Titre: Title:	Réalisation d'un système de conversion des couleurs pour un capteur d'images CMOS à photodétecteur sans filtre optique
Auteur: Author:	Mohamed Sebbar
Date:	2011
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Sebbar, M. (2011). Réalisation d'un système de conversion des couleurs pour un capteur d'images CMOS à photodétecteur sans filtre optique [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/674/

UNIVERSITÉ DE MONTRÉAL

RÉALISATION D'UN SYSTÈME DE CONVERSION DES COULEURS POUR UN CAPTEUR D'IMAGES CMOS À PHOTODÉTECTEUR SANS FILTRE OPTIQUE

MOHAMED SEBBAR

DÉPARTEMENT DE GÉNIE ÉLECTRIQUE

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES

(GÉNIE ÉLECTRIQUE)

OCTOBRE 2011

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

RÉALISATION D'UN SYSTÈME DE CONVERSION DES COULEURS POUR UN CAPTEUR D'IMAGES CMOS À PHOTODÉTECTEUR SANS FILTRE OPTIQUE

Présenté par : SEBBAR Mohamed

en vue de l'obtention du diplôme de : Maîtrise ès Sciences Appliquées

a été dûment accepté par le jury d'examen constitué de :

M. KHOUAS Abdelhakim, Doct., président

M. AUDET Yves, Ph.D., membre et directeur de recherche

M. DAVID Jean-Pierre, Ph.D., membre et codirecteur de recherche

M. DIACONU Vasile, Ph.D., membre

DÉDICACE

À mes parents

REMERCIEMENTS

Je voudrai tout d'abord remercier mon directeur de recherche, monsieur Yves Audet, pour m'avoir accepté dans son équipe de recherche. Mes remerciements vont également à lui et à monsieur Jean-Pierre David pour leur suivi exemplaire tout au long de la réalisation du projet de ce mémoire. Leur disponibilité et leur aide constante ont grandement contribué à rendre mon travail sur mon projet de recherche plaisant.

J'adresse aussi mes remerciement à monsieur Abdelhakim Khouas, président du jury d'examen, et monsieur Vasile Diaconu, membre, pour leur acceptation de participer au jury d'examen de ce mémoire.

Je souhaite également adresser mes remerciements aux techniciens Jacques Girardin, Décarie Gaétan et Laurent Mouden du département Génie Électrique pour leur soutien technique tout au long de la réalisation de mon projet de recherche.

RÉSUMÉ

L'équipe de recherche de mon directeur de recherche, monsieur Yves Audet, travail sur un capteur d'images CMOS (Complementary Metal-Oxide-Semiconductor) sans filtre optique novateur. Les capteurs CCD (Charge-Coupled Device) et CMOS actuellement sur le marché sont basés sur des photodétecteurs sensibles à l'intensité lumineuse et non pas à la longueur d'onde. Pour les rendre sensible à la couleur, des filtres chromatiques sont déposés sur chaque photodétecteur afin que chaque photodétecteur ne détecte qu'une longueur d'onde spécifique. Cette façon de faire a deux inconvénients majeurs. Premièrement, l'utilisation du filtre chromatique réduit la sensibilité du capteur puisque ce filtre absorbe une partie de l'intensité lumineuse. Deuxièmement, chaque photodétecteur ne détecte qu'une couleur obligeant le recours à des algorithmes mathématiques d'extrapolation pour déduire les deux couleurs manquantes pour chaque pixel à partir des couleurs des pixels adjacents.

Le nouveau capteur novateur remédie à ces deux inconvénients en ne nécessitant pas le dépôt des filtres optiques chromatiques sur les photodétecteurs. Il détecte les couleurs en utilisant la propriété d'un matériau semi-conducteur selon laquelle la profondeur de pénétration des ondes électromagnétiques varie avec la longueur d'onde. En utilisant cette propriété, le capteur permet de détecter trois couleurs avec chaque pixel. Le nouveau capteur est réalisable entièrement en technologie CMOS contrairement à un capteur de la compagnie Foveon qui a vue le jour en 2002 et qui utilise le même principe d'absorption de la lumière dans un semi-conducteur pour filtrer la lumière selon la longueur d'onde. Ce capteur n'est pas compatible avec la technologie CMOS parce qu'il utilise trois photodiodes enfouies verticalement dans le semi-conducteur. Ceci rend la réalisation de ce capteur coûteuse.

Le présent travail de recherche est une continuité dans les efforts de développement du nouveau capteur d'images. Il consiste à trouver une méthode pour la conversion des couleurs de l'espace de couleurs du capteur en couleurs dans un espace standard. La conversion des couleurs est réalisée à l'aide d'une matrice 3x3 de conversion des couleurs. Une méthode de calcul de cette matrice tout en considérant le bruit dans l'image a été identifiée et implémentée sous forme d'un GUI Matlab. La considération du bruit dans l'image est indispensable vue la nature du capteur pour qui la matrice de conversion peut avoir de grands coefficients dans la diagonale. Or, ceux-ci sont responsables de la dégradation des performances en bruit. Par la suite, un circuit numérique

de conversion des couleurs a été conçu en VHDL. Une méthode graphique d'évaluation de la réponse spectrale d'un capteur d'images selon les régions de l'espace de couleur a été développée. Le deuxième volet de ce travail est de concevoir un prototype de démonstration pour le nouveau capteur. Ce prototype a été réalisé à l'aide d'une carte de développement à base de FPGA d'Altera jumelée à un écran couleur tactile. Le FPGA a été utilisé pour réaliser des circuits de contrôle, d'acquisition et d'affichage contrôlés par un processeur NIOS II. La réalisation du prototype de démonstration a été terminée avec succès.

ABSTRACT

The research team of my supervisor, Mr Yves Audet, work on an innovator CMOS (Complementary Metal-Oxide-Semiconductor) image sensor without optical filter. The CCD (Charge-Coupled Device) and CMOS sensors on the market today are based on photodetectors which are sensitive to the light intensity and not to the wavelength. To make them sensitive to color, color filters are deposited on each photodetector so that each photodetector detects only a specific wavelength. This approach has two major drawbacks. First, the use of the color filter reduces the sensitivity of the sensor as the filter absorbs some of the light intensities. Second, each photodetector detects only one color requiring the use of mathematical extrapolation algorithms to deduce the two missing colors at each pixel from the colors of adjacent pixels.

The new innovative image sensor overcomes these disadvantages by not requiring the deposit of the color optical filters on the photodetectors. It detects color by using the property of a semiconductor material that the penetration depth of electromagnetic waves varies with wavelength. Using this property, the sensor can detect three colors with each pixel. The new sensor is made entirely in CMOS technology as opposed to a Foveon sensor, a company founded in 2002 and uses the same principle of light absorption in a semiconductor to filter the light according to the wavelength. This sensor is not compatible with CMOS technology because it uses three photodiodes buried vertically into the semiconductor. This makes the realization of this sensor expensive.

This research work is a continuity in the development effort of the new sensor. It is to find a method for color conversion from sensor color space to a standard color space. The color conversion is performed using a 3x3 color conversion matrix. A method of calculating this matrix while considering the noise in the image has been identified and implemented as a Matlab GUI. The consideration of noise in the image is essential because of the nature of the sensor for which the conversion matrix can have large coefficients in the diagonal. However, they are responsible for the degradation of noise performance. Subsequently, a digital circuit for color conversion was designed in VHDL. A graphical method for evaluating the spectral response of an image sensor according to the regions of color space was developed. The second part of this work is to develop a demonstration prototype for the new sensor. This prototype was made using a development board based on Altera FPGA coupled with a color touch screen. The FPGA was used to make

circuits for monitoring, acquisition and display controlled by a Nios II processor. The demonstration prototype was successfully completed.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

ADC	Analog Digital Converter
ASIC	Application Specific Integrated Circuit
CCD	Charge Coupled Device
CFA	Color Filter Array
CMOS	Complementary Metal-Oxide-Semiconductor
FPGA	Field Programmable Gate Array
GUI	Graphical User Interface
I2C	Inter Integrated Circuit
LSB	Least Significant Bit
LUT	Look-Up Table
MOS	Metal Oxide Semiconductor
MSB	Most Significant Bit
PPD	Pinned Photodiode
RGB	Red Green Blue
SI	Système International
SMI	Sensitivity Metamerism Index
SOPC	System On Programmable Chip
VHDL	Very high speed integrated Hardware Description Language

LISTE DES ANNEXES

INTRODUCTION

Au cours des deux dernières décennies, les progrès technologique réalisés au niveau matériel et logiciel ont permis un grand essor à la vente de l'électronique grand public. Ils sont basés sur la conversion de l'information analogique en sa forme numérique. Comme dans de nombreux autres domaines où les appareils numériques ont remplacé leurs prédécesseurs analogiques, les fabricants et les consommateurs ont perdu l'intérêt pour les caméras à film conventionnel et se sont tourné plutôt vers les appareils photo numériques. Ceci est principalement dû au fait que la capture et le développement de photos en utilisant les procédés chimiques et mécaniques ne peuvent pas fournir aux utilisateurs le confort des appareils photo numériques qui enregistrent, stockent et manipulent électroniquement les photographies en utilisant des capteurs d'images et des processeurs embarqués.

Des fonctionnalités telles que l'affichage d'une image immédiatement après sa capture, la capacité de stocker des milliers d'images sur un petit périphérique de mémoire, la capacité de supprimer des images de ce périphérique afin de permettre sa réutilisation, et la possibilité d'éditer des images avec son appareil photo numérique font de ces appareils des produits électroniques de consommation très attractifs.

Pour créer une image d'une scène, les caméras numériques utilisent une série de lentilles pour concentrer la lumière sur un capteur d’images qui échantillonne la lumière et l’enregistre sous forme d’information électronique qui est ensuite convertis en données numériques. Le capteur d’images est une matrice de photodétecteurs sensibles à la lumière, appelés pixels, qui enregistrent l'intensité de la lumière qui frappe leur surface. Malheureusement, les photodétecteurs sont des dispositifs avec une large bande spectrale qui ne peuvent pas enregistrer les informations de couleur. Parmi les technologies existantes développées pour surmonter ce problème, la solution à un seul capteur d'image offre des compromis entre les performances, la complexité et le coût [1]. Ainsi, la plupart des appareils photo numériques d'aujourd'hui sont des dispositifs à capteur unique qui capturent des scènes visuelles en couleur en utilisant une matrice de photodétecteurs achromatiques en conjonction avec une matrice de filtres colorés.

Il ne devrait pas être surprenant que le domaine des appareils photo numérique à un capteur d'images unique est considéré comme l'un des domaines de recherche qui se développe le plus rapidement. Un nombre incalculable de produits commerciaux dans des champs d'application diversifiés ont vu le jour grâce à ces recherches.

Les capteurs d'images sont omniprésents dans plusieurs types d'applications : on les trouve, en plus des caméras numériques, dans les caméscopes, les scanners, les fax, les photocopieurs, les lecteurs de codes à barres, les systèmes de surveillance, les instruments d'observation spatiaux, les équipements d'imagerie médicale ainsi que dans les systèmes de vision industriels où ils sont utilisés pour le tri et le contrôle de la production.

Il existe deux principales technologies couramment utilisées de nos jours pour réaliser les capteurs d'images : la technologie CCD (Charge Coupled Devices) et la technologie CMOS (Complementary Metal-Oxyde-Semiconductor). La première technologie à être utilisée pour réaliser des capteurs d'images était la technologie CCD qui était utilisée longtemps exclusivement jusqu'à ce que l'utilisation de la technologie CMOS pour réaliser des capteurs d'images arrive à maturité. La technologie CCD avait longtemps offert de très bonnes performances par rapport à la technologie CMOS. Il s'agit d'une technologie mature qui a atteint des niveaux de rendement et de performances près des limites théoriques et cela fait des années qu'elle n'a pas connu d'améliorations significatives. Cependant, cette technologie a de nombreux points faibles : elle nécessite plusieurs niveaux de tensions élevés, la consommation en courant est élevée et le plus important est que le procédé CCD est un procédé spécial couteux et qui, en plus, ne permet pas d'intégrer au capteur d'autres circuits électroniques. Les capteurs d'images CMOS réalisés à l'aide du procédé CMOS semblable au procédé avec lequel sont réalisés les circuits intégrés numériques et les microprocesseurs présente la solution aux points faibles de la technologie CCD. En effet, les capteurs CMOS ont un coût de fabrication bas et une faible consommation en puissance. De plus, il est possible d'intégrer facilement d'autres circuits à même la puce du capteur.

Cette dernière décennie, la fabrication des capteurs d'images à l'aide de la technologie CMOS a connue des avancées technologiques majeures. Ceci a causé une réorientation de la recherche qui a abandonné progressivement les capteurs CCD au profit des capteurs CMOS.

Malgré les avancées technologiques de tous genres, la sélectivité sur la couleur dans un capteur d'images n'a pas connu de progrès majeurs. Pour que le photodétecteur, l'élément détectant l'intensité lumineuse dans le capteur d'image qui est achromatique par nature, puissent être sélectif à la couleur, des filtres chromatique sont déposés sur ces photodétecteurs. Cette méthode présente plusieurs inconvénients : elle est couteuse, l'utilisation du filtre chromatique réduit la sensibilité du capteur et il faut utiliser des algorithmes mathématiques d'extrapolation pour déduire les deux couleurs manquantes pour chaque pixel à partir des couleurs des pixels adjacents.

Une compagnie appelée Foveon [2] qui a vu le jour en 2002 propose un nouveau capteur d'images utilisant le semi-conducteur comme filtre chromatique. Cependant, son capteur n'est pas réalisable avec un procédé CMOS standard contrairement au capteur innovateur sur lequel travail l'équipe de recherche de mon directeur de recherche qui est entièrement réalisable avec un procédé CMOS standard et à faible coût. L'utilisation du semi-conducteur comme filtre chromatique repose sur la propriété de celui-ci selon laquelle la profondeur de pénétration des photons dans le semiconducteur dépend de sa longueur d'onde. Ensuite, les charges électriques générées à différentes profondeurs sont dirigé par un champ électrique vers des collecteurs.

Ce projet de recherche fait partie des efforts de recherche et de développement concernant ce capteur d'image innovateur. Les travaux précédents des autres membres de l'équipe ont permis de réaliser le capteur d'images, de réaliser les circuits analogiques de lecture et de caractériser le capteur d'images. Dans la chaine de traitement d'image pour produire une image sur un media d'affichage à partir d'une image brute du capteur, la conversion des couleurs est une étape importante. La première partie de ce travail s'intéresse à la problématique de la conversion des couleurs en tenant compte des particularités du nouveau capteur d'images. La réalisation de ce travail nous a amené à l'évaluation de la réponse spectrale du capteur à l'aide d'une nouvelle méthode graphique. La deuxième partie s'intéresse à la conception et la réalisation d'un prototype de démonstration permettant d'acquérir des images et de les afficher sur un écran LCD couleur tactile.

Dans ce travail, nous voulions démontrer le bon fonctionnement du nouveau capteur en réalisant un prototype de démonstration contenant une chaine allégée de traitement d'images et permettant de produire une image finale pour affichage. Ma première contribution est l'émulation de la

caméra numérique de démonstration à l'aide d'une carte de prototypage FPGA permettant l'acquisition et l'affichage d'images élaborées par le nouveau capteur.

Lors de la réalisation d'un nouveau capteur, on a souvent besoin d'évaluer la fidélité de la représentation des couleurs de ce dernier par rapport à l'œil humain. Dans la littérature, il existe plusieurs métriques permettant d'évaluer la réponse spectrale d'un capteur. Cependant, cette évaluation est réalisée à l'aide de chiffres qui, en présence d'un capteur achromatique, ne donnent aucune information sur la déficience du capteur pour la reproduction de certaines gammes de couleurs ou de distinguer l'intervalle de couleur médiocrement représenté par le capteur. Le prototype du nouveau capteur utilisé dans ce travail s'est avéré ne pas capter fidèlement les couleurs. On avait besoin d'évaluer ses réponses spectrales en profondeur. Ma deuxième contribution est une nouvelle méthode graphique que nous avons développée pour l'évaluation des réponses spectrales d'un capteur d'images.

Ce mémoire est divisé en trois chapitres suivis d'une conclusion générale. Le premier chapitre est une introduction sur le phénomène de la couleur et la colorimétrie, la science qui s'intéresse à comment la mesurer et la codifier. Le deuxième chapitre présente les capteurs d'images numériques couleurs, leurs différentes technologies et le principe de détection de la couleur à l'aide du filtre Bayer. Il présente également le principe de la détection des couleurs dans le nouveau capteur d'image sur lequel travail l'équipe de recherche. À la fin du chapitre, les différents bruits dans un capteur d'images CMOS sont présentés. Le troisième chapitre présente les travaux réalisés pour l'implémentation d'un algorithme de conversion des couleurs, une nouvelle méthode graphique pour l'évaluation de la réponse spectrale d'un capteur d'images et la conception et la réalisation d'un prototype de démonstration à base d'une carte de prototypage FPGA et un écran couleur tactile.

CHAPITRE 1 LA COULEUR

1.1 La lumière

La lumière est indispensable pour nous car c'est elle qui nous permet de voir la forme des objets et leurs couleurs. L'étude de ces propriétés et des phénomènes visuels impliqués dans sa perception sont cruciaux pour l'apprivoiser dans toutes sortes d'applications.

1.1.1 Nature de la lumière

L'optique est la partie de la physique qui permet d'étudier les propriétés de la lumière et, en plus, de développer un groupe de techniques qui permettent de l'utiliser.

En optique, la lumière est considérée à la fois une onde électromagnétique et une particule d'énergie. On appel cette dualité la dualité onde-corpuscule. Durant des siècles et avant d'accepter cette dualité, les deux conceptions de la lumière se sont affrontées. La nature ondulatoire de la lumière est montrée par certains phénomènes comme la diffraction et l'interférence. Ces phénomènes sont étudiés dans ce qu'on appel l'optique ondulatoire. De l'autre coté, la nature corpusculaire de la lumière expliquait bien au $17^{ème}$ siècle des phénomènes optiques comme la propagation linéaire, la réflexion sur un objet, la réfraction dans un milieu et la dispersion [3]. La description et l'étude de ces phénomènes dans ce qu'on appelait l'optique géométrique supposait des rayons lumineux rectilignes formés de particules. On a noté également les échanges d'énergie entre la matière et le rayonnement lors de l'émission de la lumière et de son absorption par les atomes qui sont des phénomènes discontinues et qui ne s'expliquent que par une nature corpusculaire de la lumière. L'étude de ces phénomènes avait conduit à l'optique corpusculaire basée sur le modèle du photon.

Le fondement de la conception ondulatoire de la lumière en tant qu'onde électromagnétique a été établit par James Clerk Maxwell au $19^{ème}$ siècle. Celui-ci a révélé l'existence des ondes électromagnétique qui sont des ondes constituées de la superposition d'un champ électrique et d'un champ magnétique oscillants avec une fréquence f et se déplaçants à une vitesse v dans le milieu de propagation. Lorsque ce milieu est le vide, le couple champ électrique/champ magnétique se déplace à la vitesse c_0 = 299 792 458 m/s [4]. Par la suite, il a établit que les

radiations lumineuses sont de nature électromagnétique et qu'elles ne sont qu'une partie infime de ces ondes. C'est la théorie de Maxwell qui permet de décrire mathématiquement la lumière en tant qu'onde électromagnétique. Depuis les travaux d'Albert Einstein et de Max Plank au début du 20ème siècle, on considère que la lumière est une onde électromagnétique discontinue dont les énergies sont concentrées en de petits paquets, quantas d'énergie, qu'on appel les photons. Ceux-ci sont des particules sans masse mais qui possèdent des quantités de mouvement et des quantités d'énergie. Cette dualité est généralisée à toute particule subatomique en mouvement par la théorie de Louis de Broglie. Cette théorie, a été confirmé par les expériences ultérieures de Davisson et Germer. De nos jours, on accepte la dualité ondes-corpuscules de la nature de la lumière. La naissance de l'électrodynamique quantique a permis de concilier définitivement les deux concepts.

Dans le reste de ce chapitre, seule la nature ondulatoire de la lumière est d'intérêt. On ne s'intéressera ni à l'interaction entre la lumière et la matière, ni aux phénomènes optiques de l'optique géométrique. Par conséquent, on parlera de la lumière comme un rayonnement électromagnétique.

La lumière dénomme les rayonnements électromagnétiques visibles capables de produire une sensation visuelle lorsqu'elles atteignent l'œil humain. Cependant, l'intervalle du domaine spectral visible désigné n'a pas de limites précises. Celles-ci dépondent du flux énergétique qui rentre dans l'œil et de la sensibilité de ce dernier. En général, la borne inférieure est prise entre 360 nm et 400 nm et la borne supérieure est prise entre 760 nm et 830 nm [5].

Parfois, la notion de lumière est étendue aux ondes électromagnétiques dans les domaines infrarouge et ultraviolet qui sont dans les extrémités immédiates du domaine visible. Dans ce cas, on spécifie de quel type de lumière on parle en utilisant les termes : « lumière visible » et « lumière invisible ».

En 1666, Isaac Newton a mis en évidence le spectre de la lumière blanche qui définit le spectre de la lumière visible ou spectre visible tout court en passant un rayon de lumière blanche dans un prisme. Le rayon en sorte sous forme de bandes colorées constitués de couleurs pures. Les mêmes bandes constituent l'arc en ciel qui résulte de la diffraction de la lumière du soleil dans les goutes de la pluie. Les couleurs qui apparaissent sont en nombre de quatre et apparaissent dans l'ordre : rouge, jaune, vert, bleu. Ces couleurs sont appelées « couleurs uniques ». La Figure 1-1 ci-

dessous montre la position du spectre de la lumière visible par rapport au spectre électromagnétique complet.

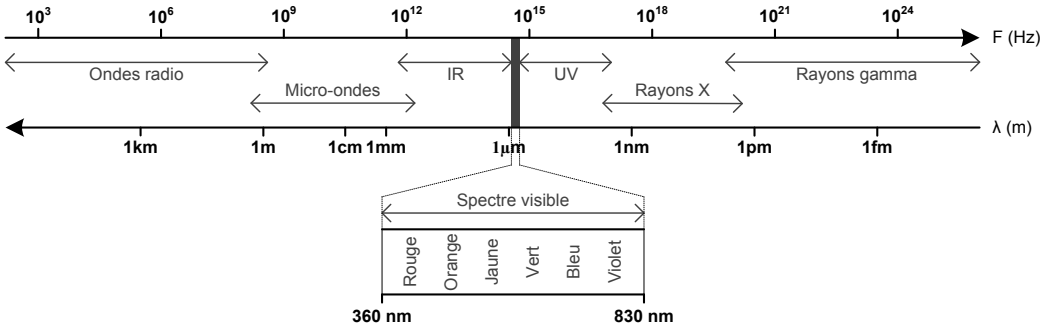


Figure 1-1 : Position de la lumière visible dans le spectre électromagnétique complet.

Jusqu'à présent, on a montré ce que représente la lumière. Dans ce qui suit, il sera question de savoir comment les grandeurs de la lumière sont définies et quelles sont leurs unités SI (Système International).

1.1.2 Radiométrie

Les radiations électromagnétiques comme phénomène physique peuvent être dimensionnées par des mesures physiques qui quantifient les énergies impliquées. Ceci est le but de la radiométrie qui détermine les grandeurs mesurables impliquant l'énergie de la radiation.

La radiométrie évalue les grandeurs physiques d'un rayonnement en termes d'énergie. La Figure 1-2 ci-dessous montre le lien entre les grandeurs d'un rayonnement du point de vue énergétique. On remarque que les grandeurs sont définit pour caractériser les différents aspects du rayonnement énergétique : émission de la source, réception, réémission, …etc.

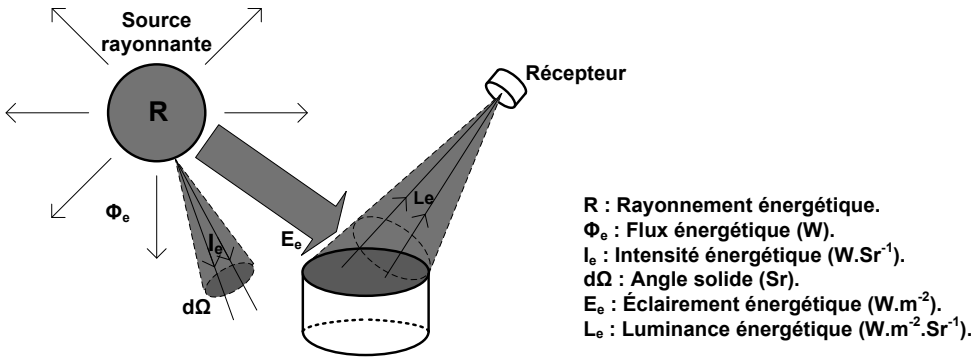


Figure 1-2 : Grandeurs radiométriques principales.

Les grandeurs radiométriques sont définies dans la publication de 1987 conjointe entre la Commission Internationale de l'Éclairage (CIE) et la Commission Électrotechnique Internationale (CEI). La CIE publie le document sous la référence Vocabulaire International de l'Éclairage (ILV) (publication 17.4-1987). La CEI publie ce document sous la référence Vocabulaire Électrotechnique International (VEI), Chapitre 845: Éclairage (publication 50(845)).

Les principales grandeurs radiométriques sont :

a) Flux énergétique :

Le flux énergétique Φ_e, ou puissance rayonnante, est l'énergie « émise par une source rayonnante, transmise par un milieu de propagation, ou reçue sur une surface » [6] par unité de temps. Il s'agit d'une énergie par unité de temps ou puissance.

L'unité du flux énergétique est le Watt (W).

b) Énergie rayonnante :

L'énergie rayonnante Q_e, est l'«intégrale par rapport au temps du flux énergétique Φ_e pendant une durée donnée Δt » [6].

$$Q_e = \int_{\Delta t} \Phi_e dt \qquad (1\text{-}1)$$

L'unité de l'énergie rayonnante est le Joule (J).

c) Intensité énergétique :

L'intensité énergétique I_e d'un rayon lumineux dans une direction donnée est le « quotient du flux énergétique $d\Phi_e$ quittant la source et se propageant dans l'élément d'angle solide $d\Omega$ contenant la direction donnée, par cet élément d'angle solide» [6].

$$I_e = \frac{d\Phi_e}{d\Omega} \qquad (1\text{-}2)$$

L'unité de l'intensité énergétique I_e est le Watt par stéradian (W/sr ou $W.sr^{-1}$).

d) Éclairement énergétique :

L'éclairement énergétique E_e en un point d'une surface est le « quotient du flux énergétique $d\Phi_e$ reçu par élément de surface contenant le point, par l'air dA de cet élément » [6].

$$E_e = \frac{d\Phi_e}{dA} \qquad (1\text{-}3)$$

L'unité de l'éclairement énergétique E_e est le Watt par m^2 (W/m^2 ou $W.m^{-2}$).

e) Exitance énergétique

L'exitance énergétique M_e en un point d'une surface est le « quotient du flux énergétique $d\Phi_e$ quittant un élément de surface contenant le point, par l'air dA de cet élément » [6].

$$M_e = \frac{d\Phi_e}{dA} \qquad (1\text{-}4)$$

L'unité de l'exitance énergétique M_e est le Watt par m^2 (W/m^2 ou $W.m^{-2}$).

f) Luminance énergétique

La luminance énergétique L_e, ou radiance, dans une direction donnée, en un point donné d'une surface réelle ou fictive est une « grandeur définie par la formule

$$L_e = \frac{d\Phi_e}{dA.\cos\theta.d\Omega} \qquad (1\text{-}5)$$

où $d\Phi_e$ est le flux lumineux transmis par un faisceau élémentaire passant par le point donné et se propageant dans l'angle solide $d\Omega$ contenant la direction donnée; dA est l'aire d'une section de ce faisceau au point donné; θ est l'angle entre la normale à cette section et la direction du faisceau » [6].

L'unité de la luminance énergétique L_e est le Watt par m^2 par stéradian ($W/m^2/sr$ ou $W.m^{-2}.sr^{-1}$).

g) Résumé des principales grandeurs radiométriques

Le Tableau 1.1 ci-dessous résume les principales grandeurs radiométriques.

