 J19 VGA_R2
IO_L7P_T1_D09_14 J17 VGA_G0
IO_L7N_T1_D10_14 J18 VGA_B3
IO_L8P_T1_D11_14 K18 VGA_B1
IO_L8N_T1_D12_14 K18 VGA_B2
IO_L9P_T1_DQS_14 N18 VGA_B0
IO_L9N_T1_DQS_D13_14 N19 VGA_R3
IO_L10P_T1_D14_14 P19 VGA_HS
IO_L10N_T1_D15_14 R19 VGA_VS
IO_L11P_T1_SRCC_14 M18 JC2
IO_L11N_T1_SRCC_14 M19 JC8
IO_L12P_T1_MRCC_14 L17 JC7
IO_L12N_T1_MRCC_14 K17 JC1
IO_L13P_T2_MRCC_14 N17 JC3
IO_L13N_T2_MRCC_14 P17 JC9
IO_L14P_T2_SRCC_14 P18 JC4
IO_L14N_T2_SRCC_14 R18 JC10
IO_L16N_T2_A15_D31_14 W19 BTNL
IO_L17P_T2_A14_D30_14 T17 BTNR
IO_L17N_T2_A13_D29_14 T18 BTNU
IO_L18P_T2_A12_D28_14 U17 BTND
IO_L18N_T2_A11_D27_14 U18 BTNC
IO_L19P_T3_A10_D26_14 V16 SW1
IO_L19N_T3_A09_D25_VREF_14 V17 SW0
IO_L20P_T3_A08_D24_14 W16 SW2
IO_L20N_T3_A07_D23_14 W17 SW3
IO_L21P_T3_DQS_14 V15 SW5
IO_L21N_T3_DQS_A06_D22_14 W15 SW4
IO_L22P_T3_A05_D21_14 W13 SW7
IO_L22N_T3_A04_D20_14 W14 SW6
IO_L23P_T3_A03_D19_14 U15 LED5
IO_L23N_T3_A02_D18_14 U16 LED0
IO_L24P_T3_A01_D17_14 V13 LED8
IO_L24N_T3_A00_D16_14 V14 LED7
XC7A35T-1CPG236C
IC7B
BANK 16
IO_L6P_T0_16 A14 JB1
IO_L6N_T0_VREF_16 A15 JB7
IO_L11P_T1_SRCC_16 C15 JB9
IO_L11N_T1_SRCC_16 B15 JB3
IO_L12P_T1_MRCC_16 A16 JB2
IO_L12N_T1_MRCC_16 A17 JB8
IO_L13P_T2_MRCC_16 C16 JB10
IO_L13N_T2_MRCC_16 B16 JB4
IO_L14P_T2_SRCC_16 C17 PS2_CLK
IO_L14N_T2_SRCC_16 B17 PS2_DATA
IO_L19P_T3_16 B18 UART_TXD_IN
IO_L19N_T3_VREF_16 A18 UART_RXD_OUT
XC7A35T-1CPG236C
VCC3V3
C37
100nF
GND
IC9
4 VDD OUT 3 CLK100MHZ
1 STBY GND 2
DSC1033CC1-100.0000T
IC7C
BANK 34
IO_L1P_T0_34 R2 SW15
IO_L1N_T0_34 T2 SW10
IO_L2P_T0_34 R3 SW11
IO_L2N_T0_34 T3 SW9
IO_L3P_T0_DQS_34 T1 SW14
IO_L3N_T0_DQS_34 U1 SW13
IO_L5P_T0_34 V2 SW8
IO_L5N_T0_34 W2 SW12
IO_L6P_T0_34 V3 LED9
IO_L6N_T0_VREF_34 W3 LED10
IO_L9P_T1_DQS_34 U3 LED11
IO_L9N_T1_DQS_34 U2 AN0
IO_L11P_T1_SRCC_34 U4 AN1
IO_L11N_T1_SRCC_34 V4 AN2
IO_L12P_T1_MRCC_34 V5 CLK100MHZ
IO_L12N_T1_MRCC_34 W4 AN3
IO_L13P_T2_MRCC_34 W7 CA
IO_L13N_T2_MRCC_34 W6 CB
IO_L14P_T2_SRCC_34 U8 CC
IO_L14N_T2_SRCC_34 V8 CD
IO_L16P_T2_34 U5 CE
IO_L16N_T2_34 V5 CF
IO_L19P_T3_34 U7 CG
IO_L19N_T3_VREF_34 V7 DP
XC7A35T-1CPG236C
IC7D
BANK 35
IO_L1P_T0_AD4P_35 G3 JA10
IO_L1N_T0_AD4N_35 G2 JA4
IO_L2P_T0_AD12P_35 H2 JA9
IO_L2N_T0_AD12N_35 J2 JA3
IO_L3P_T0_DQS_AD5P_35 H1 JA7
IO_L3N_T0_DQS_AD5N_35 J1 JA1
IO_L5P_T0_AD13P_35 K2 JA8
IO_L5N_T0_AD13N_35 L2 JA2
IO_L6N_T0_VREF_35 L1 LED15
IO_L7P_T1_AD6P_35 J3 XA1_P
IO_L7N_T1_AD6N_35 K3 XA1_N
IO_L8P_T1_AD14P_35 L3 XA2_P
IO_L8N_T1_AD14N_35 M3 XA2_N
IO_L9P_T1_DQS_AD7P_35 M2 XA3_P
IO_L9N_T1_DQS_AD7N_35 M1 XA3_N
IO_L10P_T1_AD15P_35 N2 XA4_P
IO_L10N_T1_AD15N_35 N1 XA4_N
IO_L12P_T1_MRCC_35 N3 LED13
IO_L12N_T1_MRCC_35 P3 LED12
IO_L19N_T3_VREF_35 P1 LED14
XC7A35T-1CPG236C
IC7E
BANK 216
MGTRREF_216 C7
MGTPTXP0_216 D2
MGTPTXN0_216 D1
MGTPTXP1_216 B2
MGTPTXN1_216 A2
MGTREFCLK0P_216 B8
MGTREFCLK0N_216 A8
MGTREFCLK1P_216 B10
MGTREFCLK1N_216 A10
MGTPRXP0_216 B4
MGTPRXN0_216 A4
MGTPRXP1_216 B6
MGTPRXN1_216 A6
GND
XC7A35T-1CPG236C
XADC1_P R154 100 XA1_P
C33 1nF No Load
XADC1_N R155 100 XA1_N
XADC2_P R156 100 XA2_P
C34 1nF No Load
XADC2_N R157 100 XA2_N
XADC3_P R158 100 XA3_P
C35 1nF No Load
XADC3_N R159 100 XA3_N
XADC4_P R160 100 XA4_P
C36 1nF No Load
XADC4_N R161 100 XA4_N
Title
Basys 3
Rev
C.0
Copyright 2014
Circuit
FPGA BANKS
Doc# 500-183
Engineer MTA
Author GMA
Date 5/21/2014
Sheet# 6 out of 8
DIGILENT
BEYOND THEORY





Title	Basys 3
Rev	C.0
Copyright	2014
Circuit	FPGA POWER
Doc#	500-183
Engineer	MTA
Author	GMA
Date	5/21/2014
Sheet#	7 out of 8

DIGILENT
BEYOND THEORY



Title	Basys 3
Rev	C.0
Copyright	2014
Circuit	POWER
Doc#	500-183
Engineer	MTA
Author	GMA
Date	5/21/2014
Sheet#	8 out of 8

DIGILENT
BEYOND THEORY