Tableau 1.1 : Résumé des principales grandeurs radiométriques.

GRANDEUR (nom en anglais)	**Unité S.I.**	**SYMBOLE**	**DÉFINITION ILV #**
Flux énergétique Φ_e (Radiant flux)	Watt	W	845-01-24
Énergie rayonnante Q_e (Radiant energy)	Joule	J = W.S	845-01-27
Intensité énergétique I_e (Radiant intensity)	Watt par stéradian	$W.sr^{-1}$	845-01-30
Éclairement énergétique E_e (Irradiance)	Watt par m^2	$W.m^{-2}$	845-01-37
Éxitance énergétique M_e (Radiant exitance)	Watt par m^2	$W.m^{-2}$	845-01-47
Luminance énergétique L_e (Radiance)	Watt par m^2 par stéradian	$W.m^{-2}.sr^{-1}$	845-01-34

Dans cette section, on a introduit la radiométrie et les grandeurs radiométriques. Cependant, la lumière est une observation dont les dimensions sont quantifiées par le système visuel d'un observateur. En fonction de la sensibilité de son système, l'observateur va quantifier les grandeurs différemment. Ceci est le but de la photométrie qui exprime les grandeurs en pondérant les grandeurs énergétiques par la courbe de sensibilité de l'œil. La photométrie et les grandeurs photométriques sont introduites dans la partie 1.1.3 dessous. En tenant compte de la courbe de sensibilité de l'œil, la même quantité d'énergie va donner des grandeurs photométriques différentes selon sa composition spectrale et les quantités d'énergie en dehors du spectre visible vont donner des mesures photométriques nulles.

1.1.3 Photométrie

La photométrie, ou radiométrie optique, s'intéresse à exprimer les grandeurs visuels en fonction des grandeurs énergétiques de la radiation. Les grandeurs visuelles ne dépendent pas uniquement des grandeurs physiques, mais aussi de la réponse visuelle de l'observateur pour qui la composition spectrale du rayonnement va influencer ce que son système visuel va percevoir

comme sensation. En effet, le système visuel a une particularité psychophysiologique se manifestant par une courbe de sensibilité relative spectrale qui a la forme d'une courbe en cloche. Cette courbe montre que le système visuel n'a pas la même sensibilité sur l'ensemble du spectre visible. Cette sensibilité diffère aussi d'un individu à un autre. Ceci a amené à définir un observateur de référence représentatif par la Commission Internationale de l'Éclairage (CIE) qu'on appel observateur photométrique de référence CIE. La CIE définit pour cet observateur une fonction d'efficacité lumineuse spectrale V(λ) pour la vision photopique à des niveaux de luminance d'au moins plusieurs candelas par mètre carré dans lesquelles les cônes sont les principaux récepteurs actifs et une fonction d'efficacité lumineuse spectrale *V'(λ)* pour la vision scotopique à des niveaux de luminance inférieurs à quelques centièmes de candela par mètre carré dans lesquelles les bâtonnets sont les principaux récepteurs actifs [5]. Sur la première courbe, la sensibilité visuelle est maximale à 555 nm se trouvant à la partie des jaunes du spectre proche des verts. Sur la deuxième courbe, la sensibilité visuelle maximale est décalée vers les bleus et se trouve à 507 nm. Les deux courbes s'utilisent avec l'unité de base photométrique du SI, la candela. À ce jour, CIE n'a pas encore définit une fonction d'efficacité lumineuse spectrale pour la vision mésopique, vision intermédiaire entre la vision photopique et la vision scotopique. La Figure 1-3 ci-dessous montre l'efficacité lumineuse spectrale des visions photopique et scotopique tels que définis par la CIE.

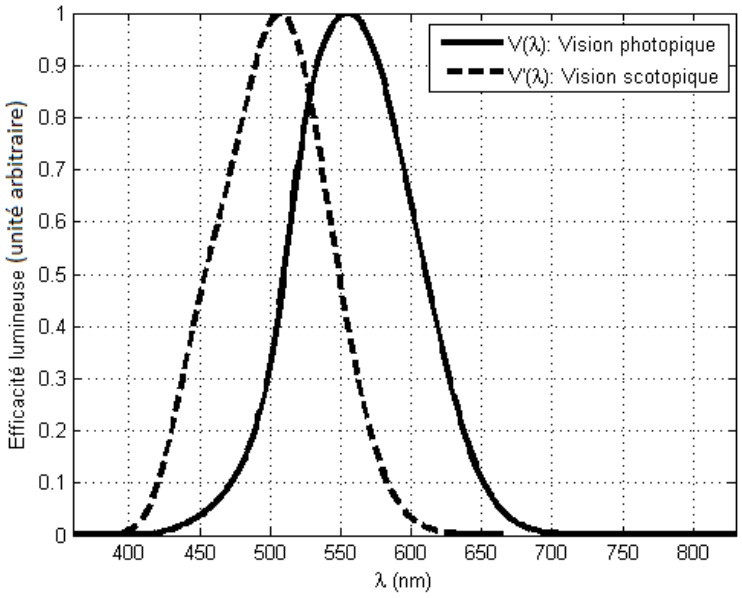


Figure 1-3 : Efficacité lumineuse spectrale de la vision humaine.

Les grandeurs photométriques sont définies dans la publication de 1987 conjointe entre la Commission Internationale de l'Éclairage (CIE) et la Commission Électrotechnique Internationale (CEI). La CIE publie le document sous la référence Vocabulaire International de l'Éclairage (ILV) (publication 17.4-1987). La CEI publie ce document sous la référence Vocabulaire Électrotechnique International (VEI), Chapitre 845: Éclairage (publication 50(845)).

a) Flux lumineux

Le flux lumineux Φ_v, est une « grandeur dérivée du flux énergétique Φe par l'évaluation du rayonnement d'après son action sur l'observateur de référence photométrique CIE » [6].

Pour la vision photopique :

$$\Phi_v = K_m . \int_0^\infty \Phi_{e,\lambda} . V(\lambda) . d\lambda \quad (1\text{-}6)$$

Pour la vision scotopique :

$$\Phi'_v = K'_m . \int_0^\infty \Phi_{e,\lambda} . V'(\lambda) . d\lambda \quad (1\text{-}7)$$

Où $\Phi_{e,\lambda}$ est la répartition spectrale du flux énergétique, $V(\lambda)$ est l'efficacité lumineuse spectrale à la vision photopique, $V'(\lambda)$ est l'efficacité lumineuse spectrale à la vision scotopique, $K_m = 683 \ \ lm.W^{-1}$ et $K'_m = 1700 \ \ lm.W^{-1}$.

L'unité du flux lumineux est le lumen (lm).

b) Énergie lumineuse

L'énergie lumineuse Q_v, ou quantité de lumière, est l'«intégrale par rapport au temps du flux lumineux Φ_v pendant une durée donnée Δt» [6].

$$Q_v = \int_{\Delta t} \Phi_v dt \quad (1\text{-}8)$$

L'unité de l'énergie lumineuse est le lumen seconde (lm.s)

c) Intensité lumineuse

L'intensité lumineuse I_v d'une source lumineuse dans une direction donnée est le « quotient du flux lumineux $d\Phi_v$ quittant la source et se propageant dans l'élément d'angle solide $d\Omega$ contenant la direction donnée, par cet élément d'angle solide» [6].

$$I_v = \frac{d\Phi_v}{d\Omega} \qquad (1\text{-}9)$$

L'unité de l'intensité lumineuse I_v est la candela (cd).

La candela est l'unité photométrique de base du Système International d'unité (SI) pour l'intensité lumineuse. Elle est définit de la manière suivante:

> « La candela est l'intensité lumineuse, dans une direction donnée, d'une source qui émet un rayonnement monochromatique de fréquence 540 x 1012 hertz et dont l'intensité énergétique dans cette direction est de 1/683 Watt par stéradian » [6].

La candela est indépendante du milieu de propagation parce qu'elle utilise la fréquence au lieu de la longueur d'onde dans sa définition.

$1\ cd = 1\ lm.sr^{-1}$

d) Éclairement lumineux

L'éclairement lumineux E_v, ou éclairement tout court, en un point d'une surface est le « quotient du flux énergétique $d\Phi_v$ reçu par élément de surface contenant le point, par l'air dA de cet élément » [6].

$$E_v = \frac{d\Phi_v}{dA} \qquad (1\text{-}10)$$

L'unité de l'éclairement lumineux E_v est le lux (lx).

e) Exitance lumineuse

L'exitance lumineuse M_v en un point d'une surface est le « quotient du flux énergétique $d\Phi_v$ quittant un élément de surface contenant le point, par l'air dA de cet élément » [6].

$$M_v = \frac{d\Phi_v}{dA} \qquad (1\text{-}11)$$

L'unité de l'exitance lumineuse M_v est le lux (lx).

f) Luminance lumineuse

La luminance lumineuse L_v, également appelée luminance visuelle ou luminance tout court, dans une direction donnée, en un point donné d'une surface réelle ou fictive est une « grandeur définie par la formule

$$L_v = \frac{d\Phi_v}{dA.\cos\theta.d\Omega} \qquad (1\text{-}12)$$

où $d\Phi_v$ est le flux lumineux transmis par un faisceau élémentaire passant par le point donné et se propageant dans l'angle solide $d\Omega$ contenant la direction donnée; dA est l'aire d'une section de ce faisceau au point donné; θ est l'angle entre la normale à cette section et la direction du faisceau »[6].

L'unité de la luminance lumineuse L_v est la candela par m^2 (cd/m^2 ou $cd.m^{-2}$).

g) Différence entre luminance et luminosité

La luminance et la luminosité mesurent la même chose, mais dans deux échelles différentes. La luminance permet de décrire si une surface émet plus ou moins de lumière dans une échelle linéaire. La luminosité quand à elle, décrit si cette surface apparait émettre plus ou moins de lumière tel que perçu par la vision humaine. Cette dernière a une échelle qui n'est pas linéaire pour interpréter les variations de la luminance. Ceci est dû à la capacité d'adaptation de l'œil qui rend ce dernier capable de voir aussi bien en forte lumière qu'on lumière faible grâce à l'iris qui compense les variations de l'intensité lumineuse, quelle soit en diminution ou en augmentation, de façon à ce que l'œil ne prend en considération qu'une portion de cette variation.

h) Résumé des principales grandeurs photométriques

Le Tableau 1.2 ci-dessous donne un résumé des principales grandeurs photométriques.

Tableau 1.2 : Résumé des principales grandeurs photométriques.

GRANDEUR (nom en anglais)	**Unité S.I.**	**SYMBOLE**	**DÉFINITION ILV #**
Flux lumineux Φ_v (Luminous flux)	lumen	lm	845-01-25
Énergie lumineuse Q_v (Quantity of light)	lumen seconde	lm.s	845-01-28
Intensité lumineuse I_v (Luminous intensity)	candela	cd = $lm.sr^{-1}$	845-01-31
Éclairement lumineux E_v (Illuminance)	lux	lx = $lm.m^{-2}$	845-01-38
Éxitance lumineuse M_v (Luminous exitance)	lux	lx = $lm.m^{-2}$	845-01-48
Luminance lumineuse L_v (Luminance)	candela par m^2	$cd.m^{-2}$	845-01-35

1.1.4 Les illuminants standard de la CIE

a) Répartition spectrale d'une lumière

Chaque lumière se distingue par une composition spectrale propre à elle. Ainsi, la composition spectrale de la lumière du soleil diffère de celle de la lumière de la lune, qui diffère de celle de la lumière d'une lampe à incandescence …etc. La répartition spectrale, aussi appelée densité spectrale, en énergie d'un rayonnement est représentée par un diagramme sur lequel on note sur l'axe des ordonnées la densité spectrale de chaque radiation qui compose le rayonnement et on représente les différentes radiations sur l'axe des abscisses par leurs longueurs d'onde. Souvent, la courbe de répartition spectrale énergétique est normalisée par rapport à la densité spectrale d'une longueur d'onde de référence choisie arbitrairement. Cette normalisation permet de caractériser une lumière par une répartition spectrale énergétique relative. La Figure 1-4 ci-dessous montre une comparaison de la répartition spectrale énergétique relative de la lumière du soleil à midi, de la lumière d'une lampe fluorescente et de la lumière d'une lampe incandescente. Le soleil et la lampe à incandescence se caractérisent par un spectre d'émission uniforme et

continu. Par contre, la répartition spectrale énergétique de la lumière d'une lampe fluorescente montre l'existence de plusieurs piques de densité spectrale superposés à une courbe de répartition spectrale continue. Ces piques ont pour origine la décharge de vapeurs de mercure.

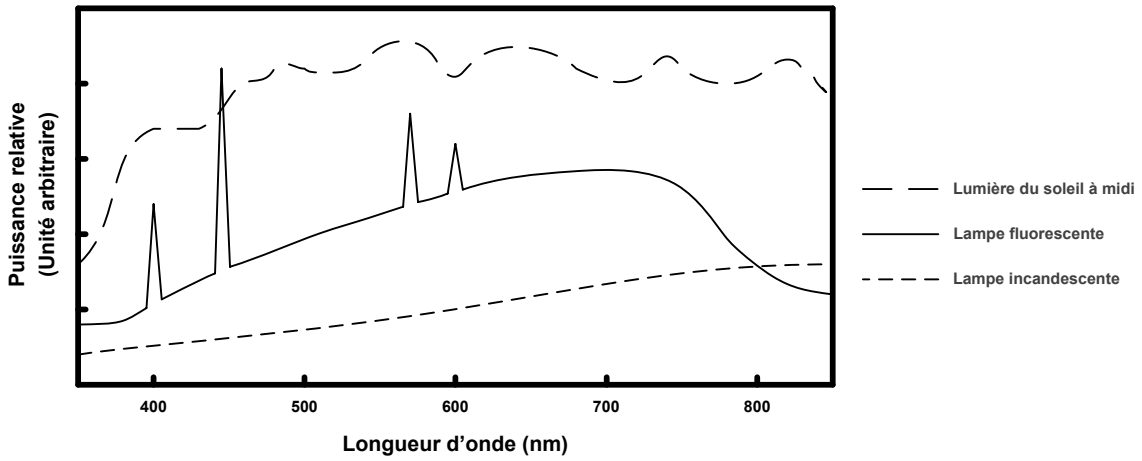


Figure 1-4 : Répartition spectrale énergétique relative de trois types de sources lumineuses.

b) Température de couleur d'une lumière

Au fil des heures d'une journée, la lumière du jour change et on remarque également que la dominante des couleurs change du bleuté au milieu de la journée au jaune orange au crépuscule. Pour caractériser la dominante colorée due à la lumière on décrit celle-ci par sa température de couleur. Le premier à avoir l'idée de comparer les variations de couleur de la lumière du jour avec celles d'un corps noir non coloré théorique chauffé à haute température entre 2000 et 10000 K et dont la couleur passe du rouge, au jaune, au blanc et enfin au bleu est Lord Kelvin [7]. La température de couleur permet de déterminer la température, qu'elle soit effective ou virtuelle, d'une source de lumière à partir de sa couleur. Elle est donc un indice de neutralité du blanc [7]. Son unité de mesure est les Kelvins (K). Une température de 0 K est égale à -273,15 ºC.

La lumière émise par un ciel bleu a une température de couleur de 10000 K. Celle d'un ciel entièrement nuageux est égale à 6500 K. Alors qu'une ampoule incandescente donne une lumière à 2800 K. Le Tableau 1.3 dessous résume les températures de lumières courantes.

Tableau 1.3 : Températures de lumières courantes.

Source de lumière	**Température de couleur (K)**
Bougie	1800
Lampe à incandescence	2500
Lampe halogène	3400
Télévision	9300

c) Illuminants CIE standards

Dans la vie courante, ils existent de multitude de sources de lumière qu'elles soient naturelles comme la lumière du soleil ou artificielles. Pour les différencier et les classer dans des classes de sources standards, la CIE a normalisé plusieurs illuminants et familles d'illuminants dont elle définit la répartition spectrale. Les illuminants sont toujours normalisés relativement à une longueur donnée qui est souvent $\lambda = 560nm$ pour un grand nombre d'illuminants [8]. Aucune unité n'est utilisée pour définir la répartition spectrale de ces illuminants.

Parmi les illuminants standardisés par la CIE, on trouve les illuminants de type lumière du jour pour lesquels elle a définit la famille des illuminants D. Ces illuminants représentent les différentes phases de la lumière du jour. Dans cette famille, on trouve l'illuminant D65 correspondant à la moyenne des lumières durant la journée et dont la température de couleur et de 6504 K. On trouve aussi l'illuminant D50 correspondant à la lumière solaire directe ayant une température de couleur de 5003K. Le Tableau 1.4 ci-dessous donne la température de couleur de plusieurs illuminants CIE.

Tableau 1.4 : Illuminants CIE standards et leurs températures de couleur.

Nom	**Description**	**Température de couleur (K)**
A	Lampe tungstène et incandescente	2856
B	(obsolète) lumière solaire directe à midi	4874
C	(obsolète) lumière moyenne du jour ou lumière d'un ciel nordique	6774
D50	Lumière solaire directe (au lever ou au couché du soleil)	5003
D55	Lumière du jour mi-matinée ou mi-après-midi	5503
D65	Lumière de midi	6504
D75	Lumière d'un ciel nordique	7500
E	Point blanc théorique des systèmes trichromatiques	5600
F	Série de F1 à F12 représentant les sources fluorescentes	

La Figure 1-5 ci-dessous montre les répartitions spectrales des illuminants A et D65.

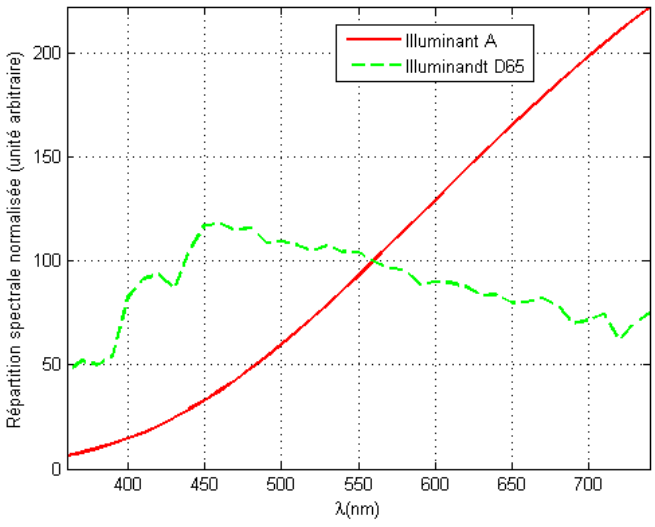


Figure 1-5 : Répartitions spectrales des illuminants A et D65.

1.2 La colorimétrie

1.2.1 Introduction

Lorsqu'on essaie de transmettre de l'information sur une couleur, on se rend compte rapidement que le vocabulaire du langage courant est relativement restreint et qu'il nous donne à peine quelques centaines de mots pour décrire l'infinité de couleurs qu'on trouve dans la nature. En plus, il est peu précis lorsqu'il s'agit de reproduire une couleur à partir d'une description. Cette situation pose un grave problème pour l'utilisation industrielle de la couleur. Trouver une façon pour coder la couleur s'impose rapidement. Cependant, la vision des couleurs étant un processus tri-variant faisant intervenir à la fois des principes physiques, des variables physiologiques et des inconnus psychologiques, cela pose des problèmes vis-à-vis de coder une couleur avec la même valeur quelque soit l'observateur. La solution est apportée par la colorimétrie, la science qui, établie autour de la théorie trichromatique et du concept d'un observateur de référence, s'intéresse à comment mesurer et codifier la couleur. La méthodologie choisie par la Commission Internationale de l'éclairage, CIE, pour la colorimétrie s'appuie sur les principes physiques. Pour se débarrasser des variables et inconnus reliés à l'observateur, elle a définit un « observateur colorimétrique de référence ». Ceci lui a permit d'établir des méthodes objectives pour évaluer la couleur en associant à chaque stimulus visuel élémentaire coloré une mesure. Le stimulus visuel élémentaire correspond à une zone limitée du champ visuel sans texture apparente placée sous un éclairage uniforme et sur un fond de couleur neutre.

La colorimétrie se base sur la recherche de différences entre les stimuli de couleur. On parle de colorimétrie visuelle lorsque l'œil est utilisé pour faire les comparaisons entre ces stimuli. La colorimétrie physique, quant à elle, utilise des récepteurs physiques pour faire les mesures sur ces stimuli.

1.2.2 Théorie trichromatique

Depuis longtemps, les peintres, les teinturiers et les coloristes avaient remarqué que trois couleurs permettaient de reproduire par mélange toutes les couleurs. Cependant, ce n'est qu'après les travaux de Young et de Maxwell que la théorie trichromatique, ou la trichromie, a vue le jour. Dans ces travaux en 1801, Young avait supposé l'existence de trois types de cônes rouge, vert et bleu au niveau de la rétine. Cette hypothèse de trivariance visuelle a été vérifiée ultérieurement

par microspectrophotométrie et par des méthodes électrophysiologiques. Quand à Maxwell, il avait démontré dans ses études expérimentales, publiées à partir de 1885, qu'il pouvait obtenir n'importe quelle couleur en mélangeant trois couleurs rouge, vert et violet. Il est important de noter que la trivariance de la couleur n'est pas une propriété physique des radiations électromagnétique. Elle rend seulement compte d'une propriété physiologique de l'œil.

Selon la théorie trichromatique, trois couleurs fondamentales suffisent pour obtenir, par mélange, toutes les autres couleurs. Elle se base sur un principe expérimental d'égalisation de lumières colorées. En effet, elle statue que :

- Un stimulus de couleur peut être égalisé par l'addition, en proportions convenables, de trois couleurs primaires fixes bien choisies. Il s'agit de la synthèse additive.

- Si les couleurs primaires ne permettent pas l'égalisation, celle-ci est possible en combinant le stimulus couleur avec l'une d'elles.

Les expériences d'égalisation des couleurs consistent à présenter à un observateur normal, un champ photométrique montrant le mélange additif de trois lumières de couleurs primaires à côté du stimulus couleur à égaliser formant le stimulus de référence. Dans ces expériences, la proportion des trois lumières colorées du mélange est modifiée progressivement jusqu'à ce que l'observateur ne perçoive plus de différences entre les deux stimuli. La Figure 1-6 montre le principe des expériences d'égalisation de lumières colorées.

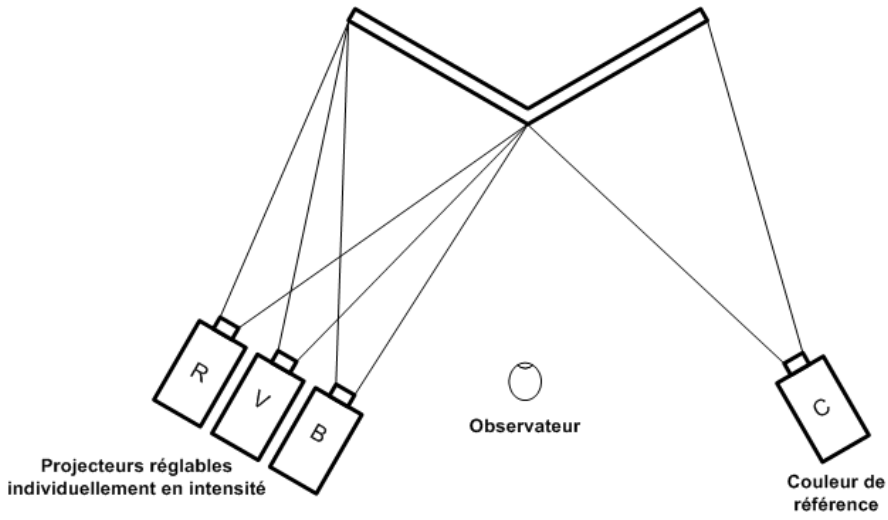


Figure 1-6 : Égalisation d'une couleur de référence par trois primaires.

Les expériences d'égalisation sont basées sur cinq hypothèses sur les propriétés des stimuli de couleur [8]. Si on considère quatre stimuli de couleur A, B, C et D, ceux-ci vérifieront les propriétés suivantes :

- Réflectivité : A Ξ A.

- Symétrie : Si A Ξ B alors B Ξ A.

- Transitivité : Si A Ξ B et B Ξ C alors A Ξ C.

- Additivité : Si A Ξ B alors A+C Ξ B+C.

- Dilatation : Si A Ξ B alors k.A Ξ k.B avec k>0.

Dans l'énoncé de ces propriétés, le symbole « Ξ » est équivaut à « indifférentiable de ».

À partir des quatre premières propriétés, on peut déduire les deux propriétés additionnelles suivantes :

- Additivité (2) : Si A Ξ B et C Ξ D alors A+C Ξ B+D.

- Simplification : Si A+C Ξ B+C alors A Ξ B.

Ces hypothèses sur les propriétés des stimuli de couleurs ont conduits Grassman à faire des études expérimentales qui lui ont permis d'annoncer les trois lois expérimentales suivantes appelées lois de Grassman :

- Trois paramètres indépendants sont nécessaires et suffisants pour égaliser un stimulus coloré.

- Les couleurs perçues et non les répartitions spectrales sont significatives dans un mélange additif de lumières colorées.

- La couleur résultante d'une modification graduelle d'une ou de plusieurs lumières d'un mélange additif de lumières colorées sera également graduellement modifiée.

Les lois de Grassman se vérifient dans le domaine photopique. Par contre, à la limite du mésopique et prés de l'éblouissement, les propriétés d'additivité et de dilatation ne sont plus valides [8]. La propriété de transitivité, quand à elle, n'est plus vérifiée passé le seuil différentiel [8].

1.2.3 Synthèse additive

Le mélange additif de lumières colorées primaires décrit dans les expériences d'égalisation des couleurs correspond à une synthèse additive ou les lumières colorées sont superposées au même endroit d'un écran diffusant. La synthèse additive peut être également réalisée par juxtaposition spatiale de ces lumières, comme c'est le cas dans les moniteurs où les triades de pixels émettant les lumières colorées sont suffisamment petites et rapprochées pour ne pas être perçues séparées. Elle peut être aussi réalisée par juxtaposition temporelle comme dans le cas d'un disque tournant à une vitesse suffisante comportant plusieurs secteurs colorés.

Les expériences d'égalisation avaient démontré qu'il est possible d'égaliser un stimulus coloré S avec un minimum de trois sources lumineuses monochromatiques A, B et C dans des proportions a, b et c respectivement. Ceci donne :

$$S = a.A + b.B + c.C \qquad (1\text{-}13)$$

Cependant, si les sources A, B et C sont des sources monochromatiques fixes, l'égalisation de certaines couleurs ne sera pas possible sans ajouter la lumière de l'une des trois sources A, B ou C au stimulus S à égaliser. Pour ces couleurs, un ou plusieurs coefficients de pondération a, b et c de la relation (13) ci-dessus est négatif.

En théorie, le choix des couleurs primaires n'est pas unique. Par contre, l'expérience a démontré que, pour égaliser une grande gamme de couleurs sans introduire de coefficients négatifs, il est pertinent de choisir les couleurs monochromatiques aux extrémités et au centre du spectre visible. Ce choix répond également à une condition sur les trois couleurs primaires qui spécifie qu'aucune d'elles ne doit résulter du mélange des deux autres. C'est ainsi que le rouge, le vert et bleu sont devenus les couleurs primaires pour la synthèse additive. La Figure 1-7 montre le résultat du mélange additif de ces trois couleurs primaires en proportions identiques.

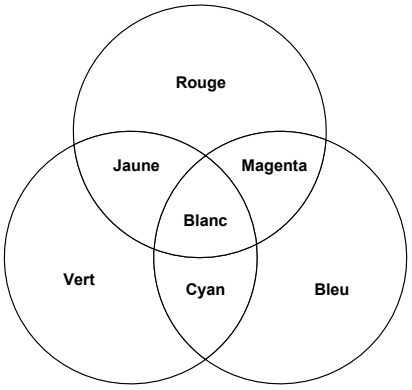


Figure 1-7 : Mélange des couleurs primaires d'une synthèse additive.

On constate que :

- Le mélange du rouge et du vert donne le jaune : $R + V = J$.

- Le mélange du vert et du bleu donne le cyan : $V + B = C$.

- Le mélange du bleu et du rouge donne le magenta : $B + R = M$.

- Le mélange des trois couleurs primaires ensemble donne le blanc : $R + V + B = W$.

- L'absence des trois couleurs primaires donne le noir.

Le jaune, le cyan et le magenta sont des couleurs complémentaires respectivement au bleu, rouge et vert. Le mélange additif de deux couleurs complémentaires produit le blanc. Ainsi :

$$J + B = C + R = M + V = W .$$

Dans la synthèse additive, la luminosité d'une couleur résultante du mélange de deux couleurs primaires, ou des trois couleurs ensembles, est respectivement deux fois, ou trois fois plus lumineuse.

1.2.4 Expression colorimétrique de la couleur

a) Espace colorimétrique CIE-RGB

- Fonctions colorimétriques RGB

Les études d'égalisation des couleurs monochromatiques menées dans les années 1920 par John Guild, W. David Wright et autres chercheurs avaient montrées que des échantillons de couleur peuvent être égalisés par la combinaison de trois couleurs primaires monochromatiques qui sont

le bleu, le vert et le rouge. Les résultats pris en compte par la CIE pour définir les fonctions colorimétriques $\bar{r}(\lambda)$, $\bar{g}(\lambda)$ et $\bar{b}(\lambda)$ de l'observateur standard 2º sont une moyenne des travaux de Guild et Wright. La Figure 1-8 ci-dessous montre le graphe de ces fonctions.

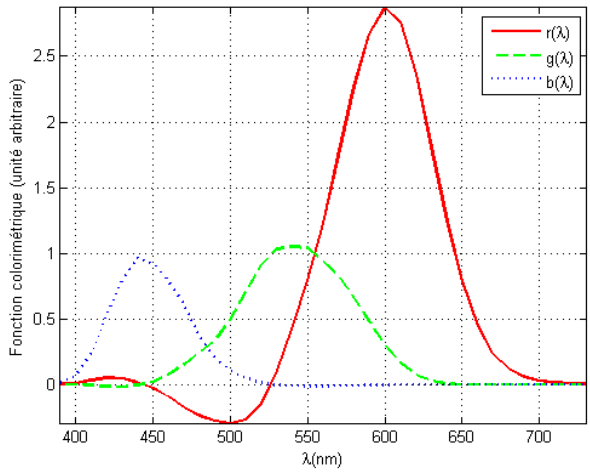


Figure 1-8 : Fonctions colorimétriques CIE-RGB.

On constate qu'égaliser les couleurs monochromatiques dont les longueurs d'ondes se situent entre 445 nm et 526 nm nécessite la soustraction du rouge. Ceci correspond à ajouter du rouge à la couleur à égaliser. Les fonctions colorimétriques XYZ ont été définit par CIE pour résoudre le problème des valeurs négatives.

- Coordonnées trichromatiques RGB

On associant aux trois primaires R, G et B trois vecteurs directeurs normés $\vec{R}$, $\vec{G}$ et $\vec{B}$, on forme un espace vectoriel à trois dimensions d'origine O, montré à la Figure 1-9. Cet espace est appelé l'espace RGB.

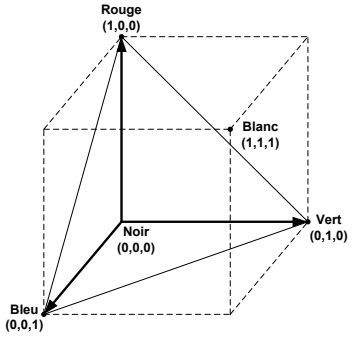


Figure 1-9 : Domaine des couleurs des primaires RGB.

Une couleur C peut être décrite par synthèse additive par un vecteur $\vec{OC}$ de coordonnées R, G et B appelées coordonnées trichromatique tel que :

$$\vec{OC} = R.\vec{R} + G.\vec{G} + B.\vec{B} \quad (1\text{-}14)$$

Les trois coordonnées trichromatiques se calculent comme suit :

$$R = \int_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{r}(\lambda).d\lambda \quad (1\text{-}15)$$

$$G = \int_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{g}(\lambda).d\lambda \quad (1\text{-}16)$$

$$B = \int_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{b}(\lambda).d\lambda \quad (1\text{-}17)$$

où $C(\lambda)$ est la répartition spectrale de la couleur C.

Sur une base discrète, l'intégration sur l'intervalle $[\lambda_{min}, \lambda_{max}]$ devient une sommation discrète sur le même intervalle comme le montre les nouvelles relations suivantes :

$$R = \sum_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{r}(\lambda).d\lambda \quad (1\text{-}18)$$

$$G = \sum_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{g}(\lambda).d\lambda \quad (1\text{-}19)$$

$$B = \sum_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{b}(\lambda).d\lambda \quad (1\text{-}20)$$

- Diagramme de chromaticité CIE-rg

Un problème existe lorsqu'on cherche à analyser les couleurs à partir de leurs coordonnées trichromatiques : deux stimuli de même chrominance et de luminance différentes auront des coordonnées trichromatiques différentes. Afin de faire abstraction de la luminance, il suffit de

normaliser les composantes trichromatiques en divisant chaque coordonnée par la somme des trois. Ainsi, les coordonnées normalisées r, g et b se calculent de la façon suivante :

$$r = \frac{R}{R+G+B} \qquad (1\text{-}21)$$

$$g = \frac{G}{R+G+B} \qquad (1\text{-}22)$$

$$b = \frac{B}{R+G+B} \qquad (1\text{-}23)$$

Dans l'espace RGB, la chromaticité d'un stimulus c.à.d. sa teinte et sa saturation sans tenir compte de la variation de la luminance, correspond à la projection de ces coordonnées trichromatiques R, G et B sur le triangle délimité par les valeurs r, g et b, appelé le triangle de Maxwell et qui correspond à une représentation graphique des couleurs où $r+g+b=1$.

Le triangle de Maxwell nécessite trois coordonnées trichromatiques. Pour simplifier la représentation des couleurs, on n'utilise que les deux coordonnées r et g, b étant calculable des deux premières puisque la somme de r, g et b est toujours égale à 1. La correspondance en géométrie de cette opération est la projection du triangle de Maxwell sur le plan orthonormé (r,0,g) nommé diagramme CIE-rg. Il s'agit d'une présentation très pratique de l'espace CIE-RGB.

b) Espace colorimétrique CIE-XYZ

- Fonctions colorimétriques XYZ

À partir des primaires R, G et B, la CIE a dérivé de nouvelles primaires irréelles X, Y et Z qui ont les propriétés suivantes :

- Elles produisent toujours des cordonnées trichromatiques positives qui permettent de représenter toutes les couleurs.

- Elles ont été dérivées de façon à ce que des valeurs égales de X, Y et Z produisent le blanc.

- La luminance d'une couleur est déterminée par la primaire Y. En fait, la fonction colorimétrique $\bar{y}(\lambda)$ coïncide avec la courbe de l'efficacité lumineuse spectrale de l'œil V(λ).

La Figure 1-10 ci-dessous montre les courbes des fonctions colorimétriques $\bar{x}(\lambda)$, $\bar{y}(\lambda)$ et $\bar{z}(\lambda)$ de l'observateur standard 2º telles que définit par CIE en 1931. On constate que les trois courbes ont tout le temps des valeurs positives. En 1964, CIE a définit des fonctions colorimétriques supplémentaires $\bar{x}_{10}(\lambda)$, $\bar{y}_{10}(\lambda)$ et $\bar{z}_{10}(\lambda)$ pour un observateur standard 10º.

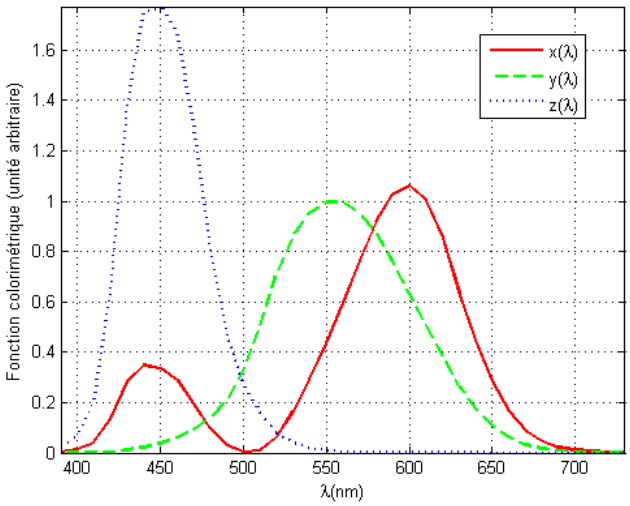


Figure 1-10 : Fonctions colorimétriques CIE-XYZ 1931.

- Coordonnées trichromatiques XYZ

Par synthèse additive, toute couleur C peut être décrite par un vecteur de coordonnées X, Y, Z tel que :

$$\vec{OC} = X.\vec{X} + Y.\vec{Y} + Z.\vec{Z} \quad (1\text{-}24)$$

Les coordonnées X, Y et Z sont les coordonnées trichromatiques XYZ qui peuvent s'exprimer sur une base discrète comme suit :

$$X = \sum_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{x}(\lambda).d\lambda \quad (1\text{-}25)$$

$$Y = \sum_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{y}(\lambda).d\lambda \quad (1\text{-}26)$$

$$Z = \sum_{\lambda_{min}}^{\lambda_{max}} C(\lambda).\bar{z}(\lambda).d\lambda \qquad (1\text{-}27)$$

où $C(\lambda)$ est la distribution spectrale de la couleur C.

En considérant que :

$$C(\lambda) = I(\lambda) \times R(\lambda) \qquad (1\text{-}28)$$

où $I(\lambda)$ est la distribution spectrale de la lumière I et $R(\lambda)$ est la réflectance spectrale de l'objet R de couleur C.

Alors, les trois coordonnées trichromatiques s'expriment par les relations suivantes :

$$X = k.\sum_{\lambda_{min}}^{\lambda_{max}} I(\lambda).R(\lambda).\bar{x}(\lambda).d\lambda \qquad (1\text{-}29)$$

$$Y = k.\sum_{\lambda_{min}}^{\lambda_{max}} I(\lambda).R(\lambda).\bar{y}(\lambda).d\lambda \qquad (1\text{-}30)$$

$$Z = k.\sum_{\lambda_{min}}^{\lambda_{max}} I(\lambda).R(\lambda).\bar{z}(\lambda).d\lambda \qquad (1\text{-}31)$$

où k est un coefficient de normalisation qui s'obtient par la relation suivante :

$$k = \frac{100}{\sum_{\lambda_{min}}^{\lambda_{max}} I(\lambda).\bar{y}(\lambda).d\lambda} \qquad (1\text{-}32)$$

- Conversion RGB ↔ XYZ

La conversion du système RGB vers XYZ et réciproquement est possible grâce aux simples transformations linéaires suivantes :

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.488718 & 0.1762040 & 0.0000000 \\ 0.310680 & 0.8129850 & 0.0102048 \\ 0.200602 & 0.0108109 & 0.9897950 \end{bmatrix} . \begin{bmatrix} R \\ G \\ B \end{bmatrix} \qquad (1\text{-}33)$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 2.370670 & -0.513885 & 0.00529818 \\ -0.900040 & 1.4253000 & -0.0146949 \\ -0.470634 & 0.0885814 & 1.00940000 \end{bmatrix} . \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad (1\text{-}34)$$

- Diagramme de chromaticité CIE-xy

Pour les applications pratiques de colorimétrie et afin de simplifier la représentation et l'analyse des couleurs, la CIE a choisi de représenter les couleurs dans le repère géométrique xy en faisant abstraction de la luminance. Pour cela, il faut calculer les coordonnées normalisées x, y et z de la façon suivante :

$$x = \frac{X}{X+Y+Z} \qquad (1\text{-}35)$$

$$y = \frac{Y}{X+Y+Z} \qquad (1\text{-}36)$$

$$z = \frac{Z}{X+Y+Z} \qquad (1\text{-}37)$$

avec : $x + y + z = 1$

La coordonnée z se déduit directement des deux autres coordonnées x et y. Pour cela, cette coordonnée n'est souvent pas utilisée. Pratiquement, la couleur est spécifiée soit par ses coordonnées trichromatiques X, Y et Z, soit par sa chromaticité x et y et sa luminance Y.

La Figure 1-11 ci-dessous montre la représentation du diagramme de chromaticité CIE-xy 1931. La courbe à la forme de fer à cheval. Les noms approximatifs des zones de couleurs pour un observateur adapté à la lumière du jour sont indiqués sur le diagramme. La courbe reliant les primaires RGB en périphérie du fer à cheval représente les couleurs monochromatiques et la droite qui lie le bleu (400 nm) au rouge (700 nm) est appelée droite des pourpres.

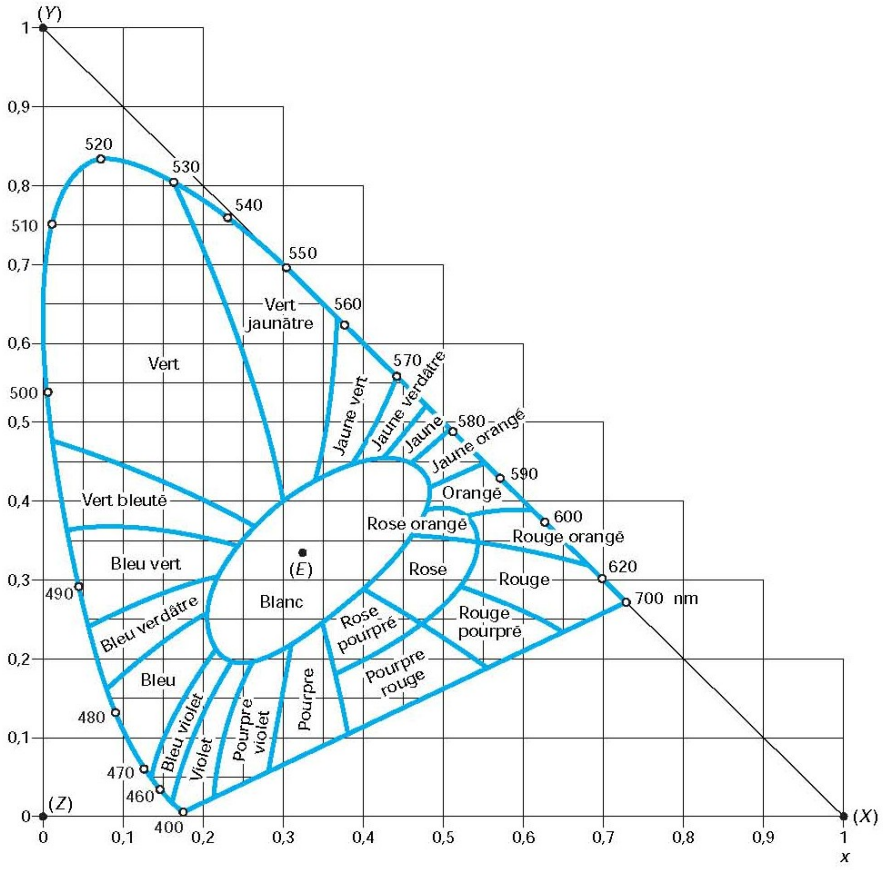


Figure 1-11 : Représentation du diagramme de chromaticité CIE-xy (tiré de [9]).

Le principe d'additivité s'applique au diagramme de chromaticité. Toute couleur à l'intérieur de la courbe en forme de fer à cheval peut être déterminée par le mélange additif d'un stimulus monochromatique et d'une lumière blanche. La couleur résultante se situe sur la droite reliant les deux points. Plus la proportion du blanc dans le mélange additif est importante plus la couleur résultante est désaturée et le point de couleur se situe dans la partie centrale du diagramme. Au contraire, plus la proportion du blanc est faible plus la couleur résultante du mélange additif est saturée lorsque évaluée par un observateur et plus la couleur se rapproche de la courbe des couleurs pures sur le diagramme de chromaticité. Une couleur peut également être le résultat d'un mélange de couleurs autre qu'un mélange de blanc et de couleur pure.

Pour déterminer la couleur complémentaire d'une couleur du spectre, il suffit de tracer la droite qui passe par la couleur choisie et le blanc de référence et de l'étendre dans la direction opposée jusqu'à atteindre la courbe des couleurs monochromatiques. Le lieu d'intersection est la couleur complémentaire.

c) Espace colorimétrique CIE-LAB

- Calcul des coordonnées rectangulaires et cylindriques

L'espace colorimétrique CIE-XYZ, l'espace (x,y,Y) et le diagramme de chromaticité CIE-xy ne sont pas visuellement uniforme parce que des distances égales dans ces espaces ne correspondent pas à des différences perceptives équivalentes entre stimulus de couleur. Alors, la CIE a introduit en 1976 deux nouveaux espaces uniformes, CIE-LAB et CIE-LUV qui résultent d'une transformation mathématique non linéaire du system CIE-XYZ 1931. Le système CIE-LAB possède deux modes de représentation :

- représentation en coordonnées rectangulaire L*a*b* où L* représente l'axe de clarté, a* représente l'axe rouge/vert et b* représente l'axe jaune/bleu.

- représentation en coordonnées cylindriques L* C* hº où L* représente l'axe de clarté, C* représente l'axe de chroma et hº représente l'angle de teinte.

Les coordonnées L*, a* et b* se calculent à l'aide des relations suivante :

$$L^* = 116.f\left(\frac{Y}{Y_n}\right) - 16 \qquad (1\text{-}38)$$

$$a^* = 500.\left[f\left(\frac{X}{X_n}\right) - f\left(\frac{Y}{Y_n}\right)\right] \qquad (1\text{-}39)$$

$$a^* = 200.\left[f\left(\frac{Y}{Y_n}\right) - f\left(\frac{Z}{Z_n}\right)\right] \qquad (1\text{-}40)$$

où : $$f(t) = \begin{cases} t^{1/3} & si : t > \left(\frac{6}{29}\right)^3 \\ \frac{841}{108}.t + \frac{4}{29} & si : t \le \left(\frac{6}{29}\right)^3 \end{cases}$$

et X_n, Y_n et Z_n sont les composantes trichromatiques d'un stimulus blanc de référence.

Les relations pour calculer les composantes cylindriques C* et hº sont les suivantes :

$$C^* = \left[(a^*)^2 - (b^*)^2 \right]^{1/2} \qquad (1\text{-}41)$$

$$h = \arctan\left(\frac{b^*}{a^*} \right) \qquad (1\text{-}42)$$

- Calcul de l'écart de couleur

L'écart de couleur ΔE^*_{ab} entre deux stimulus de couleur se calcul comme la distance euclidienne entre leurs points dans l'espace CIE-LAB. En cordonnées rectangulaires, le calcul se fait selon la relation suivante :

$$\Delta E^*_{ab} = \left[(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2 \right]^{1/2} \qquad (1\text{-}43)$$

où : $\Delta L^* = L^*_1 - L^*_0$

$\Delta a^* = a^*_1 - a^*_0$

$\Delta b^* = b^*_1 - b^*_0$

En coordonnées cylindriques l'écart de couleur se calcul avec la relation suivante :

$$\Delta E^*_{ab} = \left[(\Delta L^*)^2 + (\Delta C^*_{ab})^2 + (\Delta H^*_{ab})^2 \right]^{1/2} \qquad (1\text{-}44)$$

où : $\Delta L^* = L^*_1 - L^*_0$

$\Delta C^*_{ab} = C^*_{ab,1} - C^*_{ab,0}$

$\Delta H^*_{ab} = 2.\left(C^*_{ab,1} . C^*_{ab,0} \right)^{1/2} \sin\left(\frac{h^*_{ab,1} - h^*_{ab,0}}{2} \right)$

CHAPITRE 2 CAPTEURS D'IMAGES NUMERIQUES COULEURS

2.1 Introduction

Un dispositif d'acquisition d'images comme une caméra numérique par exemple est un dispositif électronique capable de capturer l'information lumineuse d'une scène et de la convertir en un signal électrique analogique. Ce signal, appelé signal vidéo, est par la suite échantillonné, quantifié et transformée sous la forme d'une image numérique.

Dans une caméra numérique, la lumière traverse l'objectif pour atteindre son capteur d'images. Ce dernier est l'œil de la caméra produisant le signal vidéo qui correspond point par point à la scène photographiée à l'aide d'un mécanisme de collecte du signal par balayage bidimensionnel. Les capteurs d'images sont omniprésents dans les multiples côtés de nos activités. En fait, on les trouve, en plus des caméras numériques, dans les caméscopes, les scanners, les fax, les photocopieurs, les lecteurs de codes à barres, les systèmes de surveillance, les instruments d'observation spatiaux, les équipements d'imagerie médicale ainsi que dans les systèmes de vision industriels où ils sont utilisés pour le tri et le contrôle de la production.

La capture d'image a beaucoup évolué durant le siècle dernier. Réalisée au début à l'aide de pellicules photosensibles, elle est réalisée de nos jours à l'aide de semi-conducteurs. L'avantage de ce passage est qu'un capteur d'image à semi-conducteur peut être directement intégré avec d'autres systèmes électroniques de façon à ce que ces derniers aillent la possibilité de voir.

Le photodétecteur est l'élément de base du capteur d'image à semi-conducteur. Il est l'élément qui convertit la lumière en signal électrique. Les capteurs d'images contiennent un très grand nombre de photodétecteurs organisés en matrice. Il existe deux principales technologies couramment utilisées de nos jours pour réaliser les capteurs d'images : la technologie CCD (Charge Coupled Devices) et la technologie CMOS (Complementary Metal-Oxyde-Semiconductor).

2.2 Éléments photodétecteurs

2.2.1 Captation de lumière à l'aide de semi-conducteurs

Lorsqu'un photon s'introduit dans un semi-conducteur et que son énergie est supérieure à l'énergie séparant les bandes de valence et de conduction du matériau, il y a une forte probabilité qu'il soit absorbé dans le semi-conducteur et que son énergie soit absorbée par un électron devenant excité. L'électron est excité en passant de la bande énergétique de valence pour occuper la bande de conduction. Ainsi, il se crée une paire électron-trou formant des porteurs en excès et libres de circuler dans le semi-conducteur.

Il existe plusieurs structures électroniques permettant d'utiliser le phénomène de captation de photons dans un semi-conducteur pour mesurer une intensité lumineuse.

2.2.2 Photodiode standard

Du point de vue topologique, la photodiode est le photodétecteur le plus simple. Il a été utilisé dans les premiers capteurs d'images CMOS. Le principe de fonctionnement est simple : avant la capture du signal lumineux, la photodiode est polarisée en inverse; durant le temps d'exposition, les photons incidents génèrent un courant directement proportionnel à la lumière incidente sur la photodiode; la tension résultante aux bornes de la photodiode indique l'intensité de la lumière.

L'équation ci-dessous donne la variation de la tension aux bornes de la photodiode en fonction de la puissance et de la longueur d'onde de l'intensité lumineuse incidente.

$$\frac{dV}{dt} = \frac{1{,}2.\lambda.P.\eta}{c} \qquad (2\text{-}1)$$

où : λ est la longueur d'onde de la lumière incidente et P, sa puissance, η est l'efficacité quantique et c est la vitesse de la lumière.

Il s'agit d'une équation pour une photodiode idéale et ne tient pas compte des dimensions de la photodiode. Il faut aussi noter qu'en pratique, la capacité du contour de la photodiode et les capacités des transistors reliés à celle-ci s'ajoutent à sa capacité et affectent non linéairement la variation de la tension à ces bornes [10].

2.2.3 Photodiode à couche de semi-conducteur intrinsèque

La photodiode à couche de semi-conducteur intrinsèque est connue sous le nom PPD (Pinned Photodiode). Elle a une structure PIN dans laquelle une zone intrinsèque est intercalée entre la région fortement dopée P et la région fortement dopée N formant un sandwich : silicium dopé P/silicium intrinsèque/silicium dopé N comme le montre la Figure 2-1 ci-dessous. Lors de l'exposition d'une PPD à la lumière, la région d'appauvrissement initialement sans porteurs reçoit les porteurs majoritaires générés par la lumière. Ceci a pour effet de diminuer la tension aux bornes de la PPD.

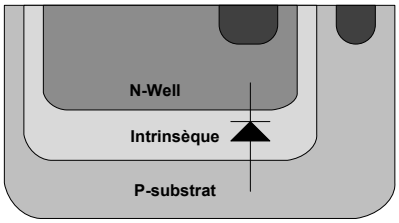


Figure 2-1 : Structure PIN d'une photodiode PPD.

Les avantages d'une photodiode PPD sont un courant de fuite en général plus petit que celui de la photodiode standard et une efficacité quantique meilleur [10]. Cependant, sa fabrication nécessite deux masques de production supplémentaires déviant du coup du procédé CMOS standard et augmentant le coût de fabrication.

2.2.4 Phototransistor

En technologie CMOS, le phototransistor peut être réalisé à l'aide de différentes méthodes de conception : verticale, latérale et circulaire. Bien qu'il s'agit de procédés CMOS, les phototransistors sont des transistors bipolaires, BJT. Comparé à la photodiode, le phototransistor a un gain β qui amplifie le photo-courant et donnant possiblement une meilleur lecture. Cependant, ceci n'est pas avantageux vue qu'il est difficile de contrôler ses caractéristiques lors de la production, ce qui donne une grande variation d'un transistor à un autre. Il en résulte un bruit à patron fixe (FPN) trop élevé pour être acceptable [10].

2.2.5 Photocondensateur

Le photocondensateur, appelé aussi photoMOS ou transistor à photogrille, est très utilisé dans les capteurs d'images modernes. Il est réalisable avec les deux technologies existantes CCD et

CMOS. Le condensateur du photocondensateur est constitué d'une plaque conductrice et transparente formant une grille séparée du substrat semi-conducteur par une couche mince d'oxyde isolant. L'application d'une tension à la grille donne naissance à une zone de déplétion appelée canal dans le semi-conducteur en analogie avec ce qui se passerait sous la grille d'un transistor MOS. Exposé à la lumière, la grille et l'isolant étant tous deux transparents, les photons pénètrent le substrat semi-conducteur et y interagissent avec les électrons. Les charges générées s'accumulent sous la grille. On appel cette région un puits de potentiel.

2.2.6 Photodiode à silicium amorphe

Le procédé utilisé pour fabriquer les photodiodes à silicium amorphe est le procédé TFA (Thin Film on ASIC) qui est formé d'une couche mince constituée de plusieurs étages de silicium amorphe déposée sur un ASIC. Le principal avantage du silicium amorphe est son coefficient d'absorption de la lumière qui est plus élevé que celui du silicium cristallin [10].

2.3 Capteurs d'images numériques

2.3.1 Capteur d'images CCD

Le photodétecteur utilisé dans les capteurs d'images CCD est le photocondensateur. Dans ce type de photodétecteurs les charges générées par l'interaction avec la lumière sont accumulées sous la grille du photocondensateur. L'étape suivante consiste à transférer les charges accumulées vers un convertisseur de charge en tension. Le principe de cette opération est schématisé sur la Figure 2-2 ci-dessous.

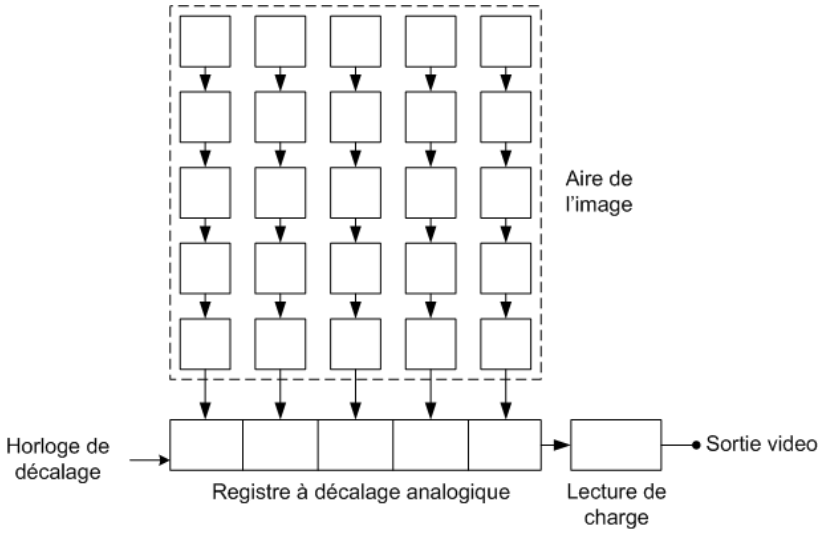


Figure 2-2 : Principe de fonctionnement d'un capteur CCD.

Les photodétecteurs constituant la matrice sensible à la lumière sont disposés à proximité les uns des autres permettant d'intégrer et de déplacer les charges accumulées. Pour balayer toute la matrice, les charges sont décalées lentement à la verticale et rapidement à l'horizontale à travers un registre à décalage analogique. Les charges passent par un lecteur de charge qui converti les charges en une tension constituant le signal vidéo de sortie.

2.3.2 Capteur d'images CMOS

L'élément photodétecteur utilisé dans un capteur d'images CMOS est la photodiode. Le courant de celle-ci est très petit et difficilement manipulable lors de la lecture de la matrice photosensible. Pour résoudre ce problème et afin d'augmenter la sensibilité du capteur, souvent les capteurs utilisant les photodiodes procèdent en mode d'intégration [11]. Le principe d'intégration est illustré sur la Figure 2-3 ci-dessous.

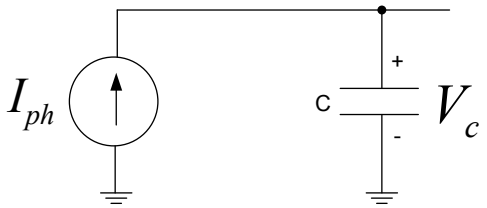


Figure 2-3 : Principe d'intégration dans un capteur CMOS.

Le courant I_{ph} correspondant au courant résultant des photons incidents sur la zone de déplétion de la photodiode charge la capacité C à une tension V_c tel que :

$$I_c = C.\frac{dV_c}{dt} \qquad (2\text{-}2)$$

d'où on peut déduire :

$$dV_c = \frac{1}{C} I_{ph}.dt \qquad (2\text{-}3)$$

En intégrant les deux côtés de l'équation entre les temps d'intégration t1 et t2, on obtient :

$$\int_{t1}^{t2} dV_c = \frac{I_{ph}}{C}.\int_{t1}^{t2} dt \qquad (2\text{-}4)$$

d'où :

$$\Delta V_c = \frac{\Delta t}{C}.I_{ph} \qquad (2\text{-}5)$$

Selon la dernière équation ci-dessus, le courant de la photodiode est amplifié avec un gain de $\frac{\Delta t}{C}$.

On remarque que l'augmentation du temps d'intégration Δt permet d'amplifier d'avantage le courant I_{ph} de façon à détecter de faibles courants.

2.3.3 Comparaison entre les technologies CCD et CMOS

La technologie CCD offre de très bonnes performances. La détection de la lumière et le transfert des charges s'effectuent avec un niveau de bruit faible et uniforme. Il s'agit d'une technologie mature et qui a atteint des niveaux de rendement et de performances près des limites théoriques et cela fait des années qu'elle n'a pas connue d'améliorations significatives. Cependant, cette technologie a des points faibles : un sous échantillonnage de la matrice photosensible n'est pas possible, elle nécessite plusieurs niveaux de tensions élevés, la consommation en courant est élevée à cause des charges capacitives élevées que représentent les dispositifs CCD et enfin le procédé CCD est un procédé spécial couteux et qui ne permet pas d'intégrer au capteur d'autres systèmes.

La technologie CMOS offre plusieurs avantages qui remédient aux points faibles de la technologie CCD. En fait, la technologie CMOS présente un faible coût puisque les usines de

fabrication CMOS sont abondantes, la consommation en puissance est faible, il est possible de sous échantillonner la matrice photosensible et il est possible d'intégrer facilement d'autres circuits à même la puce du capteur. Par contre, au niveau de la qualité de l'image, les capteurs CMOS se comparent aux capteurs CCD de milieu de gamme. Ceci est dû à leur plus faible sensibilité à la lumière et leur plus grand courant de noirceur.

2.4 Détection de la couleur

2.4.1 Réponse spectrale d'un photodétecteur

Un capteur d'image à base de semi-conducteur est, comme le montre la Figure 2-4 ci-dessous, achromatique par nature. Dans l'exemple de la Figure 2-4 ci-dessous, le pixel CMOS est sensible à jusqu'à une longueur d'onde de 1050 nm. Pour générer une image couleur, trois types de pixels : rouge, vert et bleu ou cyan, magenta et jaune sont nécessaires.

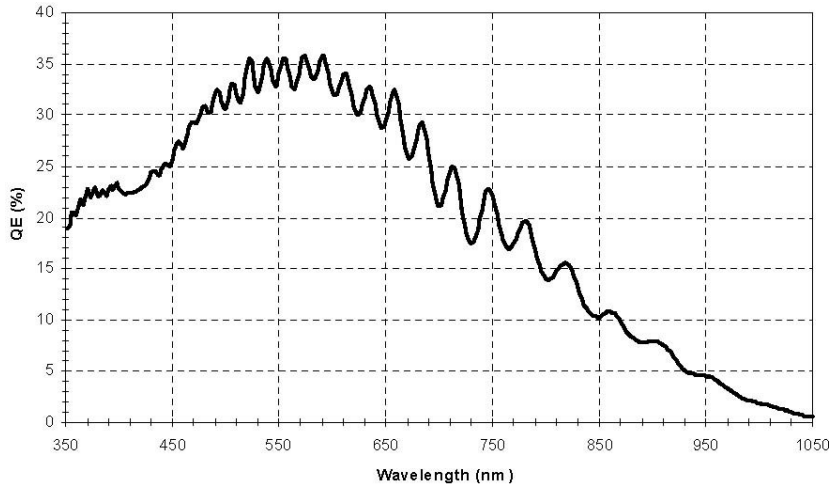


Figure 2-4 : Réponse spectrale d'un photodétecteur CMOS (tiré de [12]).

Avant l'invention du capteur d'images couleur, l'image couleur était créé en superposant trois images identiques capturées chacune avec un filtre de couleur différent. Les trois filtres de couleurs primaires différentes sont placés devant trois capteurs d'images identiques. L'image à capturer était décomposée en trois images identiques à l'aide d'un système optique.

2.4.2 Filtre de Bayer

Au début des années 70s, le scientifique Bryce Bayer de Kodak avait réalisé qu'une matrice de filtres de couleurs appelée en anglais CFA (Color Filter Array) permet la reconstruction de toutes les couleurs d'une scène à l'aide d'un seul capteur d'images. Cette matrice de filtres qui a pris le nom de filtre de Bayer est superposée aux pixels du capteur d'images. Les deux patrons de couleurs les plus communs pour le filtre de Bayer sont représentés sur la Figure 2-5 ci-dessous. Le patron du filtre RGB est constitué de 50% vert, 25% rouge et 25% bleu. La proportion élevée du vert vient d'une volonté d'imiter le système visuel humain qui est plus sensible à la lumière verte.

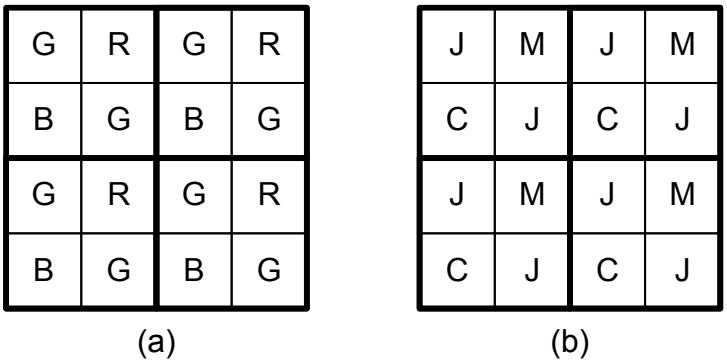


Figure 2-5 : Patrons de Bayer :

(a) RGB; (b) CMJ.

Puisque chaque pixel est filtré pour n'enregistrer qu'une des trois couleurs primaires, chaque pixel ne permet pas de déterminer la couleur du dit pixel. Une interpolation est nécessaire pour déterminer les deux autres primaires manquantes pour chaque pixel.

Le filtre de Bayer est largement utilisé dans les capteurs d'image mono-capteur présents dans les appareils photographiques numériques, les caméras vidéo et les scanners.

L'utilisation du filtre de Bayer a plusieurs inconvénients dont on peut citer : les pertes de luminosité encourues dans les filtres optiques réduisent le rendement quantique de chaque pixel; l'ajout de la couche de filtres optiques augmente les coûts de fabrication du capteur; l'étape d'interpolation pour déduire les primaires manquantes pour chaque pixel complique le traitement d'images et introduit des erreurs et des artefacts dans l'image finale.

2.4.3 Capteur multicouches de Foveon

En 2002, Foveon, une compagnie basée à Santa Clara en Californie et racheté en 2008 par Sigma, a développé un nouveau capteur d'images couleur qui se passe du filtre de Bayer [2, 13]. Le nouveau capteur exploite la propriété du silicium selon laquelle le coefficient d'absorption de la lumière varie en fonction de la longueur d'onde et de la profondeur. Le silicium est utilisé comme filtre vertical. Pour capturer les charges générées en différentes profondeur, trois photodiodes sont implantées à différentes profondeurs pour discriminer les longueurs d'onde du bleu, vert et rouge comme le montre la Figure 2-6 ci-dessous.

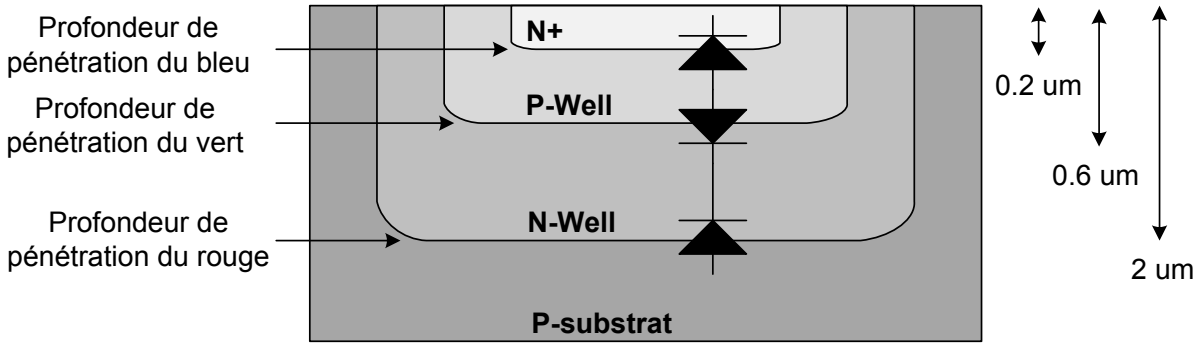


Figure 2-6 : Pixel de Foveon.

Les réponses spectrales des trois photodiodes enfouies sont représentées sur la Figure 2-7 ci-dessous.

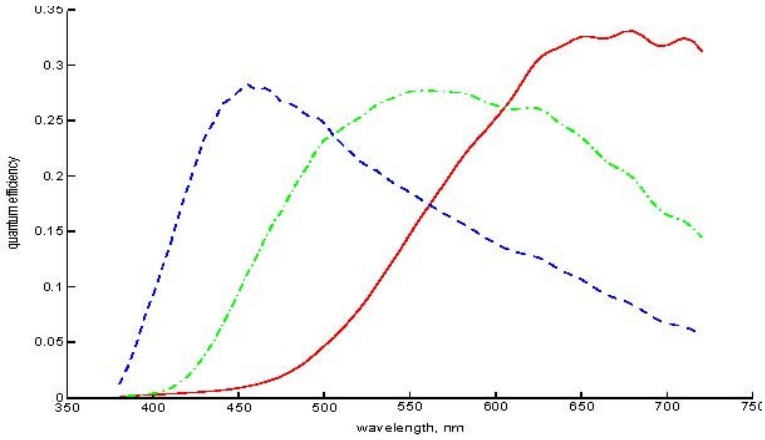


Figure 2-7 : Réponses spectrales du capteur de Foveon (tiré de [14]).

Les avantages du capteur de Foveon sont qu'il n'utilise pas le filtre de bayer, chaque pixel détecte les trois couleurs primaires et que par conséquent, on n'a pas besoin de faire de l'interpolation

lors du traitement de l'image couleur. En contre partie, l'inconvénient majeur est que la fabrication du capteur est très couteuse parce qu'elle n'utilise pas le procédé CMOS standard.

2.5 Nouveau capteur d'images couleur sans filtre optique

Le nouveau capteur d'images couleur sans filtre optique sur lequel travaille l'équipe de recherche de mon directeur de recherche utilise, comme pour le capteur de Foveon, la propriété d'un semi-conducteur selon laquelle la profondeur de pénétration des ondes électromagnétiques varie avec la longueur d'onde. La Figure 2-8 ci-dessous montre l'absorption de trois flux lumineux bleu, vert et rouge en fonction de la profondeur dans du silicium.

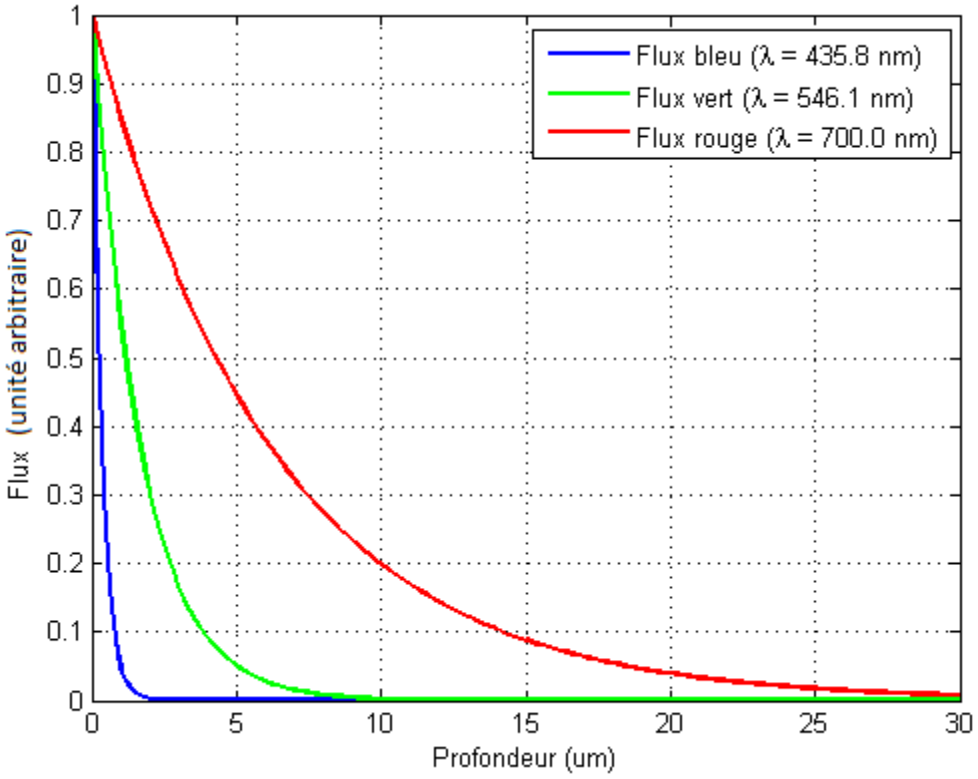


Figure 2-8 : Absorption de trois flux lumineux bleu, vert et rouge dans du silicium.

Le bleu est absorbé progressivement jusqu'à une absorption totale à une profondeur d'environ 2 µm. Le vert n'est totalement absorbé qu'à une profondeur d'environ 6 µm. Le rouge quand à lui, pénètre profondément dans le silicium et est absorbé progressivement jusqu'à une absorption totale à une profondeur d'environ 30 µm.

La Figure 2-9 montre le principe de fonctionnement du nouveau capteur d'images en montrant la structure d'un pixel pouvant détecter trois couleurs en même temps à l'aide de trois électrodes C_1, C_2 et C_3. Le flux lumineux entrant par la fenêtre du pixel génère en profondeur des photoélectrons en fonction de sa composition spectrale. Les électrons générés suivent le champ électrique E généré par les deux tensions V_{e1} et V_{e2} pour être collectés selon la profondeur de leurs sources. Ainsi, l'électrode C_1 collecte les électrons générés prés de la surface, l'électrode C_2 collecte les électrons plus profonds et l'électrode C_3 collecte les électrons plus profonds que ceux collecté par l'électrode C_2. Trois couleurs sont ainsi discriminées.

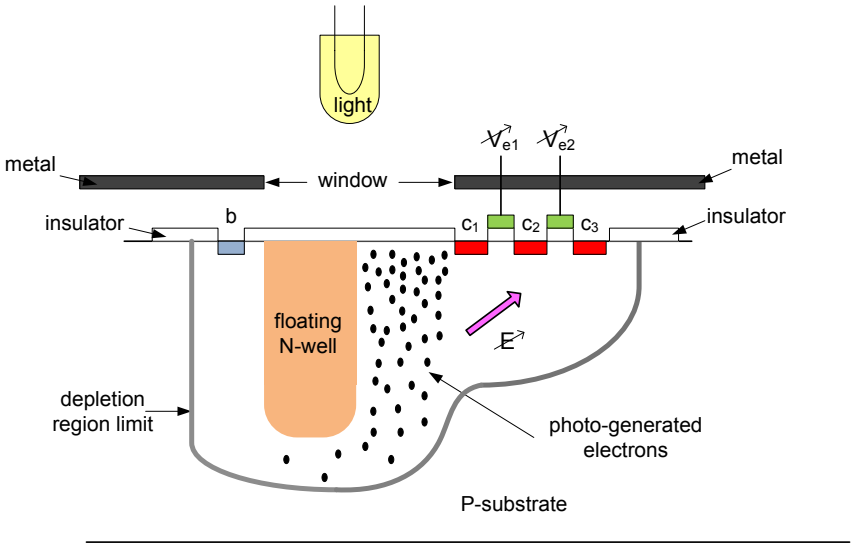


Figure 2-9 : Structure d'un pixel du capteur novateur (tiré de [15]).

Par rapport au capteur de Foveon, ce capteur à l'avantage d'être fabricable par un procédé CMOS standard.

2.6 Évaluation de la sensibilité spectrale d'un capteur d'images couleur

La sensibilité spectrale, sensibilité en fonction de la longueur d'onde de la source lumineuse à flux énergétique constant, d'un capteur d'images couleur joue un rôle important dans la qualité de l'image capturée finale. Un capteur d'image est capable de différencier les couleurs comme le fait l'œil humaine si ces réponses spectrales peuvent être obtenues par une combinaison linéaire des fonctions colorimétriques de l'observateur standard. Cette condition est appelée la condition Luther. En réalité, les capteurs d'images ne rencontrent jamais cette condition. Ceci implique qu'il y a des couleurs que le capteur d'images perçoit de façon différente de comment l'œil

humain les perçoit. Dans la littérature, il existe plusieurs métriques qui permettent d'évaluer la réponse spectrale d'un capteur. Le concept du facteur de qualité a été introduit pour la première fois par Neugebauer. À ce jour, il y a plusieurs facteurs de qualité qui ont été proposés. Ces métriques se divisent en deux catégories. La première catégorie regroupe les métriques décrivant la différence géométrique entre les fonctions colorimétriques de l'observateur standard et les réponses spectrales du capteur à l'étude. Dans cette catégorie, on trouve le facteur q de Neugebauer [16] pour l'évaluation d'un canal du capteur à la fois et son extension, le facteur μ par Vora-Trussell [17] pour l'évaluation d'un capteur multi-spectral à un nombre n de canaux. La deuxième catégorie de ces métriques utilise le calcul de l'erreur en couleur minimale pour un ensemble d'échantillons de spectres de réflectance dans les espaces de couleur CIE. Dans cette deuxième catégorie, on trouve : les facteurs Q_{st} et Q_{sf} de Shimano [18] dans lesquels l'erreur en couleur moyenne est minimisée dans l'espace CIE-XYZ sans considérer le bruit; plusieurs indices de Tajima [19]; l'indice de rendu de couleur CRI (Color Rendering Index) de Hung [20]; la figure de mérite FOM (Figure of Merit) de Sharma-Trussell [21] où l'erreur en couleur est minimisée dans un espace de couleur perceptuellement uniforme en tenant compte du bruit blanc dans le processus de capture d'images; et son extension par Quan et all [22, 23] à la mesure de qualité unifiée UMG (Unified Measure of Goodness) qui inclut les bruits dépendant du signal et indépendant du signal.

En 2006, ISO a définit dans son standard 17321 [24] l'indice de métamérisme de sensibilité SMI (Sensitivity Metamerism Index). Il est représenté par un nombre inférieur à 100 et peut avoir des valeurs négatives. Une valeur de 100 indiquerait que la condition Luther est atteinte. Une valeur de 50 correspond à la différence entre un illuminant du jour, D65, et un illuminant généré par un tube fluorescent, étant considéré comme une erreur modérée.

2.7 Conversion des couleurs en considérant le bruit

2.7.1 Principe

La conversion des couleurs est une étape importante dans la chaine de traitement d'une image brute capturée par un capteur d'images. Elle permet de convertir l'image brute ayant des couleurs appartenant à l'espace de couleurs du capteur en valeurs dans un espace colorimétrique standard indépendant du capteur. Plusieurs espaces colorimétriques sont possibles comme CIE 1931 XYZ

(observateur 2º), CIE-LAB, CIE-LUV et les espaces RGB (CIE-RGB, sRGB, AdobeRGB, RIMMRGB …).

La conversion des couleurs est réalisée à l'aide d'une matrice de conversion des couleurs M, une matrice 3x3. Elle est réalisée de la façon suivante :

$$\begin{bmatrix} R_{sRGB} \\ G_{sRGB} \\ B_{sRGB} \end{bmatrix} = M.\begin{bmatrix} S_{camera} \\ M_{camera} \\ L_{camera} \end{bmatrix} \quad (2\text{-}6)$$

avec : $M = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$

Le but d'une méthode de conversion des couleurs est de trouver la relation linéaire entre l'espace de couleurs du capteur d'images et l'espace colorimétrique de destination. L'hypothèse sous-jacente est qu'une telle transformation linéaire existe. Il s'agit de faire correspondre les mesures du capteur avec leurs valeurs dans l'espace colorimétrique choisi pour destination.

Dans la méthode présentée ci-dessous, la transformation entre les deux espaces colorimétriques est déterminée en minimisant l'erreur quadratique entre les valeurs théoriques des échantillons de couleur et celles mesurées par le capteur d'images et le niveau du bruit dans l'image après la conversion des couleurs.

2.7.2 Détermination de l'équation pour le calcul de la matrice M [25]

La relation entre la valeur d'un pixel dans l'espace colorimétrique de destination et la mesure par le capteur d'image est la suivante :

$$\hat{p} = M.q \quad (2\text{-}7)$$

où : M est la matrice 3x3 de conversion de la couleur, q est un vecteur 3x1contenant la mesure et p est le vecteur 3x1 de la valeur théorique de la couleur du pixel.

L'erreur quadratique totale ε_C entre les couleurs théoriques des échantillons et les couleurs estimées est donnée par :

$$\varepsilon_C = \sum_{i=1}^{k} \|p_i - \hat{p}_i\|^2$$

$$= \sum_{i=1}^{k} \|p_i - Mq_i\|^2$$

$$= tr([P - MQ]^T [P - MQ]) \qquad (2\text{-}8)$$

où : $\| \ \|$ représente la norme L2, P est une matrice 3xk contenant les valeurs théoriques des couleurs de k échantillons de couleurs, Q est une matrice 3xk contenant les mesures prises par le capteur d'images pour les k échantillons de couleurs et tr est l'opérateur trace() calculant la somme des coefficients de la diagonale de la matrice opérande.

En considérant Σ_D la matrice 3x3 de covariance caractérisant le bruit mesuré dans le capteur d'images et en considérant la propagation d'erreur dans un système linéaire, la matrice de covariance du bruit transformé dans l'espace colorimétrique sRGB, Σ_N, se calcul par la relation suivante :

$$\Sigma_N = M\Sigma_D M^T \qquad (2\text{-}9)$$

Particulièrement, les coefficients de la diagonale de la matrice Σ_N caractérisent la variance du bruit pour les trois canaux de couleur dans l'espace sRGB. L'ampleur du bruit dans les couleurs converties dépend de ces coefficients de la diagonale.

La fonction coût à minimiser pour trouver un compromis entre l'erreur en couleurs et la variance du bruit est la suivante :

$$J(M) = \varepsilon_C + w\varepsilon_N$$

$$= tr([P - MQ]^T [P - MQ]) + w.tr\left(M\Sigma_D M^T\right) \qquad (2\text{-}10)$$

Où : ε_N est le total de la variance du bruit dans les couleurs convertis et w est un facteur qui permet de faire le compromis entre erreurs de couleurs minimales et bruit minimal.

Pour déduire la matrice M qui minimise la fonction coût, il faut dériver cette dernière par rapport à M et mettre le résultat égal à zéro comme ce qui suit :

$$0 = \frac{\partial J(M)}{\partial M}$$

$$= \frac{\partial}{\partial M}\left\{tr([P-MQ]^T[P-MQ]) + w.tr\left(M\Sigma_D M^T\right)\right\}$$

$$= \frac{\partial}{\partial M}\left\{tr(P^TP - P^TMQ - Q^TM^TP + Q^TM^TMQ) + w.tr\left(M\Sigma_D M^T\right)\right\}$$

$$= -\frac{\partial}{\partial M}\left\{tr(P^TMQ)\right\} - \frac{\partial}{\partial M}\left\{tr(Q^TM^TP)\right\} + \frac{\partial}{\partial M}\left\{tr(Q^TM^TMQ)\right\} +$$

$$w.\frac{\partial}{\partial M}\left\{tr\left(M\Sigma_D M^T\right)\right\}$$

$$= -\frac{\partial}{\partial M}\left\{tr(QP^TM)\right\} - \frac{\partial}{\partial M}\left\{tr(PQ^TM^T)\right\} + \frac{\partial}{\partial M}\left\{tr(MQQ^TM^T)\right\} +$$

$$w.\frac{\partial}{\partial M}\left\{tr\left(M\Sigma_D M^T\right)\right\}$$

$$= -PQ^T - PQ^T + 2MQQ^T + 2w.M\Sigma_D$$

d'où finalement :

$$MQQ^T + w.M\Sigma_D = PQ^T \quad (2\text{-}11)$$

Les propriétés utilisées pour arriver à l'équation (3-6) ci-haut sont les suivantes :

$$tr(A+B) = tr(A) + tr(B)$$

$$tr(ABC) = tr(CAB) = tr(BCA)$$

$$\frac{\partial}{\partial X}tr(AX) = A^T$$

$$\frac{\partial}{\partial X}tr(AX^T) = A$$

$$\frac{\partial}{\partial X}tr(XAX^T) = X\left(A + A^T\right)$$

De l'équation (3-6), on obtient la relation permettant de calculer la matrice de conversion des couleurs M :

$$M = PQ^T \left[QQ^T + w.\Sigma_D \right]^{-1} \quad (2\text{-}12)$$

CHAPITRE 3 CONVERSION DES COULEURS ET PROTOTYPE DE DÉMONSTRATION

3.1 Introduction

L'équipe de recherche de mon directeur de recherche a conçu plusieurs générations du nouveau capteur d'images CMOS sans filtre optique. Dans le cadre de ma maitrise, j'ai travaillé principalement avec le prototype E de ce capteur. J'ai commencé par implémenter une méthode de conversion des couleurs tenant compte du bruit présent dans le capteur d'images. L'implémentation est réalisée sur Matlab sous la forme d'un GUI (Graphical User Interface). Ce GUI calcule une matrice dite matrice de conversion des couleurs optimisée pour minimiser l'erreur en couleur et sans amplifier le bruit présent dans l'image captée. L'étape suivante était de concevoir en VHDL le circuit numérique permettant d'appliquer la matrice de conversion aux couleurs du capteur d'images.

Le travail suivant était de concevoir un prototype de démonstration sur une carte de développement FPGA jumelée à un écran LCD couleur tactile. Ce prototype contient des circuits de contrôle, d'acquisition et d'affichage contrôlés par un processeur NIOS II. Une carte électronique a été développée qui permet la conversion des signaux analogiques du capteur en signaux numériques et qui fournit également différentes tensions de polarisation nécessaires au fonctionnement du capteur.

3.2 Conversion des couleurs

3.2.1 Principe

La conversion des couleurs permet de convertir l'image brute ayant des couleurs appartenant à l'espace de couleurs du capteur en valeurs dans un espace colorimétrique standard indépendant du capteur. L'espace colorimétrique sRGB est le standard largement adopté par l'industrie de l'image et de la vidéo. D'où l'intérêt de faire la conversion directement à cet espace de couleur. La conversion est réalisé à l'aide d'une méthode standard qui est l'utilisation d'une matrice de conversion 3x3.

Le problème majeur dans cette méthode est la nature de cette matrice pour notre type de capteurs qui peut avoir de grands coefficients dans la diagonale [26]. Or, ceux-ci sont responsables de la dégradation des performances en ce qui concerne le bruit. En fait, lorsque ces coefficients de la diagonale sont grands, la matrice de conversion amplifie le bruit contenu dans l'image originale. D'où, l'importance de tenir compte du bruit lors de la détermination des coefficients de la matrice M. Pour cette raison, on a opté pour la méthode de conversion des couleurs en considérant le bruit présentée dans la section 2.7.

Dans l'opération de conversion des couleurs, il s'agit de faire correspondre les mesures du capteur avec leurs valeurs dans l'espace colorimétrique choisi pour destination. Pour y arriver, une mire couleur, GretagMacbeth ColorChecker, est utilisée comme scène pour constituer des échantillons de couleurs. La mire GretagMacbeth ColorChecker est constituée de 24 échantillons carrés de 4 cm de côté, distribués sur 4 lignes. Les valeurs des triplets colorimétriques sRGB de ces échantillons sont parfaitement connues. La Figure 3-1 ci-dessous montre la mire GretagMacbeth.

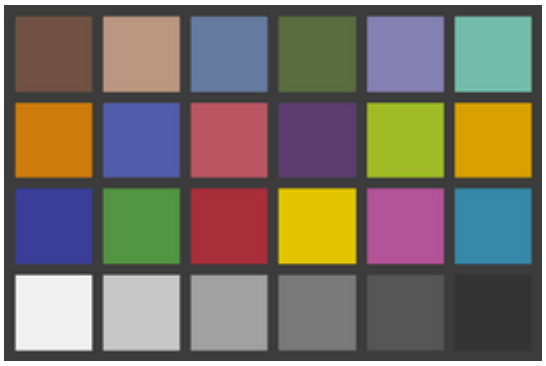


Figure 3-1 : La mire GretagMacbeth ColorCheker (tiré de [27]).

3.2.2 Implémentation Matlab

La détermination de la matrice de correction des couleurs commence par la prise d'une photo de la mire GretagMacbeth Color Cheker avec le capteur d'images. Cette image est le point de départ pour l'algorithme de calcul de la matrice de conversion M suivant :

> *1- Construire la matrice P de dimensions 3x24 contenant les valeurs théoriques des échantillons de couleurs de la mire.*

$$P = \begin{bmatrix} T\_R_1 & T\_G_1 & T\_B_1 \\ T\_R_2 & T\_G_2 & T\_B_2 \\ . & . & . \\ . & . & . \\ T\_R_k & T\_G_k & T\_B_{24} \end{bmatrix}$$

2- Construire la matrice Q de dimension 3x24 contenant les valeurs mesurées dans l'image capturée. Il s'agit des valeurs moyennes pour les pixels de chaque échantillon de couleur.

$$Q = \begin{bmatrix} M\_R_1 & M\_G_1 & M\_B_1 \\ M\_R_2 & M\_G_2 & M\_B_2 \\ . & . & . \\ . & . & . \\ M\_R_k & M\_G_k & M\_B_{24} \end{bmatrix}$$

3- Fixer la valeur de w, le coefficient pour faire le compromis entre minimiser les erreurs de couleur et minimiser le bruit.

4- Estimation de la matrice de covariance caractérisant le bruit du capteur à partir de l'image captée.

$$\Sigma_D = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix}$$

Le calcul de Σ_D est effectué selon les sous-étapes suivantes :

- Filtrer le bruit de l'image capturée en appliquant un filtre gaussien.

- Isoler le bruit en soustrayant l'image filtrée de l'image capturée.

- Construire la matrice des échantillons de bruits N.

$$N = \begin{bmatrix} N\_R_1 & N\_G_1 & N\_B_1 \\ N\_R_2 & N\_G_2 & N\_B_2 \\ . & . & . \\ . & . & . \\ N\_R_n & N\_G_n & N\_B_n \end{bmatrix}$$

- Calculer la matrice de covariance à l'aide de la commande Matlab 'cov' :

$$\Sigma_D = \mathrm{cov}(N)$$

5- Calculer M à l'aide de la relation suivante :

$$M = PQ^T\left[QQ^T + w.\Sigma_D\right]^{-1}$$

6- Normaliser la matrice de conversion M pour avoir les sommes horizontales des coefficients égales ou inférieures à un.

Dans les sous-étapes de l'étape 4, le bruit isolé de l'image capturé est un bruit aléatoire à distribution gaussienne.

L'algorithme de conversion des couleurs est implémenté sous Matlab sous la forme d'un GUI. Le code source Matlab est donné aux annexes 1 à 3. La Figure 3-2 ci-dessous montre l'interface GUI.

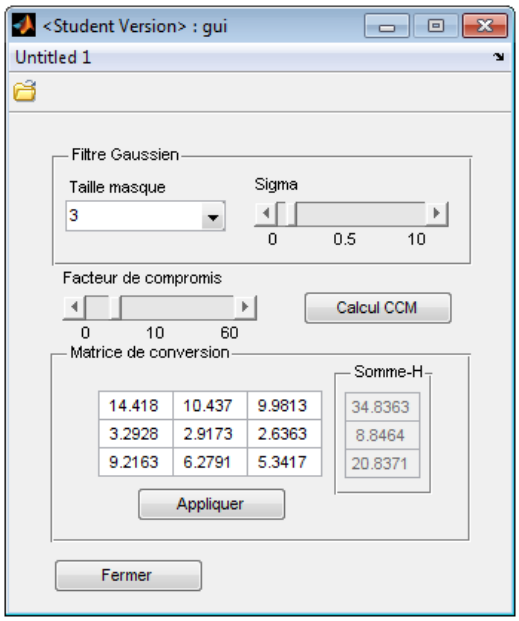


Figure 3-2 : L'interface GUI Matlab.

L'interface GUI est simple d'utilisation. Elle permet de choisir la taille du masque du filtre Gaussien et de son paramètre la déviation standard σ. Elle permet également de choisir la valeur du facteur de compromis *w*. Pour utiliser le GUI, il faut commencer par ouvrir l'image de la mire Macbeth prise par le capteur d'image en cliquant sur l'icône 'ouvrir un fichier' en haut à gauche

du GUI. Par la suite, il faut cliquer sur le bouton '*Calcul CCM*'. Ensuite, il faut placer interactivement 24 rectangles de sélection sur l'image ouverte pour indiquer à l'outil où se trouvent les 24 échantillons de couleurs constituants la mire Macbeth. La Figure 3-3 ci-dessous montre un exemple de résultat de cette opération. À la fin de l'opération, la matrice de conversion sera calculée et affichée sur le GUI dans la partie qui lui est dédiée. Le bouton '*Appliquer*' permet d'appliquer la matrice de conversion des couleurs à l'image ouverte et de visualiser le résultat.

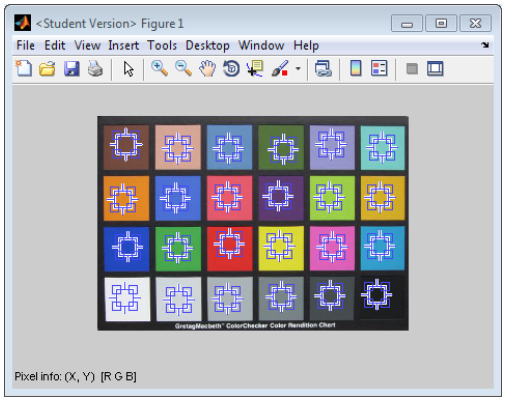


Figure 3-3 : Placement interactif de 24 rectangles de sélection.

3.2.3 Résultats

Nous avons capturé l'image de la mire GretagMacbeth à l'aide du nouveau capteur d'images en utilisant une carte de capture de Matrox. À cause de la taille limitée du capteur d'image, 120x90 pixels, nous avons divisé la mire en quatre zones égales et nous avons capturé une zone à la fois. À la fin, nous avons reconstruit la mire comme la montre la Figure 3-4 ci-dessous.

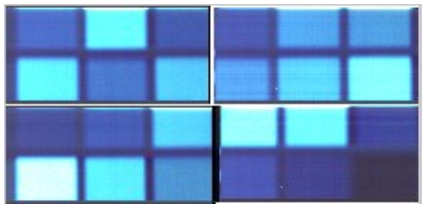


Figure 3-4 : Mire de Macbeth capté par le nouveau capteur d'images.

Le calcul de l'écart en couleurs dans l'espace uniforme CIE-Lab entre les couleurs de cette image et les échantillons de la mire de Macbeth originale montre que l'écart maximal est ΔEab_{max}=121.7249 et l'écart moyen est égale à ΔEab_{moyen}=72.4181.

Par la suite, nous avons passé l'image de la mire dans le GUI de calcul de la matrice de conversion des couleurs. La matrice de conversion obtenue pour : H=5, σ=0.5 et w=10 est la suivante :

$$M = \begin{bmatrix} 0.2524 & 0.5515 & -0.0761 \\ 0.6053 & -0.0597 & 0.2021 \\ 0.4754 & -0.4532 & 0.5246 \end{bmatrix}$$

L'application de la conversion des couleurs sur l'image de la mire de Macbeth capturé a donné le résultat montré à la Figure 3-5 ci-dessous.

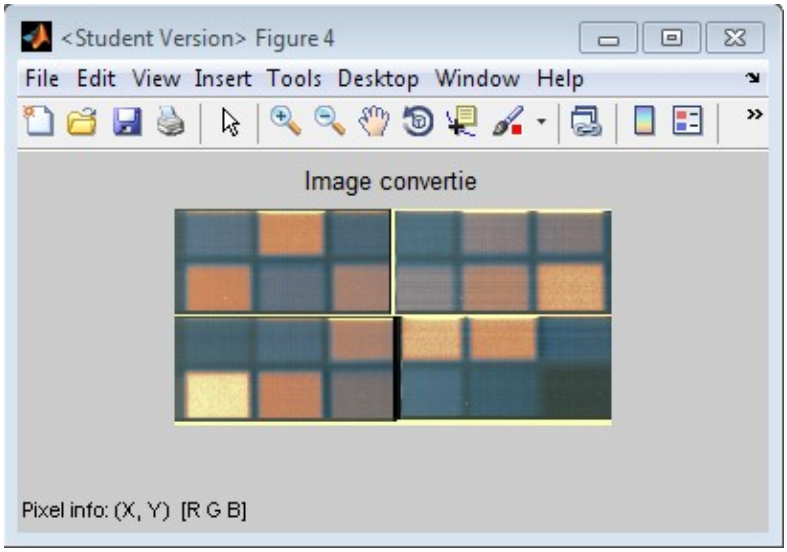


Figure 3-5 : Résultat de la conversion des couleurs.

Le résultat de la conversion montre une image qui n'est pas fidèle en couleur à la mire de Macbeth originale. La Figure 3-6 ci-dessous montre les écarts en couleurs dans l'espace uniforme CIE-Lab entre les coordonnées CIE-Lab de la mire originale et les coordonnées dans le même espace des couleurs obtenues après la conversion des couleurs. Les écarts constatés sont grands : l'écart maximal est ΔEab_{max}=69.3877 et l'écart moyen est égale à ΔEab_{moyen}=39.0281. L'écart maximal est passé de 121.7249 dans l'image originale à 69.3877 dans l'image convertie. Cependant, cet écart demeure grand.

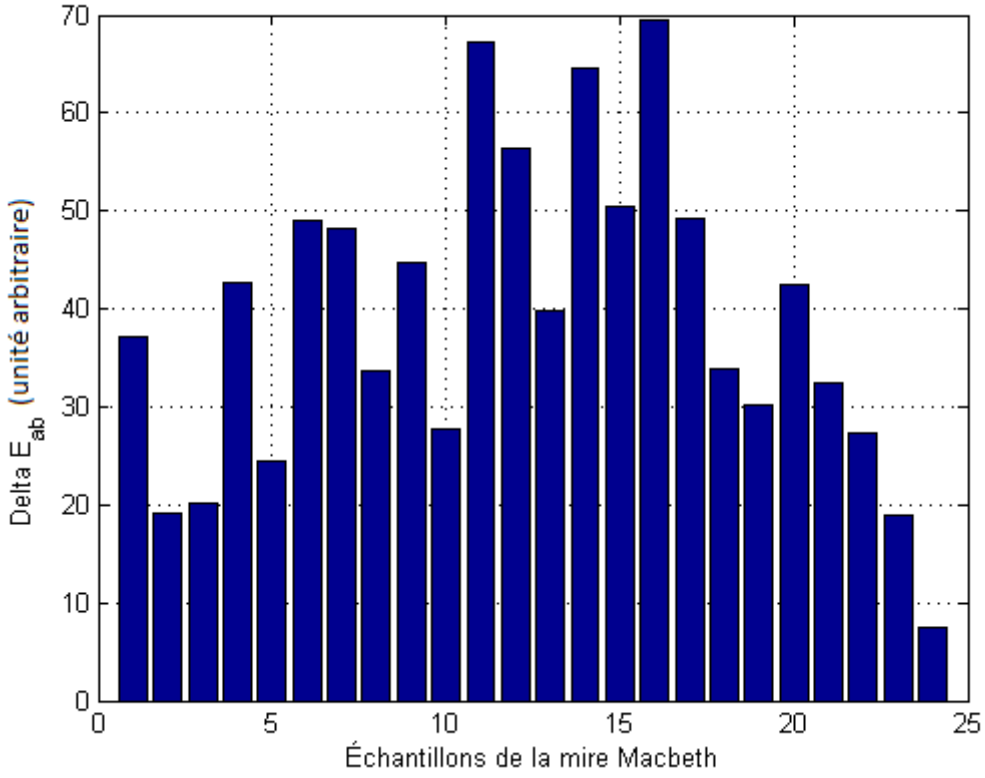


Figure 3-6 : Écarts en couleurs de l'image convertie.

Pour comprendre mieux le problème, les réponses spectrales des trois collecteurs du capteur ont été tracées à l'aide d'un monochromateur. Le graphique résultant est montré à la Figure 3-7 ci-dessous. Les réponses spectrales ont été mesurées pour des longueurs d'ondes entre 380nm et 780nm.

Ces réponses montrent des courbes qui n'ont pas des pics distincts. En calculant l'indice de métamérisme de sensibilité (SMI) du capteur, nous avons trouvé un SMI égal à 13,37. Cet indice a été calculé selon la méthode décrite dans le standard ISO 17321-1 [24]. On remarque que l'indice de métamérisme de ce capteur est faible signifiant que le capteur ne reproduit pas les couleurs fidèlement.

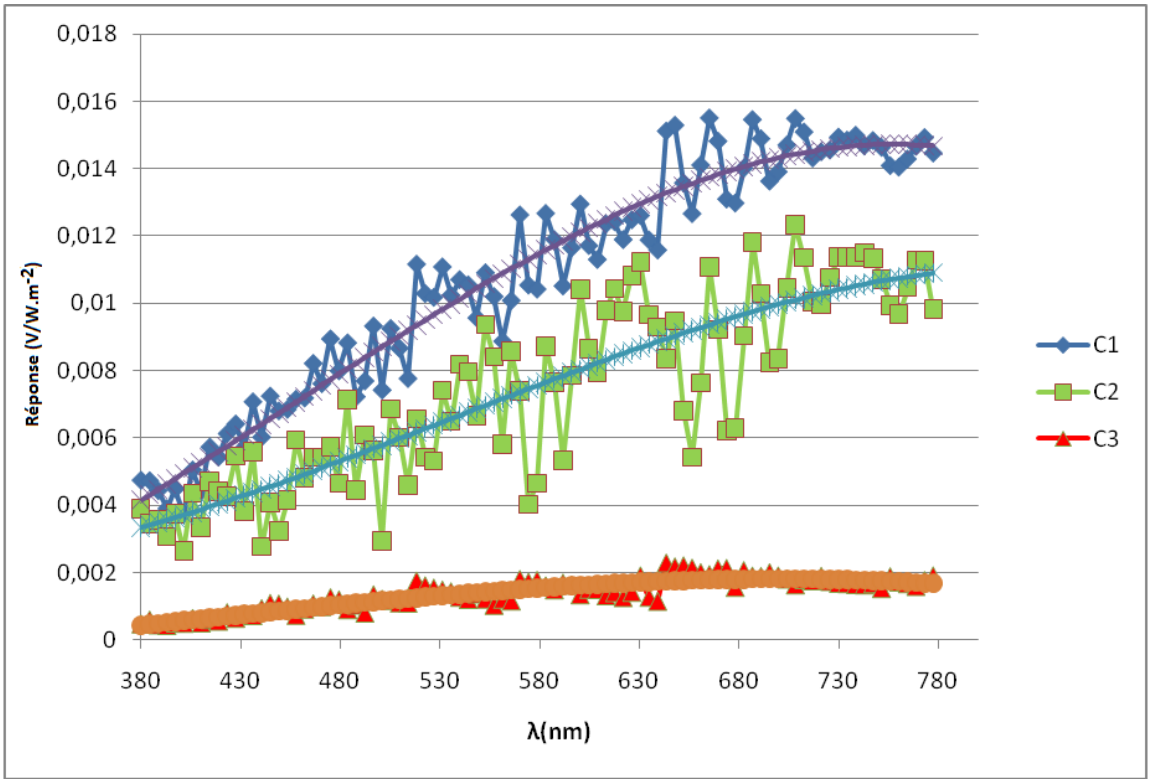


Figure 3-7 : Réponses spectrales des trois collecteurs du capteur.

L'indice SMI et les autres facteurs de qualité décrits dans la section 2.6 permettent d'évaluer les courbes spectrales d'un capteur d'image à l'aide d'un chiffre. Cependant, ils ne donnent aucune information sur la déficience du capteur pour la reproduction de certaines gammes de couleurs ou de distinguer l'intervalle de couleur médiocrement représenté par le capteur. Une méthode que nous avons développée et décrite dans la section qui suit est une tentative de surmonter ces inconvénients.

3.3 Nouvelle méthode d'évaluation de la reproduction des couleurs

3.3.1 Principe

Dans la méthode proposée pour l'évaluation de la fidélité en couleur d'un capteur d'images, la première étape consiste à générer un grand nombre de couleurs aléatoirement. La Figure 3-8 montre le spectre d'une couleur aléatoire de cette étape. Le spectre de chaque couleur aléatoire est produit en générant un nombre aléatoire de raies dont les positions dans le spectre visible sont aléatoires et dont les amplitudes sont aussi aléatoires.

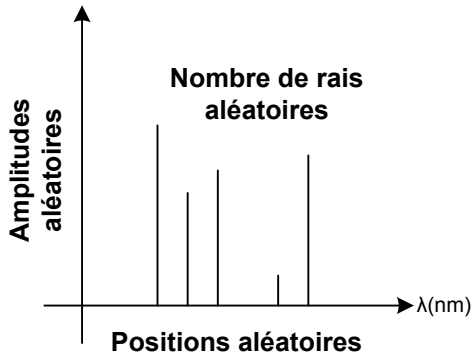


Figure 3-8 : Génération des couleurs aléatoires.

L'étape suivante résumée sur la Figure 3-9 ci-dessous consiste à calculer les composantes X, Y et Z dans l'espace de couleur CIE-XYZ de chaque couleur aléatoire en utilisant les fonctions colorimétrique CIE-XYZ. Ce calcul se fait en utilisant les équations 1-25 à 1-27. Dans la même étape, on calcul les composantes C1, C2 et C3 des couleurs aléatoires dans l'espace colorimétrique du capteur en utilisant les réponses spectrales du capteur.

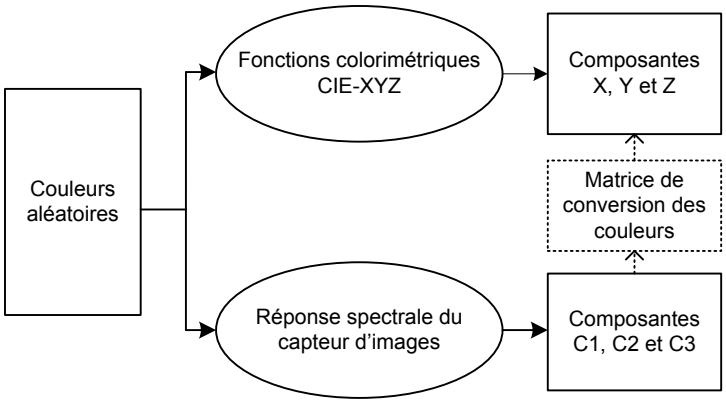


Figure 3-9 : Calcul des composantes XYZ et C1C2C3.

La dernière étape est la représentation graphique des résultats. Les premiers types de graphiques qu'il est possible de tracer est les graphiques des nuages de points des composantes XYZ et C1C2C3. Chaque graphique affiche le nuage des points dans leur espace de couleur respectif (XYZ ou capteur) et permet de faire une comparaison qualitative de leurs distributions. Le nuage de points des composantes XYZ forme une boule sphérique. En présence d'une bonne correspondance en couleurs entre les deux espaces colorimétriques, le nuage des points des composantes C1C2C3 du capteur devront former une boule semblable.

Pour tracer le deuxième type de graphiques, il faut parcourir les points du nuage des composantes C1C2C3 du capteur point par point. Pour chaque point, il faut considérer les points voisins situés dans une sphère de diamètre donné (typiquement unitaire). Pour la sphère considérée, on cherche la sphère correspondante dans le nuage des composantes XYZ et on calcule son rayon. On trace par la suite, un histogramme des fréquences des rayons des sphères XYZ. Un capteur idéal produira un histogramme avec des sphères de rayon d'environ 1.

3.3.2 Application au nouveau capteur CMOS

Le graphique des nuages de points des composantes XYZ et C1C2C3 est montré à la Figure 3-10 ci-dessous. Le nombre de couleurs aléatoires utilisées est de 20000. On a utilisé Matlab pour générer et tracer les composantes XYZ et C1C2C3 des couleurs aléatoires sur 8 bits. Le spectre visible de 390nm à 730 nm avait été divisé en 35 raies.On y remarque que les points C1C2C3 dans l'espace colorimétrique du capteur forment une sorte de bâton au lieu de former une boule comme c'est le cas pour le nuage des points des composantes XYZ. La position oblique du bâton indique que le capteur réduit les couleurs en des intensités lumineuses presque achromatiques.

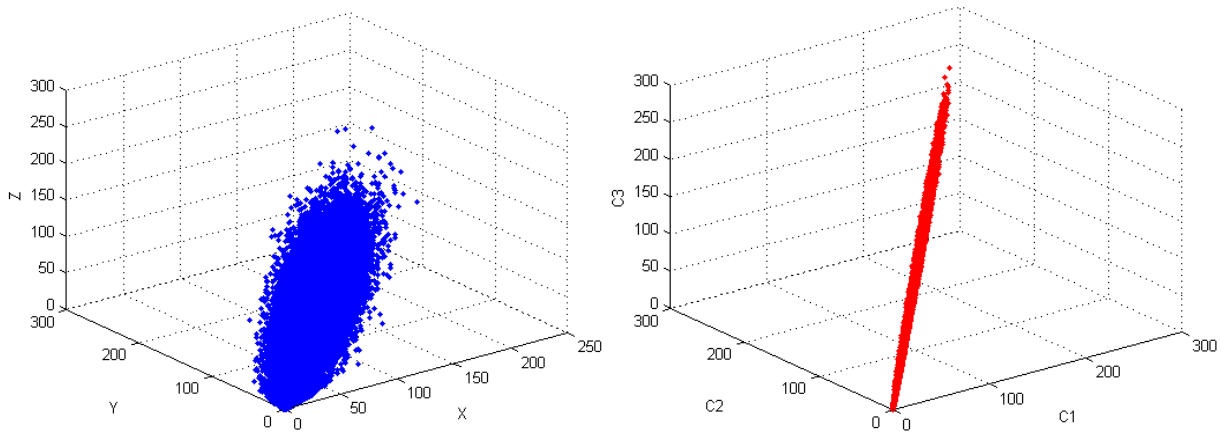


Figure 3-10 : Distribution des couleurs :

(a) composantes XYZ; (b) composantes C1C2C3.

Le deuxième graphique qu'on a tracé est l'histogramme des rayons des sphères composée des composantes XYZ. La Figure 3-11 montre l'histogramme obtenu. On voit clairement que la grande majorité des sphères ont un rayon large comparé aux rayons initiaux des sphères C1C2C3 qui étaient égaux à un.

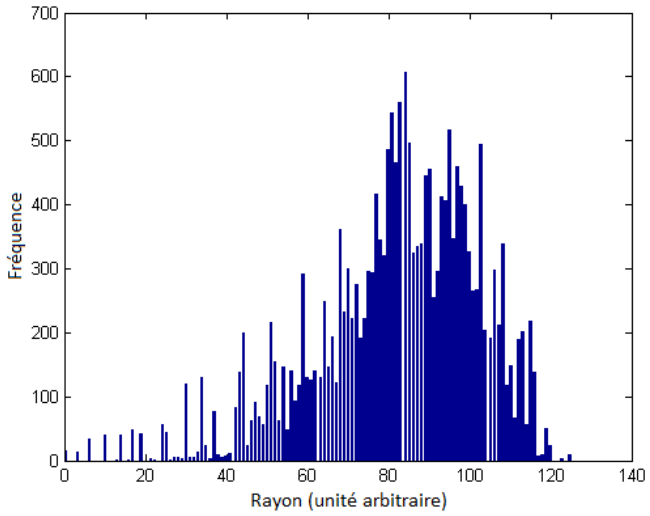


Figure 3-11 : Histogramme des rayons des sphères dans l'espace XYZ.

Dans le but de localiser les couleurs dans l'espace de couleur dépendant du capteur qui forment les sphères de rayon supérieur à zéro et inférieur à 10 dans l'espace XYZ, on a tracé le graphe de la Figure 3-12 ci-dessous. Dans celui-ci, on remarque que ces points sont dispersés le long du bâton qui forme les couleurs dans l'espace de couleur dépendant du capteur. On en conclut que le capteur n'a aucune région de couleur qui est bien représentée.

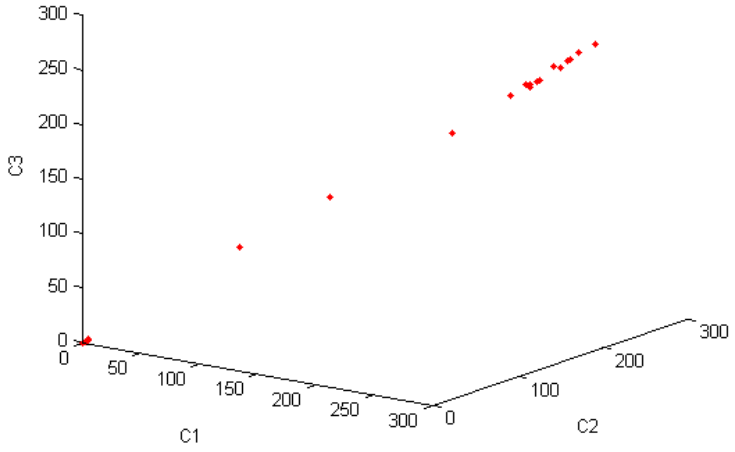


Figure 3-12 : Points dans l'espace du capteur correspondants à des sphères de rayons entre 1 et 10 dans l'espace XYZ.

3.4 Prototype de démonstration pour le nouveau capteur

3.4.1 Schéma global

Le prototype de démonstration a été développé pour le nouveau capteur dans le but de faciliter la démonstration de son fonctionnement. Le prototype est basé sur la carte de prototypage FPGA DE2-70 de Terasic jumelée à un écran LCD couleur tactile TRDB_LTM de la même compagnie. La Figure 3-13 ci-dessous montre le schéma global du prototype de démonstration.

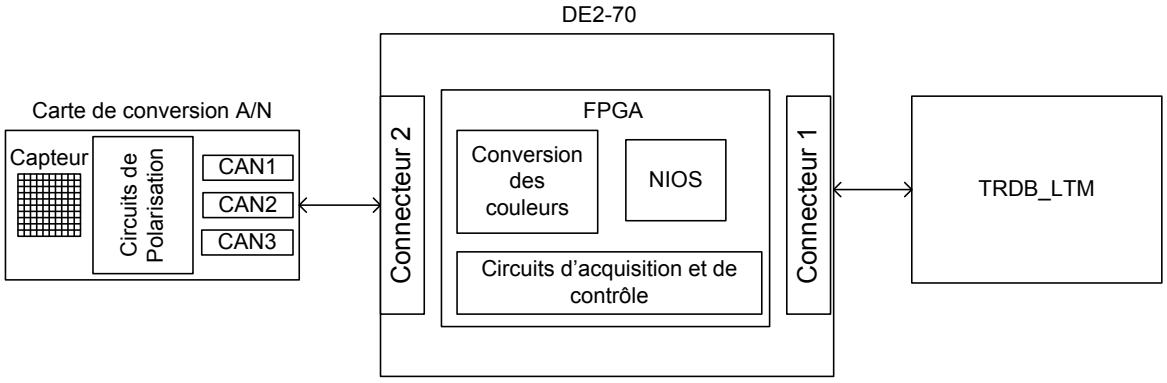


Figure 3-13 : Schéma global du prototype de démonstration.

La carte de conversion A/N permet de polariser correctement le capteur d'image à l'aide de plusieurs tensions de polarisation et de convertir les signaux analogiques des trois collecteurs du capteur en signaux numériques à l'aide de trois convertisseurs A/N. La carte DE2-70 est utilisée pour implémenter sur son FPGA les circuits numériques de conversion des couleurs, de contrôle et d'acquisition. L'écran couleur tactile est utilisé comme interface utilisateur et offre des menus permettant à l'utilisateur de contrôler le capteur CMOS et d'afficher les images acquises.

La carte de prototypage DE2-70 est construite autour d'un FPGA Cyclone II 2C70 d'Altera. Les composantes présentes sur la carte, en plus du FPGA, sont :

- USB Blaster pour la programmation et le contrôle.
- Mémoire SSRAM de 2 Méga octets.
- Mémoires SDRAM de 32 Méga octets.
- Mémoire Flash de 8 méga octets.
- 1 interrupteur.

- Oscillateur 50 MHz.
- 7 afficheurs 7 segments.
- 2 connecteurs d'extension 40 broches.

La Figure 3-14 ci-dessous montre une photo du prototype de démonstration.

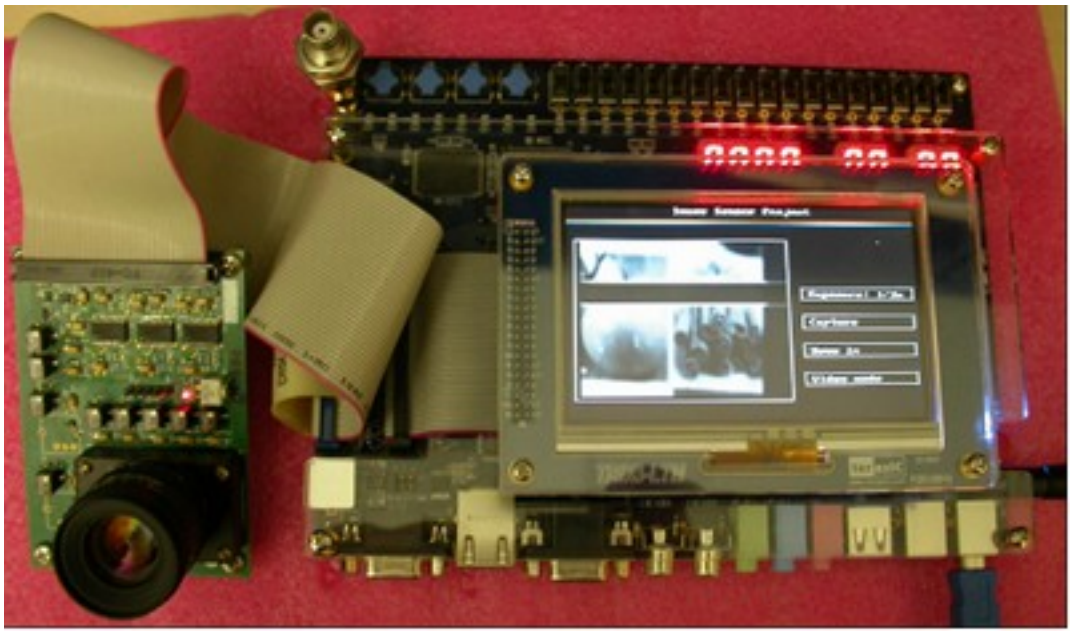


Figure 3-14 : Photo du prototype de démonstration.

3.4.2 Carte de conversion A/N

a) Cahier de charge

Le cahier de charge pour la conception de la carte de conversion A/N est le suivant :

- La carte doit générer cinq tensions de polarisation VB0 à VB4 ajustables entre 0 et 3.3V.
- La carte doit convertir trois signaux vidéo analogiques en signaux numériques sur 10 bits. À cause du manque de broches, limitées à 40 sur le connecteur de la carte DE2-70 auquel la carte de conversion A/N est connectée, les bus de sorties numériques des trois convertisseurs doivent être fusionnés en un seul bus partagé.
- La carte doit générer deux tensions de polarisation haute tension ajustables entre 0 et -100V.

- La carte doit recevoir directement sur sa surface la puce nue du capteur. La surface sur laquelle la puce sera collée doit être mise à la masse pour minimiser les bruits. La puce nue sera reliée aux traces métalliques de la carte à l'aide de fils métalliques. Cette opération sera réalisée à l'aide d'une machine de 'wire bonding'.

La Figure 3-15 ci-dessous montre le schéma de principe de la carte de conversion A/N.

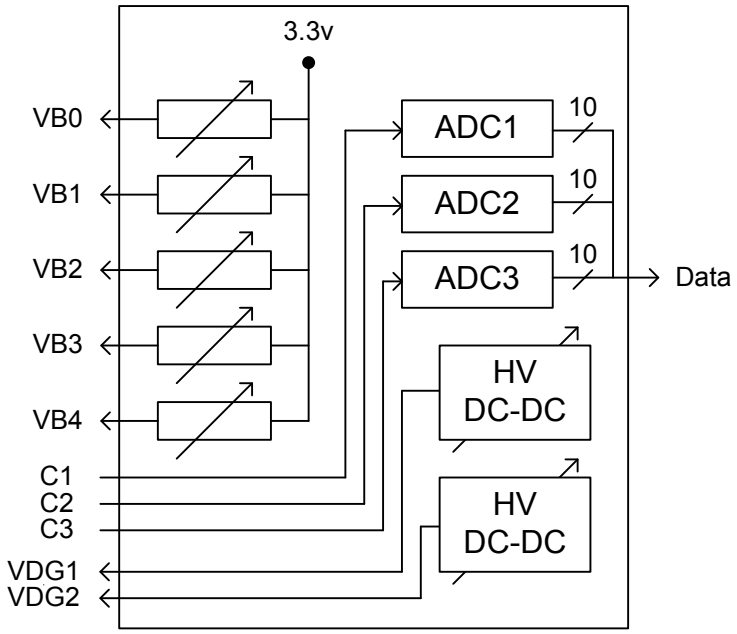


Figure 3-15 : Schéma de principe de la carte de conversion A/N.

b) Conception

Les annexes 4 et 5 donnent les schémas électriques de la carte de conversion A/N. Les cinq tensions de polarisation VB0 à VB4 sont générées à l'aide des cinq potentiomètres multi-tours R7 à R11. Ces potentiomètres sont des potentiomètres de précision.

Les conversions A/N sont réalisées à l'aide de trois circuits identiques basés sur le circuit intégré AD9200 d'Analog Devices. Il s'agit d'un convertisseur analogique numérique CMOS 10 bits capable de fonctionner jusqu'à une vitesse de 20 Méga échantillons par secondes. Ce convertisseur offre un signal pour mettre en haute impédance sa sortie numérique. Ceci est indispensable pour fusionner les sorties des trois ADCs en un seul bus accessible à tour de rôle.

Ce convertisseur A/N a été choisi pour ces caractéristiques convenables pour notre application dont on site :

- la conversion A/N se fait sur 10 bits;
- une faible consommation qui ne dépasse pas 80mW;
- la présence de la broche ‘THREE-STATE’ qui permet de mettre en haute impédance le bus numérique de sortie;
- la gamme dynamique de l’entrée qui est de 2V et qui correspond à la gamme dynamique des signaux vidéo de notre capteur d’images;
- les limites haut et bas pour le signal d’entrée sont ajustables à l’aide de tensions de référence externes.

La Figure 3-16 montre la configuration dans laquelle les trois convertisseurs sont utilisés.

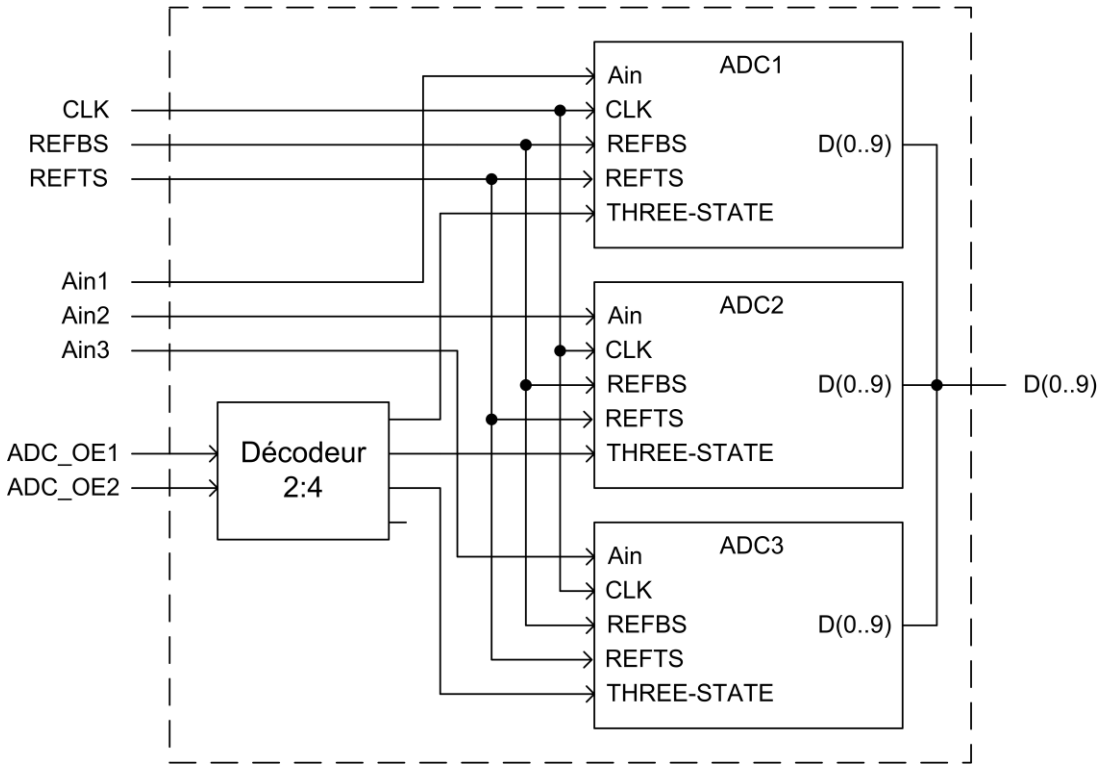


Figure 3-16 : Configuration des trois convertisseurs A/N.

Les bus de sorties numériques des trois convertisseurs sont fusionnés en un seul bus partagé. Les deux signaux ADC_OE1 et ADC_OE2, à travers le décodeur 2:4, permettent de sélectionner un seul ADC qui aura accès au bus de sortie partagé. Les bus de sortie des deux autres ADC seront en haute impédance. La raison d’être du décodeur 2:4 est de garantir matériellement qu’un seul ADC est activé à la fois et éviter ainsi qu’une erreur de programmation n’active deux circuits

ADC en même temps. La Figure 3-17 ci-dessous montre le chronogramme d'accès au bus de données partagé par les trois convertisseurs A/N.

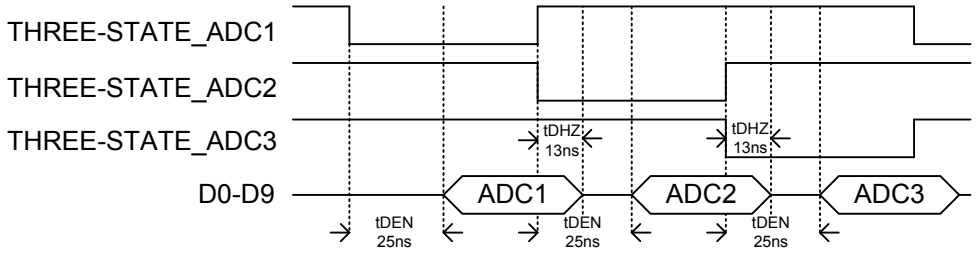


Figure 3-17 : Chronogramme d'accès au bus de données partagé.

Les deux tensions de références REFBS et REFTS qui permettent de fixer l'intervalle des tensions à l'entrée de chaque circuit ADC sont générées à l'aide du circuit réalisé autour du circuit intégré MAX6160 de la compagnie Maxim. Alimenté en 3.3V, ce circuit intégré offre une tension de référence stable entre 1.23V et 3.1V. Cette tension est utilisée pour fixer la tension REFTS des convertisseurs A/N. Les tensions REFBS de ces derniers sont obtenues à l'aide d'un potentiomètre ajustable qui prend une portion de la tension à la sortie du circuit MAX6160.

Pour obtenir les deux hautes tensions ajustables VDG1 et VDG2, le convertisseur DC-DC haute tension '0.1XS5-N0.1' de la compagnie Ultravolt est utilisé. Ce convertisseur offre l'avantage d'être miniature, à peine des dimensions de 11mmX11mmX11mm. Il est capable de fournir une tension de référence entre 0 et -100V ajustable à l'aide d'une tension de contrôle entre 0 et 2.5V. Cette tension est obtenue à l'aide d'un potentiomètre de précision.

c) Circuit imprimé

Le routage du circuit imprimé de la carte d'interface du capteur d'images et de conversion A/N est donné en annexe 6. Il est réalisé en quatre couches. L'une des deux couches internes est dédiée au plan de masse et la deuxième est dédiée aux tensions d'alimentation 3.3v et 5v. Les signaux sont routés sur les deux couches externes.

3.4.3 Circuit numérique de conversion des couleurs

Le circuit de conversion des couleurs est un circuit numérique simple qui fait la conversion des couleurs en multipliant les couleurs d'entrée [c1,c2,c3] par la matrice 3x3 de conversion des couleurs. Le code VHDL de ce circuit est donné en annexe 7. Le circuit est réalisé en architecture à pipeline comme le présente la Figure 3-18 ci-dessous qui montre le circuit de calcul de la

composante rouge. Les circuits de calcul des composantes vert et bleu sont identiques à celui de la composante rouge. La latence du pipeline est de quatre coups d'horloge.

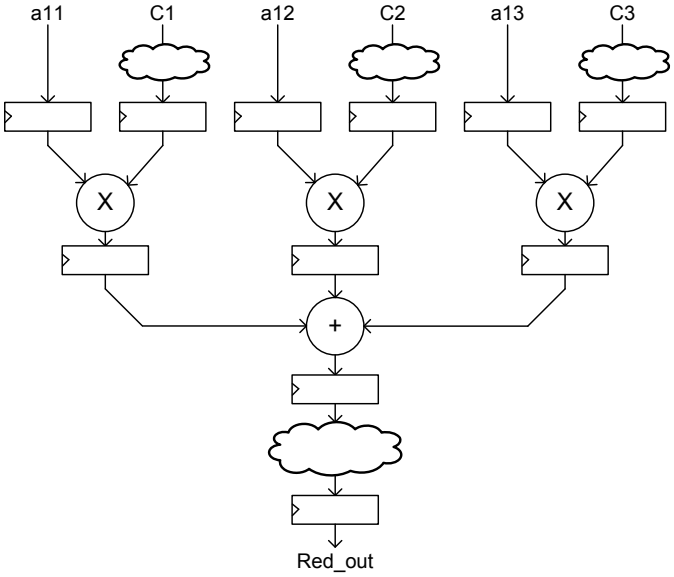


Figure 3-18 : Architecture à pipeline du circuit de calcul de la composante rouge.

Le circuit de conversion des couleurs réalise le calcul matriciel en virgule fixe. Le circuit est complètement paramétrable. Les paramètres 'mat_coef_width', 'fixed_point' et 'color_width' permettent de paramétrer respectivement le nombre de bits des coefficients de la matrice de conversion, la position du point fixe des coefficients et le nombre de bits sur lequel sont codés les couleurs d'entrée.

Pour notre application, le paramètre color_width doit être fixé à 10 puisque le nombre de bits pour C1, C2 et C3 est égal à 10. Le nombre de bits à allouer avant la virgule pour les coefficients de la matrice de conversion est de 2 bits puisqu'il s'agit de coefficients compris entre -1.0 et +1.0. Pour trouver le nombre de bits à allouer au point fixe, il faut étudier son effet sur l'erreur maximale. Cette erreur est calculable à l'aide de la formule suivante :

$$\begin{aligned} Erreur\_\max &= \pm\left[1023x2^{-(b+1)} + 1023x2^{-(b+1)} + 1023x2^{-(b+1)}\right] \\ &= \pm\left[3069x2^{-(b+1)}\right] \end{aligned} \qquad (3\text{-}1)$$

où b est le nombre de bits alloués au point fixe.

La Figure 3-19 ci-dessous montre l'erreur maximale à la sortie du circuit de conversion des couleurs en fonction du nombre de bits alloués au point fixe. L'erreur maximale décroit exponentiellement lorsque ce nombre de bits augmente.

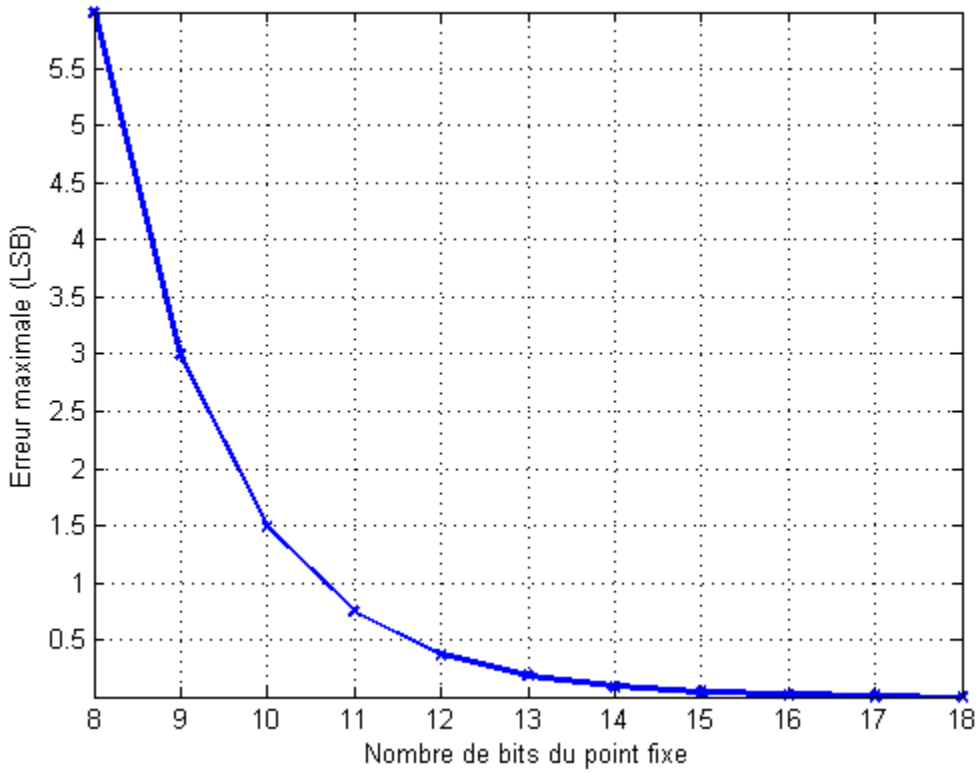


Figure 3-19 : Évolution de l'erreur maximale en fonction du nombre de bits alloués au point fixe.

Pour avoir une erreur maximale inférieure ou égale à 0.5, il faut allouer au point fixe un nombre de bits supérieur ou égal à 12.

Le circuit de conversion des couleurs a été validé par simulation. Le code VHDL du testbench utilisé est listé en annexe 8. Ce testbench génère aléatoirement 100 couleurs et les applique au circuit sous test. Ensuite, il calcul les valeurs attendues pour les sorties RGB en précision double. Enfin, il compare les valeurs calculées par le circuit sous test aux valeurs attendues et génère un message d'erreur chaque fois qu'il y a une déviation de plus de 0.5 entre les deux.

3.4.4 Circuits de contrôle, d'acquisition et d'affichage

Les circuits numériques de contrôle, d'acquisition et d'affichage sont réalisés dans le FPGA. Le 'top' du design FPGA est montré à la Figure 3-20.

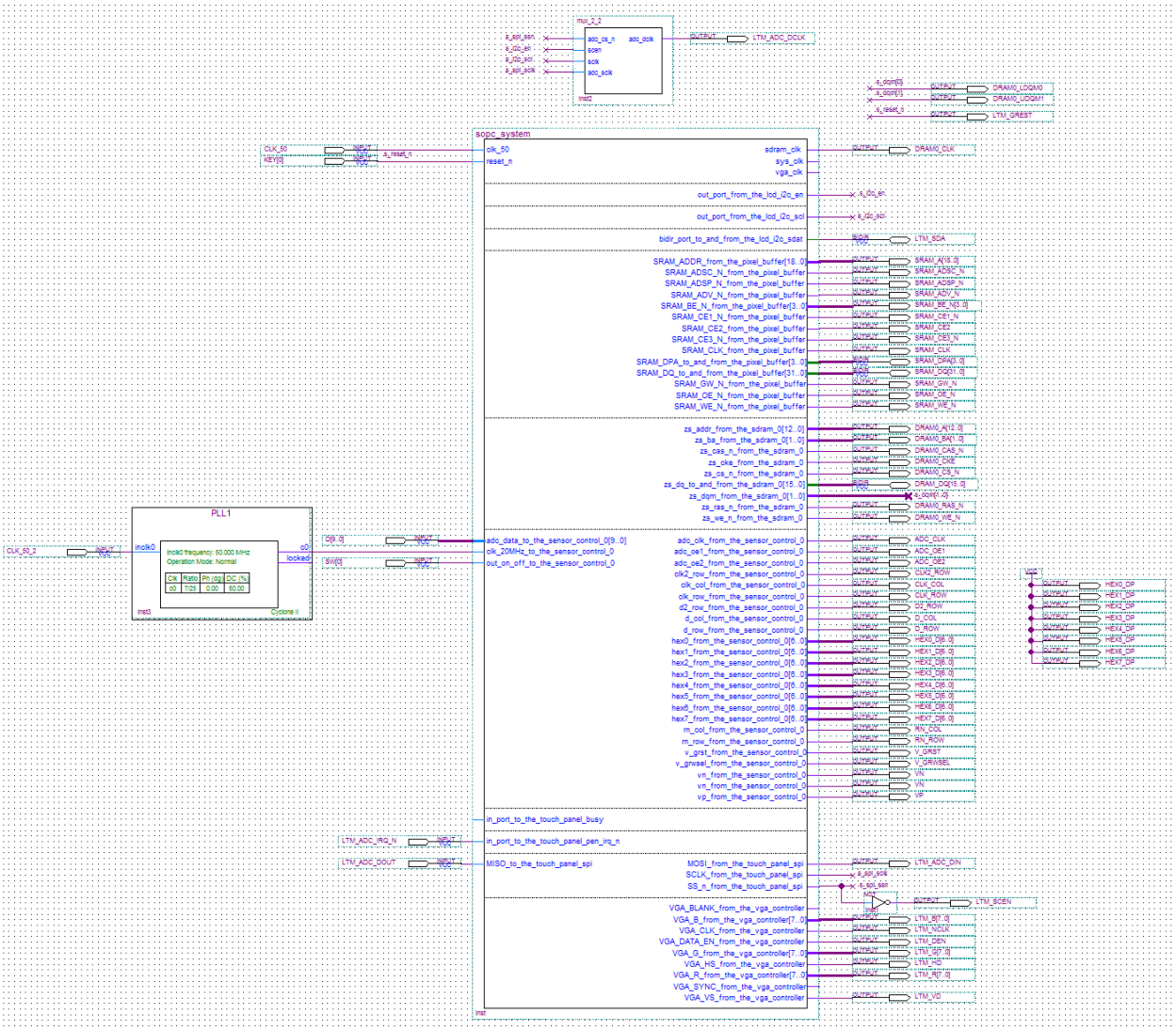


Figure 3-20 : Aperçu du 'top' du design FPGA.

Il est constitué d'une PLL qui génère une horloge de 20MHz, d'un multiplexeur et du système SOPC qui encapsule le reste des circuits autour du NIOS. Les circuits du système SOPC sont montrés à la Figure 3-21 ci-dessous.

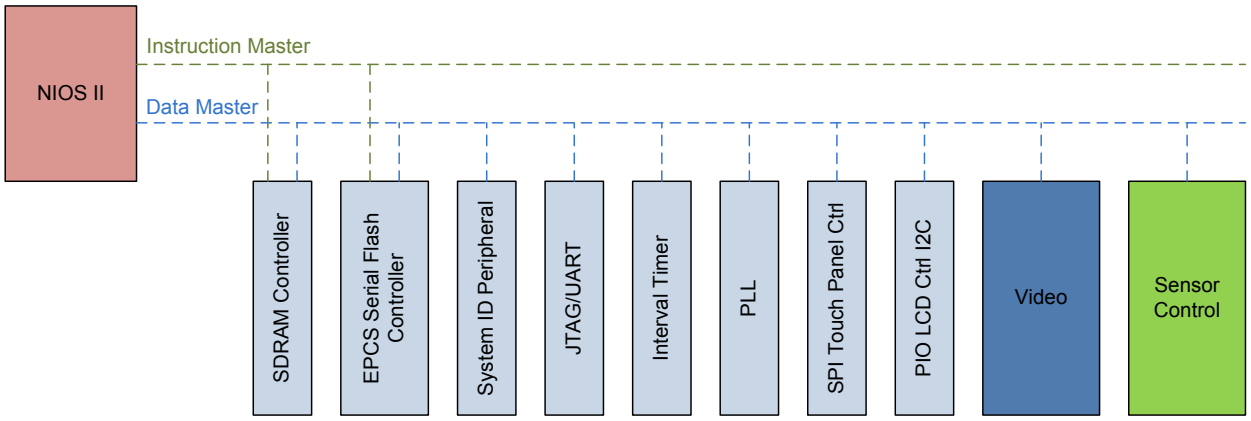


Figure 3-21 : Contenu du système SOPC.

La PLL permet de générer 3 horloges : une horloge système de 50 MHz, une horloge de 50 MHz dont la phase est de -3ns par rapport à l'horloge système requise par la SDRAM et une horloge vidéo de 25MHZ requise par l'écran LCD tactile. Le périphérique SPI maitre permet de communiquer avec le port série SPI de l'écran tactile pour l'interroger de l'état de la surface tactile. Plusieurs périphérique PIO (Parallel Input Output) sont utilisé pour simuler un maitre I2C (Inter Integrated Circuit) par logiciel. Ce dernier est utilisé pour configurer l'écran tactile.

Le composant vidéo est constitué de plusieurs cœurs de propriété intellectuelle (IP cores) du programme universitaire d'Altera [28]. La Figure 3-22 ci-dessous montre leur configuration pour réaliser le périphérique vidéo. Sur le diagramme, on voit deux chemins vidéo : le premier vient de la mémoire SRAM constituant la mémoire vidéo et le deuxième vient du tampon de caractères. Les deux flux sont fusionnés en un seul flux vidéo par le mixeur Alpha (Alpha Blender). Le premier flux composant le flux d'arrière plan pour le mixeur est généré par le contrôleur DMA (Pixel Buffer DMA Controller) qui lit la mémoire vidéo en utilisant le contrôleur SRAM. Le flux est par la suite rééchantillonné de 24 bits à 30 bits pour être compatible avec le format demandé par le mixeur Alpha qui ne fonctionne qu'en 30 bits. Ensuite la résolution du flux vidéo est convertie par le convertisseur de résolution (Scaler) de 400x240 à 800x480, la résolution de l'écran LCD. Le tampon de caractère qui génère le flux de premier plan pour le mixeur Alpha est un module qui fait le rendu de caractères ASCI vers une représentation graphique pour l'affichage. Un programme s'exécutant dans le NIOS peut envoyer des codes ASCI de caractères à travers l'interface Avalon. Ceux-ci sont sauvegardés dans une mémoire intégrée (on-chip). Un contrôleur DMA lit les caractères de la mémoire et les envoie au module de rendu graphique qui

s'occupe de faire le rendu de leur représentation graphique. Le tampon de caractère supporte un seul mode vidéo dans lequel les caractères sont dessinés en blanc sur un fond transparent. Sa résolution est de 50 caractères par 30 lignes.

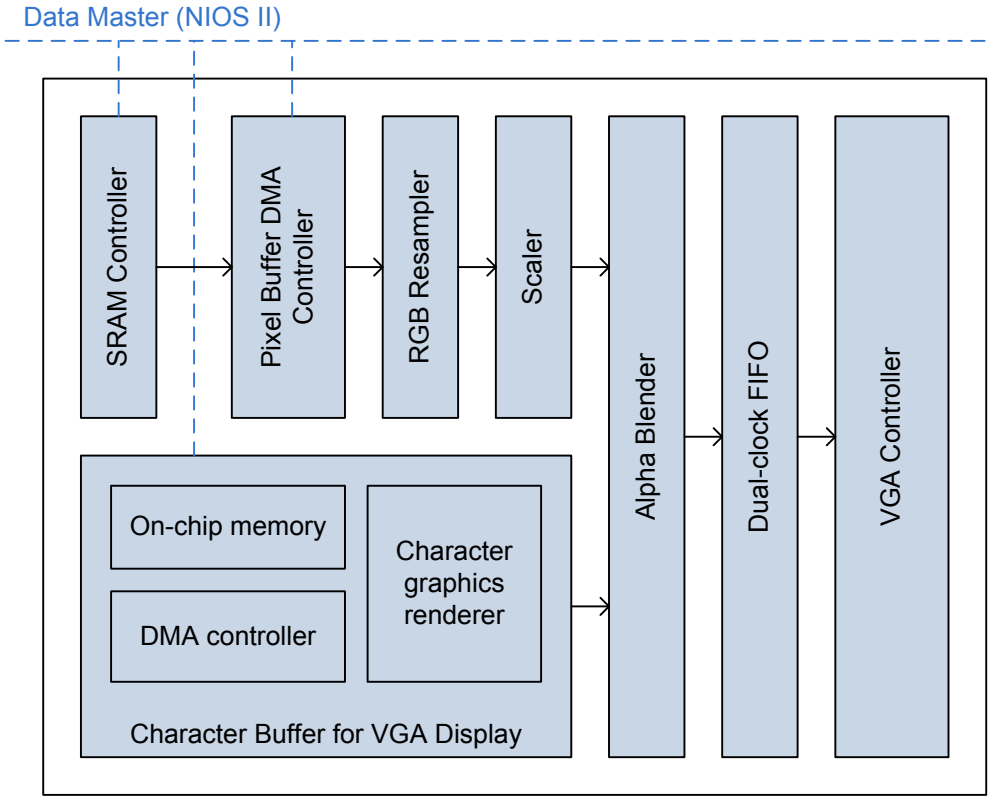


Figure 3-22 : Cœurs de propriétés intellectuelles vidéo.

Le mixeur Alpha fusionne le flux vidéo d'arrière plan avec celui de premier plan. Le résultat est sauvegardé à la FIFO double horloges. L'horloge d'écriture à la FIFO est de fréquence 50 MHz. Le contrôleur VGA lit de la même FIFO avec une horloge de fréquence 25 MHz. C'est ce contrôleur qui génère les signaux de synchronisation requis par l'écran LCD.

Le code VHDL du composant SOPC 'sensor_control' est listé en annexe 9. Son architecture interne est montrée à la Figure 3-23.

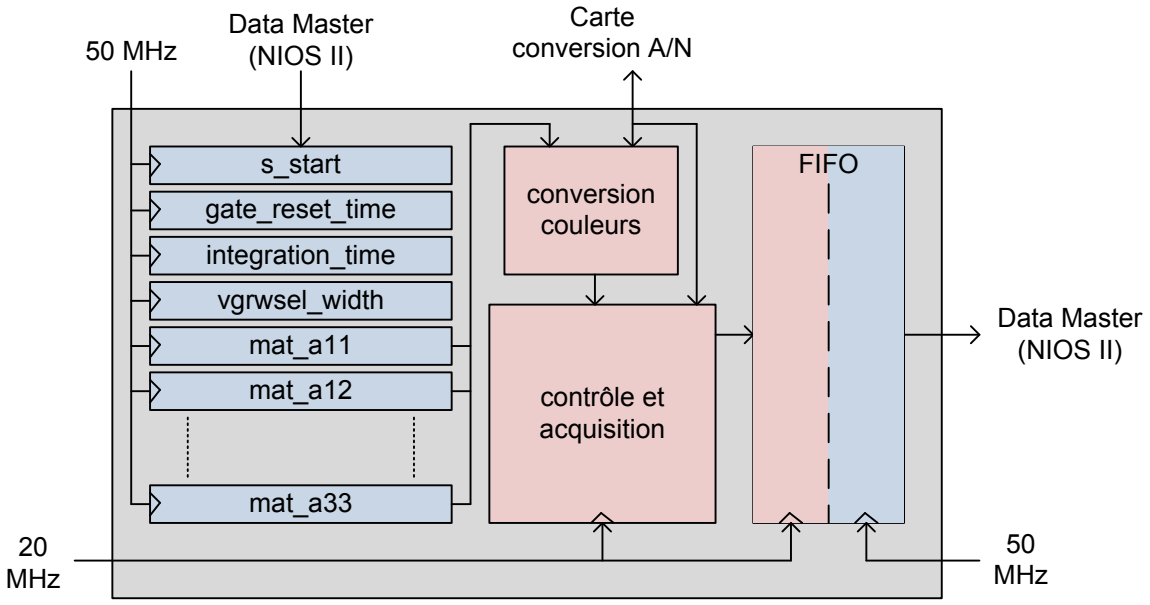


Figure 3-23 : Architecture interne du composant ‘sensor_control’.

Ce composant a 13 registres internes accessibles en écriture seulement au NIOS via le bus Avalon. Chaque registre occupe une adresse distincte. Le tableau ci-dessous donne l’adresse et la description de chacun des 13 registres.

Tableau 3.1 : Adresses des 13 registres internes accessibles en écriture seulement.

Registre	**Adresse**	**Description**
s_start	0b0000	Signal début pour démarrer l’acquisition d’une image.
gate_reset_time	0b0001	Valeur du temps de remise à zéro des pixels.
integration_time	0b0010	Valeur du temps d’intégration.
vgrwsel_width	0b0011	Longueur du pulse du signal vgrwsel.
mat_a11	0b0100	Coefficients de la matrice de conversion des couleurs.
mat_a12	0b0101	
mat_a13	0b0110	
mat_a21	0b0111	
mat_a22	0b1000	
mat_a23	0b1001	
mat_a31	0b1010	
mat_a32	0b1011	
mat_a33	0b1100	

Le circuit de contrôle permet de générer les signaux de synchronisation pour le capteur d’image. Le circuit d’acquisition permet de faire l’acquisition d’une image et de la sauvegarder dans la

FIFO à double horloge. Pour démarrer une acquisition, le programme du NIOS doit écrire un ‘1’ au registre ‘s_start’. L’image acquise est par la suite récupérable à partir de la FIFO. Celle-ci permet de partager les données par les deux domaines d’horloge. En effet, les circuits de contrôle et d’acquisition fonctionnent à 20MHz et le NIOS fonctionne à 50MHz.

La validation du module ‘sensor_control’ a été réalisée en simulation. Le testbench de la simulation est listé en annexe 10. Le résultat de la simulation pour une image de 2 lignes par 4 colonnes est donné sur le diagramme en annexe 11.

Le code source du programme du NIOS est listé aux annexes 12 à 17. Le NIOS est utilisé pour gérer l’interface utilisateur montré à la Figure 3-24 ci-dessous.

Figure 3-24 : Aperçu de l’interface utilisateur.

Cette interface est constituée d’une zone pour afficher l’image capturée et de quatre boutons de commande : ‘Exposure’, ‘Capture’, ‘Zoom’ et ‘Mode’. Le bouton ‘Exposure’ permet de changer le temps d’intégration en temps réel. Pour augmenter le temps d’intégration, il suffit de toucher le côté droit du bouton. Pour diminuer le temps d’intégration, il suffit de toucher le côté gauche du même bouton. Le bouton ‘Capture’ permet de lancer la capture d’une image. À la fin de cette opération, l’image capturée est affichée dans la zone image. Le bouton ‘Zoom’ permet d’appliquer un zoom 2x sur l’image affichée. Le dernier bouton ‘Mode’ permet de déterminer le mode de capture soit le mode image ou vidéo. Le mode vidéo permet de faire une capture d’images en continu.

3.4.5 Statistiques d'utilisation du FPGA

La Figure 3-25 ci-dessous montre le rapport d'utilisation des ressources FPGA par le projet au complet comme rapporté par Quartus.

Quartus II Version	9.0 Build 235 06/17/2009 SP 2 SJ Full Version
Revision Name	image_sensor
Top-level Entity Name	image_sensor
Family	Cyclone II
Device	EP2C70F896C6
Timing Models	Final
Met timing requirements	No
Total logic elements	9,102 / 68,416 (13 %)
Total combinational functions	8,026 / 68,416 (12 %)
Dedicated logic registers	5,002 / 68,416 (7 %)
Total registers	5055
Total pins	240 / 622 (39 %)
Total virtual pins	0
Total memory bits	657,536 / 1,152,000 (57 %)
Embedded Multiplier 9-bit elements	34 / 300 (11 %)
Total PLLs	2 / 4 (50 %)

Figure 3-25 : Statistiques d'utilisation du FPGA.

Le projet utilise 13% des LE (Logic Elements), 57% des bits mémoire et 11% des multiplicateurs 9 bits. On remarque qu'il reste amplement de place pour compléter la caméra numérique émulée par d'autres circuits de traitement d'images.

CONCLUSION

Le premier chapitre et le deuxième chapitre de ce mémoire étaient une revue de littérature sur le phénomène de la couleur et sur les capteurs d'images numériques couleurs. La réalisation de ces revues de littérature m'a permis de mieux comprendre la couleur et son utilisation dans les capteurs d'images étant donné que s'agissait d'un nouveau domaine pour moi.

Au chapitre 3, intitulé conversion des couleurs et prototype de démonstration, j'ai présenté ma contribution dans l'effort de développement du nouveau capteur. La conversion des couleurs est une étape importante dans la chaine de traitement d'image pour produire une image sur un media d'affichage à partir d'une image brute. Cette opération consiste à convertir les couleurs de l'espace colorimétrique dépendant du capteur d'image à un espace colorimétrique indépendant du capteur. Plusieurs espaces colorimétriques peuvent être la cible de cette opération. Parmi eux, on a choisi l'espace colorimétrique sRGB puisqu'il s'agit d'un standard largement adopté par l'industrie de l'image et de la vidéo. La conversion des couleurs est réalisée à l'aide d'une matrice 3x3 de conversion. À cause de la nature du capteur d'image utilisé, il fallait donner une attention particulière à la méthode utilisée pour calculer la matrice de conversion. Celle-ci aura de grands coefficients dans la diagonale comme rapporté par la compagnie Foveon pour son capteur. Or, ceux-ci sont responsables de la dégradation des performances en bruit. Une méthode a été identifiée pour le calcul de la matrice de conversion des couleurs en tenant compte du bruit dans l'image. Dans cette méthode, un compromis peut être réalisé entre l'erreur en couleur et le niveau de bruit dans l'image finale. Cette méthode a été implémentée sous forme d'un GUI Matlab qui prend comme entrée une image de la mire GretagMacbeth prise par le capteur d'image. Cette image fournit au GUI 24 échantillons de couleur dont les valeurs sRGB théoriques sont connues. Le GUI permet de calculer la matrice de conversion des couleurs en faisant un compromis entre l'erreur en couleur et le bruit dans l'image finale selon la valeur d'un facteur de compromis ajustable selon le besoin. Les résultats obtenus avec le nouveau capteur au niveau de la fidélité en couleur étaient décevants. Les écarts en couleur dans l'espace uniforme CIE-Lab entre les couleurs de l'image convertie et les échantillons de la mire originale sont grands. Pour comprendre mieux le problème, les réponses spectrales des trois collecteurs du capteur ont été tracées. Ces réponses montraient des courbes qui n'ont pas des pics distincts. D'ailleurs, le calcul

de l'indice de métamérisme de sensibilité qui est une métrique pour évaluer la réponse spectrale d'un capteur a donné une valeur petite de 13,37. Étant donné qu'il s'agit d'un chiffre ne donnant pas beaucoup d'informations et dans le but de trouver une méthode qui donnera plus d'informations, nous avons développé une nouvelle méthode pour évaluer la reproduction des couleurs basée sur la génération d'un grand nombre de couleurs aléatoirement, le calcul des composante XYZ dans l'espace colorimétrique CIE-XYZ et les composantes C1, C2 et C3 dans l'espace colorimétrique du capteur et à la fin, la représentation graphique des résultats. L'application de cette méthode au nouveau capteur a indiqué que le capteur réduisait les couleurs en des intensités lumineuses presque monochromatiques et qu'il n'a aucune région de couleur qui est bien représentée.

Le prototype de caméra a été développé en se basant sur une carte de prototypage FPGA jumelée à un écran LCD couleur tactile. Une carte d'interface du capteur et de conversion A/N a été conçue. Cette carte reçoit le capteur d'image et le système optique. Elle génère les différentes tensions de polarisation nécessaires au bon fonctionnement du capteur et converti les signaux analogiques de ce dernier en signaux numériques qui sont acheminés aux circuits numériques dans le FPGA de la carte de développement. Des circuits numériques de conversion des couleurs, de contrôle, d'acquisition et d'affichage sont réalisés dans celui-ci.

Dans ce travail, ma contribution a été la réalisation d'une caméra numérique de démonstration fonctionnelle basée sur le nouveau capteur en utilisant une carte de prototypage FPGA jumelée à un écran tactile couleur. L'utilisation de celle-ci avec des cores IP pour la vidéo a facilité le développement même si en réalité les cores IP utilisés présentaient plusieurs anomalies de fonctionnement qu'il fallait contourner. L'utilisation du NIOS pour gérer le GUI utilisateur a permis d'alléger les cycles de développement et de déverminage. La réalisation de la caméra numérique de démonstration a demandé le développement de circuits numériques pour la conversion des couleurs, le contrôle et l'acquisition. Ma deuxième contribution est notre méthode graphique pour l'évaluation de la réponse spectrale d'un capteur d'images.

Le prototype de démonstration a très bien fonctionné. Le prototype E du nouveau capteur d'images sur lequel j'ai travaillé durant ma maîtrise n'était pas fidèle dans la capture des couleurs. Un nouveau prototype F a été conçu et fabriqué par l'équipe de recherche. Ce prototype doit être caractérisé et testé pour déterminer ces performances. La nouvelle méthode d'évaluation

de la reproduction des couleurs proposée dans ce travail doit être essayée sur plusieurs capteurs afin de déterminer son utilité et ses limites.

BIBLIOGRAPHIE

[29-33]

[1] R. Lukac, "Single-sensor imaging : methods and applications for digital cameras," *Image processing series*, 2009. Disponible: http://www.crcnetbase.com/isbn/978-1-4200-5452-1.

[2] Foveon, "Direct image sensors." [En ligne]. Disponible: http://www.foveon.com/article.php?a=67. [Consulté le 6 août 2009].

[3] B. Balland, *Optique géométrique : imagerie et instruments*, 1e éd., Lausanne, [Suisse]: Presses polytechniques et universitaires romandes, 2007.

[4] National Institute of Standards and Technology, "CODATA Internationally recommended values of the Fundamental Physical Constants," *Fundamental Physical Constants from NIST*, pp., 2008. [En ligne]. Disponible: http://physics.nist.gov/cuu/Constants/. [Consulté le 19 juillet 2009].

[5] ISO, "Photométrie - Le système CIE de photométrie physique," ISO 23539, 2008.

[6] Commission Internationale de l'Éclairage, "Vocabulaire international de l'éclairage," Publications de la CIE, 17.4, 1987. StandardsStore, http://www.standardsstore.ca.

[7] D. Metz, "Comprendre la couleur et ses profils," *Comprendre la couleur et maîtriser les profils ICC*. [En ligne]. Disponible: http://www.profil-couleur.com. [Consulté le 25 mars 2009].

[8] A. Trémeau, C. Fernandez-Maloigne, et P. Bonton, *Image numérique couleur : de l'acquisition au traitement*, Paris: Dunod, 2004.

[9] D. Steen, "Colorimétrie: Éléments théoriques," in Mesures - Analyses, vol. R6440, Ed.^Eds., ed. Paris: Techniques de l'ingénieur, 2004, pp. 1-14.

[10] J.-L. Trepanier, "Capteur d'images CMOS à pixels numériques et à gamme dynamique élevée," M.Sc.A., École Polytechnique de Montréal, Qc, Canada, 2003. [En ligne]. Disponible: Proquest Dissertations and theses, http://proquest.umi.com/. [Consulté le 5 avril 2011].

[11] P. Burasa, "Caractérisation d'une matrice de pixels conçue pour capteurs d'images couleurs sans filtre optique," M.Sc.A., École Polytechnique de Montréal, Qc, Canada, 2008. [En ligne]. Disponible: Proquest Dissertations and theses, http://proquest.umi.com/. [Consulté le 5 avril 2011].

[12] K. I. Sensors, "Color correction for image sensors," pp., 2008. [En ligne]. Disponible: http://www.kodak.com/global/plugins/acrobat/en/business/ISS/supportdocs/ColorCorrectionforImageSensors.pdf. [Consulté le 7 juin 2011].

[13] R. B. Merrill, "Color separation in an active pixel cell imaging array using a triple-well structure," Brevet USA US005965875A, 1999. [En ligne]. Disponible: http://www.google.com/patents. [Consulté le 6 août 2009].

[14] R. F. Lyon et P. M. Hubel, "Eyeing the Camera: Into the Next Century," in Final Program and Proceedings of the 10th IS and T/SID Color Imaging Conference: Color Science, Systems and Applications, vol., Ed.^Eds., ed.: Society for Imaging Science and Technology, 2002, pp. 349-355.

[15] Y. Audet, "Photodétecteur pour imagerie spectrale sans filtre Déclaration d'invention DIV-426, 2010. Disponible.

[16] H. E. J. Neugebauer, "Quality factor for filters whose spectral transmittances are different from color mixture curves and its application to color photography," *Journal of the Optical Society of America*, vol. 46, no. 10, pp. 821-824, 1956.

[17] P. L. Vora et H. J. Trussell, "Measure of goodness of a set of color-scanning filters," *Journal of the Optical Society of America A (Optics and Image Science)*, vol. 10, no. 7, pp. 1499-1508, 1993.

[18] N. Shimano, "Colorimetric Evaluation of Color Image Acquisition Systems I – A Proposal of a New Colorimetric Quality Factor," *Journal of the Institute of Image Electronics Engineers of Japan*, vol. 29, no. 5, pp. 506-516, 2000.

[19] J. Tajima, "New quality measures for a set of color sensors - weighted quality factor, spectral characteristic restorability index and color reproducibility index," in Proceedings of the Color Imaging Conference: Color Science, Systems, and Applications, vol., Ed.^Eds., ed. Scottsdale, AZ, USA: Soc Imaging Sci Technol, Springfield, VA, United States, 1997, pp. 25-28.

[20] P.-C. Hung, "Comparison of camera quality indexes," in 8th IS and T/SID Color Imaging Conference: Color Science, Systems and Applications, vol., Ed.^Eds., ed. Scottsdale, AZ, United states, 2000, pp. 167-171.

[21] G. Sharma et H. J. Trussell, "Figures of Merit for Color Scanners," *IEEE Transactions on Image Processing*, vol. 6, no. 7, pp. 990-1001, 1997.

[22] S. Quan, N. Ohta, R. S. Berns, X. Jiang, et N. Katoh, "Unified measure of goodness and optimal design of spectral sensitivity functions," *Journal of Imaging Science and Technology*, vol. 46, no. 6, pp. 485-497, 2002.

[23] S. Quan, "Evaluation and optimal design of spectral sensitivities for digital color imaging," Rochester Institute of Technology, Rochester, New York, USA, Rochester, New York, USA, 2002. [En ligne]. Disponible: http://www.cis.rit.edu/mcsl/research/PDFs/Quan.pdf. [Consulté le 26 avril 2011].

[24] ISO, "Colour characterisation of digital still cameras (DSCs) - Part 1: Stimuli, metrology and test procedures," ISO 17321-1, 2006.

[25] Y.-P. Tan et T. Acharya, "A Method for Color Correction with Noise Consideration," in *Color lmaging : Device-Independent Color, Color Hardcopy and Graphic Arts V*, *San Jose, USA*, vol. 3963, 2000, pp. 329-337.

[26] P. M. Hubel, "Foveon technology and the changing landscape of digital cameras," in STOC'05: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, vol. 1, Ed.^Eds., ed. Baltimore, MD, United states: Society for Imaging Science and Technology, 2005, pp. 314-317.

[27] BabelColor, "ColorChecker," pp., 2003. [En ligne]. Disponible: http://www.babelcolor.com/main_level/ColorChecker.htm. [Consulté le 7 juillet 2011].

[28] Aletra, "Altera University Program - IP Cores." Disponible: http://www.altera.com/education/univ/materials/comp_org/ip-cores/unv-ip-cores.html. [Consulté le 3 mai 2010].

[29] W.-C. Kao, S.-H. Wang, C.-C. Kao, C.-W. Huang, et S.-Y. Lin, "Color Reproduction for Digital Imaging Systems," in *IEEE International Symposium on Circuits and Systems*, 2006, pp. 4599-4602. [En ligne]. Disponible: IEEE Xplore, http://ieeexplore.ieee.org/. [Consulté le 11 mai 2010].

[30] P. Green et L. MacDonald, *Colour engineering : achieving device independent colour*, Chichester, England: Wiley, 2002.

[31] H. R. Kang, *Computational color technology*, Bellingham, Wash.: SPIE Press, 2006.

[32] H. J. Trussell et M. J. Vrhel, *Fundamentals of digital imaging*, Cambridge, U.K.: Cambridge University Press, 2008.

[33] W. Sanial, *Traité d'éclairage*, 2e éd., Toulouse, France: Cépadués-Editions, 2007.

ANNEXE 1 – Code Matlab de 'calculCCM.m'

```
function calculCCM(handles)
% Cette fonction calcul la matrice de correction des couleurs.
%

global im_rgb;

% Les valeurs sRGB de la mire de Macbeth
load P.mat;

Q = zeros(3,24);

% Première ligne de la mire Macbeth
Q(:,1) = getColor([15 6 16 16]);
Q(:,2) = getColor([54 6 16 16]);
Q(:,3) = getColor([90 6 16 16]);
Q(:,4) = getColor([124 6 16 16]);
Q(:,5) = getColor([160 6 16 16]);
Q(:,6) = getColor([198 6 16 16]);

% Deuxième ligne de la mire Macbeth
Q(:,7) = getColor([15 33 16 16]);
Q(:,8) = getColor([54 33 16 16]);
Q(:,9) = getColor([90 33 16 16]);
Q(:,10) = getColor([124 33 16 16]);
Q(:,11) = getColor([160 33 16 16]);
Q(:,12) = getColor([198 33 16 16]);

% Troisième ligne de la mire Macbeth
Q(:,13) = getColor([15 62 16 16]);
Q(:,14) = getColor([54 62 16 16]);
Q(:,15) = getColor([90 62 16 16]);
Q(:,16) = getColor([124 62 16 16]);
Q(:,17) = getColor([160 62 16 16]);
Q(:,18) = getColor([198 62 16 16]);

% Quatrième ligne de la mire Macbeth
Q(:,19) = getColor([15 90 16 16]);
Q(:,20) = getColor([54 90 16 16]);
Q(:,21) = getColor([90 90 16 16]);
Q(:,22) = getColor([124 90 16 16]);
Q(:,23) = getColor([160 90 16 16]);
Q(:,24) = getColor([198 90 16 16]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calcul de la matrice de covariance du bruit
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hsize = get(handles.hSizeMenu, 'value');
v = get(handles.sigmaText,'string');
sigma = str2num(v);
H = fspecial('gaussian',[hsize hsize],sigma);
im_filtred = imfilter(im_rgb,H,'replicate');
```

```
% figure(2), clf,
% imshow(im_filtred, 'InitialMagnification', 100, 'Border', 'loose');
% title('Image filtrée');
% axis ij;
% axis image;
% impixelinfo;

im_noise = im_rgb - im_filtred;
% figure(3), clf,
% imshow(im_noise, 'InitialMagnification', 100, 'Border', 'loose');
% title('Bruit de l''image');
% axis ij;
% axis image;
% impixelinfo;

im_noise_r = im_noise(:,:,1);
im_noise_g = im_noise(:,:,2);
im_noise_b = im_noise(:,:,3);
covariance = cov(double([im_noise_r(:),im_noise_g(:),im_noise_b(:)]))

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Calcul de la matrice de conversion des couleurs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
v = get(handles.factorText,'string');
w = str2num(v);
Mat = P*Q'*inv((Q*Q')+(w*covariance));

% Normalisation de la matrice de conversion des couleurs
Mat1 = Mat(1,:);
ind = find(Mat1>0);
sum1 = sum(Mat1(ind));
Mat2 = Mat(2,:);
ind = find(Mat2>0);
sum2 = sum(Mat2(ind));
Mat3 = Mat(3,:);
ind = find(Mat3>0);
sum3 = sum(Mat3(ind));
Mat = Mat./max([sum1,sum2,sum3]);

set(handles.a11 ,'String', num2str(Mat(1,1)));
set(handles.a12 ,'String', num2str(Mat(1,2)));
set(handles.a13 ,'String', num2str(Mat(1,3)));
set(handles.a21 ,'String', num2str(Mat(2,1)));
set(handles.a22 ,'String', num2str(Mat(2,2)));
set(handles.a23 ,'String', num2str(Mat(2,3)));
set(handles.a31 ,'String', num2str(Mat(3,1)));
set(handles.a32 ,'String', num2str(Mat(3,2)));
set(handles.a33 ,'String', num2str(Mat(3,3)));

set(handles.sH1 ,'String', num2str(Mat(1,1)+Mat(1,2)+Mat(1,3)));
set(handles.sH2 ,'String', num2str(Mat(2,1)+Mat(2,2)+Mat(2,3)));
set(handles.sH3 ,'String', num2str(Mat(3,1)+Mat(3,2)+Mat(3,3)));
```

ANNEXE 2 – Code Matlab de 'applyCCM.m'

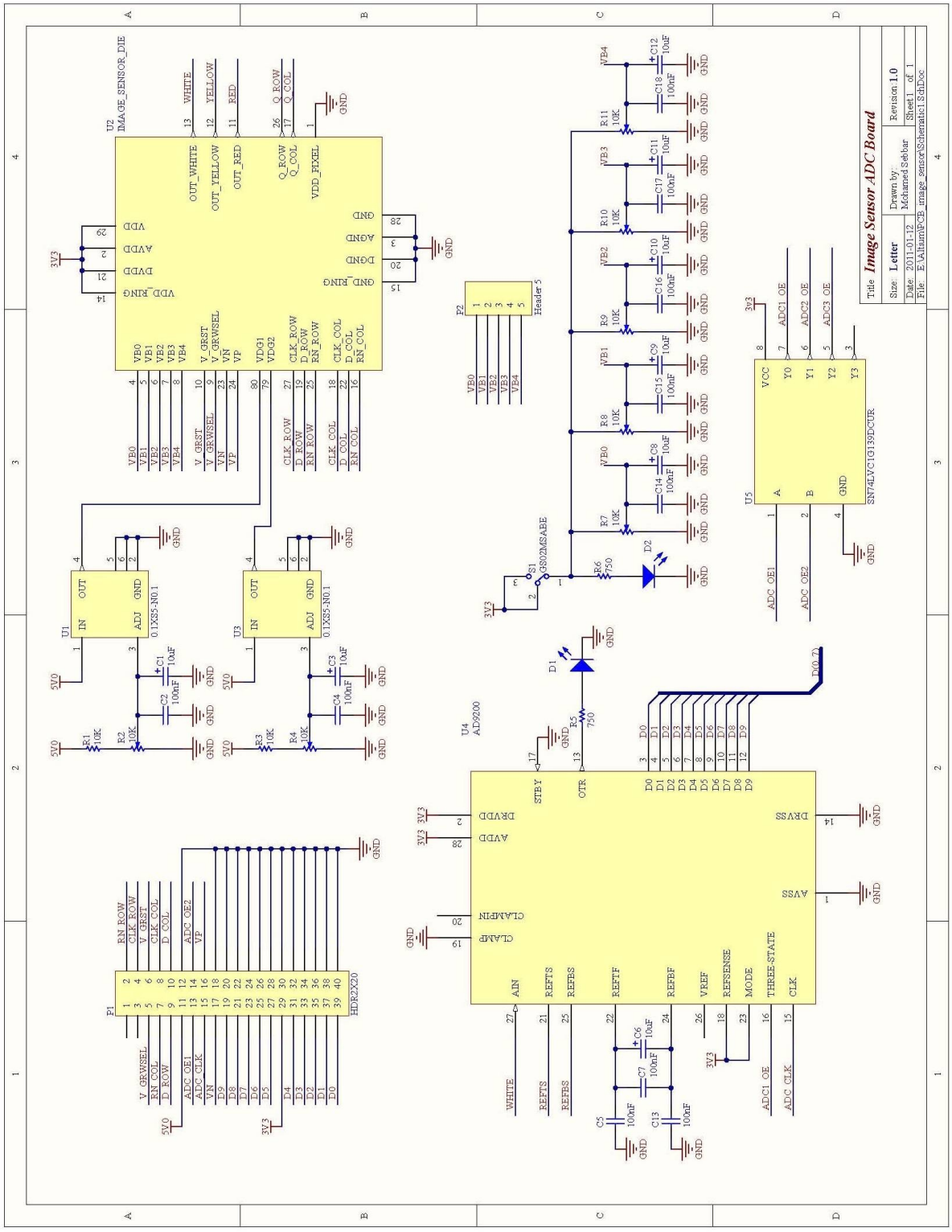
```
function applyCCM(handles)
% Cette fonction applique la matrice de conversion des couleurs
% à l'image ouverte et affiche l'image résultante.
%

global im_rgb;

% Récupérer du GUI la matrice de conversion
a11 = str2double(get(handles.a11, 'String'));
a12 = str2double(get(handles.a12, 'String'));
a13 = str2double(get(handles.a13, 'String'));
a21 = str2double(get(handles.a21, 'String'));
a22 = str2double(get(handles.a22, 'String'));
a23 = str2double(get(handles.a23, 'String'));
a31 = str2double(get(handles.a31, 'String'));
a32 = str2double(get(handles.a32, 'String'));
a33 = str2double(get(handles.a33, 'String'));

Mat = [a11 a12 a13
       a21 a22 a23
       a31 a32 a33];

imd_r = double(im_rgb(:,:,1))/255;
imd_g = double(im_rgb(:,:,2))/255;
imd_b = double(im_rgb(:,:,3))/255;

[M, N] = size(imd_r);
s = M*N;

imd_rgb = [reshape(imd_r,1,s); reshape(imd_g,1,s); reshape(imd_b,1,s)];

% Appliquer la matrice de conversion
imt_rgb = Mat * imd_rgb;

imt_r = imt_rgb(1,:);
imt_r = reshape(imt_r, M, N);
imt_g = imt_rgb(2,:);
imt_g = reshape(imt_g, M, N);
imt_b = imt_rgb(3,:);
imt_b = reshape(imt_b, M, N);

imt_r = uint8(round(imt_r*255));
imt_g = uint8(round(imt_g*255));
imt_b = uint8(round(imt_b*255));

ims_rgb = cat(3,imt_r,imt_g,imt_b);

% Affichage de l'image convertie
figure(4), clf,
imshow(ims_rgb, 'InitialMagnification', 100, 'Border', 'loose');
title('Image convertie');
```

```
axis ij;
axis image;
impixelinfo;
```

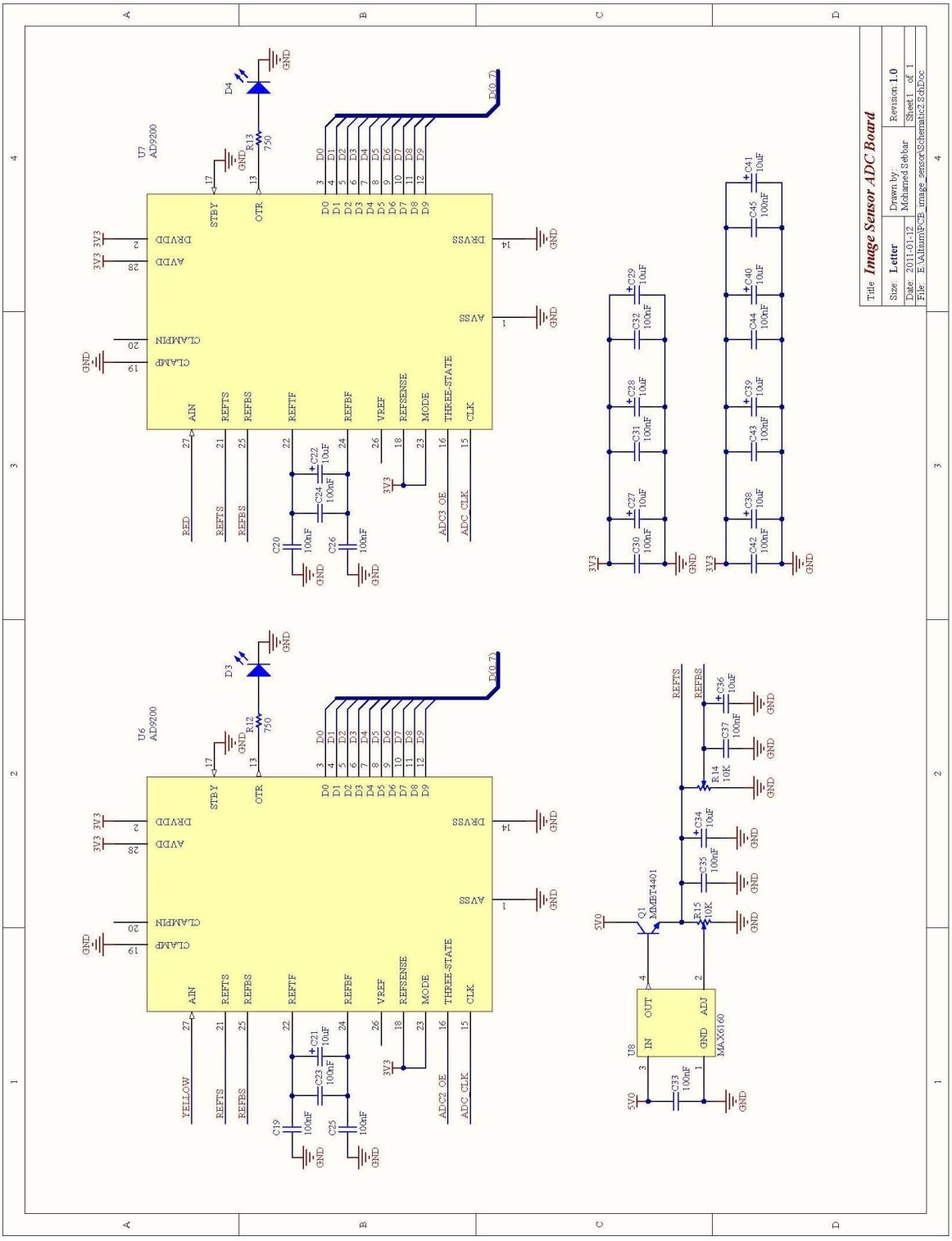
ANNEXE 3 – Code Matlab de ‘getColor.m’

```
function color = getColor(rect)
% Cette fonction permet de placer interactivement un rectangle de
% sélection d'une région dans l'image. Il calcul par la suite les
% valeurs RGB moyennes dans la région sélectionnée.
%

global im_rgb;

% Placer le rectangle de sélection
figure(1);
title('Placez intéractivement le rectangle. Double-cliquez pour finir.');
H = imrect(gca, rect);
setFixedAspectRatioMode(H, 'true');
position = wait(H);

% Position du rectangle de sélection
xmin = uint8(position(1));
ymin = uint8(position(2));
width = uint8(position(3));
height = uint8(position(4));

% Calcul des valeurs RGB moyennes
im_cropped = im_rgb(ymin:ymin+width,xmin:xmin+height,:);
r = im_cropped(:,:,1);
g = im_cropped(:,:,2);
b = im_cropped(:,:,3);

r_mean = mean(r(:));
g_mean = mean(g(:));
b_mean = mean(b(:));

color = [r_mean, g_mean, b_mean]';
title('');
```

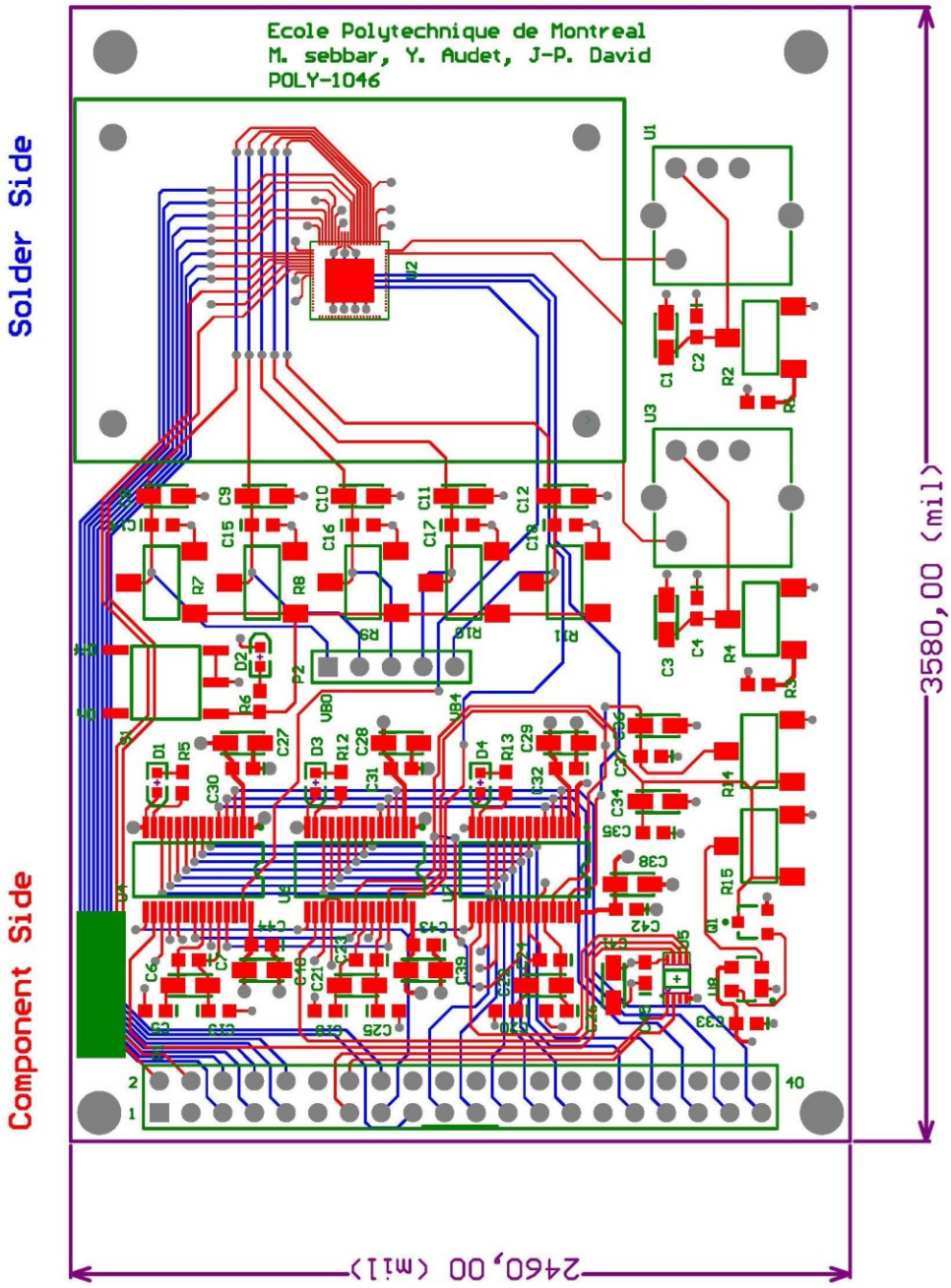
ANNEXE 4 – ‘Schematic1.SchDoc’ du circuit de conversion A/N



ANNEXE 5 – ‘Schematic2.SchDoc’ du circuit de conversion A/N



ANNEXE 6 – Circuit imprimé du circuit de conversion A/N



BOM

Commentaire	Déscription	Désignateur	Quantité
10uF	Solid Tantalum Chip Capacitor, Standard T491 Series - Industrial Grade	C1, C3, C6, C8, C9, C10, C11, C12, C21, C22, C27, C28, C29, C34, C36, C38, C39, C40, C41	19
100nF	Ceramic Chip Capacitor - Standard	C2, C4, C5, C7, C13, C14, C15, C16, C17, C18, C19, C20, C23, C24, C25, C26, C30, C31, C32, C33, C35, C37, C42, C43, C44, C45	26
LED_HSMS-C190	Surface Mount Chip LED, Red	D1, D2, D3, D4	4
MMBT4401	NPN General Purpose Amplifier	Q1	1
10K	Thick Film Chip Resistor, 1 Ohm to 10M Ohm Range, 5% Tolerance, 0603 Size, 0.1 W	R1, R3	2
10K	Trimming Potentiometer	R2, R4, R7, R8, R9, R10, R11, R14, R15	9
750	Thick Film Chip Resistor, 1 Ohm to 10M Ohm Range, 5% Tolerance, 0603 Size, 0.1 W	R5, R6, R12, R13	4
GSD2MSABE	Switch, Subminiature, SMT	S1	1
AD9200	Complete 10-Bit, 20 MSPS, 80 mW CMOS A/D Converter	U4, U6, U7	3
MAX6160	Low-Cost, Low-Dropout, 3-Terminal Voltage Reference	U8	
SN74LVC1G139DCUR	2-To-4 Line Decoder	U5	1

ANNEXE 7 – Code VHDL de ‘color_conversion.vhd’

```
--------------------------------------------------------------------------------
-- Titre         : Color conversion
-- Projet        : Capteur d'images
--------------------------------------------------------------------------------
-- Fichier       : color_conversion.vhd
-- Auteur        : Mohamed Sebbar
--------------------------------------------------------------------------------
-- Description   : Ce circuit permet de faire la conversion des couleurs en
--                 appliquant la matrice de conversion 3x3 aux couleurs RGB
--                 à l'entrée.
--------------------------------------------------------------------------------
-- Historique de modification :
-- 2011/06/14 : création
--------------------------------------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;
use ieee.numeric_std.all;


entity color_conversion is
  generic (
    mat_coef_width : integer := 9;
    fixed_point    : integer := 7;
    color_width    : integer := 10
  );

  port(
    reset_n  : in std_logic;
    clk      : in std_logic;

    -- Composantes RGB d'entrées
    red_in   : in std_logic_vector(color_width-1 downto 0);
    green_in : in std_logic_vector(color_width-1 downto 0);
    blue_in  : in std_logic_vector(color_width-1 downto 0);

    -- Coefficients de la matrice de conversion 3x3
    mat_a11  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a12  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a13  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a21  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a22  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a23  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a31  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a32  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a33  : in std_logic_vector(mat_coef_width-1 downto 0);

    -- Composantes RGB converties
    red_out  : out std_logic_vector(color_width-1 downto 0);
    green_out: out std_logic_vector(color_width-1 downto 0);
    blue_out : out std_logic_vector(color_width-1 downto 0)
  );

end color_conversion;

architecture behav of color_conversion is

signal s_red_in   : std_logic_vector(color_width downto 0);
signal s_green_in : std_logic_vector(color_width downto 0);
```

```
signal s_blue_in  : std_logic_vector(color_width downto 0);
signal s_mul_11 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_12 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_13 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_21 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_22 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_23 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_31 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_32 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_mul_33 : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_sum_r : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_sum_g : std_logic_vector(mat_coef_width+color_width downto 0);
signal s_sum_b : std_logic_vector(mat_coef_width+color_width downto 0);

begin

-------------------------------------------------------------------------------
-- Ce process inverse les couleurs RGB et les concatène avec un bit signe
-- à zéro (nombres positifs).
-------------------------------------------------------------------------------
couleur_inverser : process(clk, reset_n)
begin
  if (reset_n = '0') then
    s_red_in   <= (others => '0');
    s_green_in <= (others => '0');
    s_blue_in  <= (others => '0');
  elsif (clk'event and clk='1') then
--    s_red_in   <= std_logic_vector(to_unsigned(1023,color_width)) - red_in;
--    s_green_in <= std_logic_vector(to_unsigned(1023,color_width)) - green_in;
--    s_blue_in  <= std_logic_vector(to_unsigned(1023,color_width)) - blue_in;
    s_red_in   <= '0'&(not red_in);
    s_green_in <= '0'&(not green_in);
    s_blue_in  <= '0'&(not blue_in);
  end if;
end process couleur_inverser;

-------------------------------------------------------------------------------
-- Ce process fait le calcul matriciel RGB(3x1)*M(3x3) en pipeline.
-------------------------------------------------------------------------------
calcul_matriciel : process(clk, reset_n)
begin
  if (reset_n = '0') then
    s_mul_11 <= (others => '0');
    s_mul_12 <= (others => '0');
    s_mul_13 <= (others => '0');
    s_mul_21 <= (others => '0');
    s_mul_22 <= (others => '0');
    s_mul_23 <= (others => '0');
    s_mul_31 <= (others => '0');
    s_mul_32 <= (others => '0');
    s_mul_33 <= (others => '0');
    s_sum_r  <= (others => '0');
    s_sum_g  <= (others => '0');
    s_sum_b  <= (others => '0');
  elsif (clk'event and clk='1') then
    s_mul_11 <= mat_a11 * s_red_in;
    s_mul_12 <= mat_a12 * s_green_in;
    s_mul_13 <= mat_a13 * s_blue_in;
    s_sum_r  <= s_mul_11 + s_mul_12 + s_mul_13;
    s_mul_21 <= mat_a21 * s_red_in;
    s_mul_22 <= mat_a22 * s_green_in;
    s_mul_23 <= mat_a23 * s_blue_in;
    s_sum_g  <= s_mul_21 + s_mul_22 + s_mul_23;
```

```
    s_mul_31 <= mat_a31 * s_red_in;
    s_mul_32 <= mat_a32 * s_green_in;
    s_mul_33 <= mat_a33 * s_blue_in;
    s_sum_b  <= s_mul_31 + s_mul_32 + s_mul_33;
  end if;
end process calcul_matriciel;

-------------------------------------------------------------------------------
-- Ce process calcul la sortie R. Si le résultat du calcul matriciel est
-- négatif, la sortie R est nulle. Sinon, R est égal au résultat entier du
-- calcul matriciel.
-------------------------------------------------------------------------------
calcul_sortie_r : process(clk, reset_n)
begin
  if (reset_n = '0') then
    red_out  <= (others => '0');
  elsif (clk'event and clk='1') then
    if (s_sum_r(mat_coef_width+color_width) = '1') then
      red_out <= (others => '0');
--   elsif ( s_sum_r(mat_coef_width+color_width downto fixed_point) > std_logic_vector(
--                 to_unsigned(1023,(mat_coef_width-fixed_point)+color_width+1)) ) then
--      red_out <= (others => '1');
    else
      red_out <= s_sum_r(color_width+fixed_point-1 downto fixed_point) +
s_sum_r(fixed_point-1);
    end if;
  end if;
end process calcul_sortie_r;

-------------------------------------------------------------------------------
-- Ce process calcul la sortie G. Si le résultat du calcul matriciel est
-- négatif, la sortie G est nulle. Sinon, G est égal au résultat entier du
-- calcul matriciel.
-------------------------------------------------------------------------------
calcul_sortie_g : process(clk, reset_n)
begin
  if (reset_n = '0') then
    green_out<= (others => '0');
  elsif (clk'event and clk='1') then
    if (s_sum_g(mat_coef_width+color_width) = '1') then
      green_out <= (others => '0');
--   elsif ( s_sum_g(mat_coef_width+color_width downto fixed_point) > std_logic_vector(
--                 to_unsigned(1023,(mat_coef_width-fixed_point)+color_width+1)) ) then
--      green_out <= (others => '1');
    else
      green_out <= s_sum_g(color_width+fixed_point-1 downto fixed_point) +
s_sum_g(fixed_point-1);
    end if;
  end if;
end process calcul_sortie_g;

-------------------------------------------------------------------------------
-- Ce process calcul la sortie B. Si le résultat du calcul matriciel est
-- négatif, la sortie B est nulle. Sinon, B est égal au résultat entier du
-- calcul matriciel.
-------------------------------------------------------------------------------
calcul_sortie_b : process(clk, reset_n)
begin
  if (reset_n = '0') then
    blue_out <= (others => '0');
  elsif (clk'event and clk='1') then
    if (s_sum_b(mat_coef_width+color_width) = '1') then
      blue_out <= (others => '0');
```

```
--   elsif ( s_sum_b(mat_coef_width+color_width downto fixed_point) > std_logic_vector(
--                 to_unsigned(1023,(mat_coef_width-fixed_point)+color_width+1)) ) then
--      blue_out <= (others => '1');
    else
      blue_out <= s_sum_b(color_width+fixed_point-1 downto fixed_point) +
s_sum_b(fixed_point-1);
    end if;
  end if;
end process calcul_sortie_b;


end behav;
```

ANNEXE 8 – Code VHDL de ‘color_conversion_tb.vhd’

```
--------------------------------------------------------------------------------
-- Titre         : Color conversion Testbench
-- Projet        : Capteur d'images
--------------------------------------------------------------------------------
-- Fichier       : color_conversion_tb.vhd
-- Auteur        : Mohamed Sebbar
--------------------------------------------------------------------------------
-- Description   : Testbench pour le module color_conversion.
--------------------------------------------------------------------------------
-- Historique de modification :
-- 2011/06/14 : création
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_signed.all;
use ieee.math_real.all;
use std.textio.all;
use ieee.std_logic_textio.all;

-- Ce testbench test le module color_conversion.
entity color_conversion_tb is
end color_conversion_tb;

architecture arch of color_conversion_tb is

constant periode : time := 20 ns;
constant mat_coef_width : integer := 18;--14;
constant fixed_point    : integer := 16;--12;
constant color_width    : integer := 10;

signal clk     : std_logic := '1';
signal reset_n : std_logic := '1';
signal state   : std_logic := '0';
signal red_in   : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal green_in : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal blue_in  : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal mat_a11  : real := 0.25241;
signal mat_a12  : real := 0.5515;
signal mat_a13  : real := -0.076142;
signal mat_a21  : real := 0.60532;
signal mat_a22  : real := -0.059697;
signal mat_a23  : real := 0.20208;
signal mat_a31  : real := 0.47538;
signal mat_a32  : real := -0.45318;
signal mat_a33  : real := 0.52462;
signal fp_mat_a11 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a12 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a13 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a21 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a22 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a23 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a31 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a32 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal fp_mat_a33 : std_logic_vector(mat_coef_width-1 downto 0) := (others => '0');
signal red_out  : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal red_d1   : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal red_d2   : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal red_d3   : std_logic_vector(color_width-1 downto 0) := (others => '0');
```

```
signal calc_R   : real := 0.0;
signal green_out: std_logic_vector(color_width-1 downto 0) := (others => '0');
signal green_d1 : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal green_d2 : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal green_d3 : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal calc_G   : real := 0.0;
signal blue_out : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal blue_d1  : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal blue_d2  : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal blue_d3  : std_logic_vector(color_width-1 downto 0) := (others => '0');
signal calc_B   : real := 0.0;

-------------------------------------------------------------------------------
-- Déclaration du module à vérifier.
-------------------------------------------------------------------------------
component color_conversion
  generic (
    mat_coef_width : integer := 9;
    fixed_point     : integer := 7;
    color_width    : integer := 10
  );

  port(
    reset_n  : in std_logic;
    clk      : in std_logic;

    -- Composantes RGB d'entrées
    red_in   : in std_logic_vector(color_width-1 downto 0);
    green_in : in std_logic_vector(color_width-1 downto 0);
    blue_in  : in std_logic_vector(color_width-1 downto 0);

    -- Coefficients de la matrice de conversion 3x3
    mat_a11  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a12  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a13  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a21  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a22  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a23  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a31  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a32  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a33  : in std_logic_vector(mat_coef_width-1 downto 0);

    -- Composantes RGB converties
    red_out  : out std_logic_vector(color_width-1 downto 0);
    green_out: out std_logic_vector(color_width-1 downto 0);
    blue_out : out std_logic_vector(color_width-1 downto 0)
  );
end component color_conversion;

begin

-------------------------------------------------------------------------------
-- instanciation du module à vérifier.
-------------------------------------------------------------------------------
  UUT : color_conversion
  generic map(
    mat_coef_width => mat_coef_width,
    fixed_point    => fixed_point,
    color_width    => color_width
  )
  port map(
    reset_n  => reset_n,
    clk      => clk,
```

```
    red_in   => red_in,
    green_in => green_in,
    blue_in  => blue_in,
    mat_a11  => fp_mat_a11,
    mat_a12  => fp_mat_a12,
    mat_a13  => fp_mat_a13,
    mat_a21  => fp_mat_a21,
    mat_a22  => fp_mat_a22,
    mat_a23  => fp_mat_a23,
    mat_a31  => fp_mat_a31,
    mat_a32  => fp_mat_a32,
    mat_a33  => fp_mat_a33,
    red_out  => red_out,
    green_out=> green_out,
    blue_out => blue_out
  );

  -- Génération de l'horloge
  clk <= not clk after periode/2;

  -- Coefficients de la matrice de conversion en points fixes
  fp_mat_a11                                                              <=
std_logic_vector(to_signed(integer(mat_a11*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a12                                                              <=
std_logic_vector(to_signed(integer(mat_a12*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a13                                                              <=
std_logic_vector(to_signed(integer(mat_a13*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a21                                                              <=
std_logic_vector(to_signed(integer(mat_a21*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a22                                                              <=
std_logic_vector(to_signed(integer(mat_a22*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a23                                                              <=
std_logic_vector(to_signed(integer(mat_a23*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a31                                                              <=
std_logic_vector(to_signed(integer(mat_a31*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a32                                                              <=
std_logic_vector(to_signed(integer(mat_a32*(2.0**fixed_point)),mat_coef_width));
  fp_mat_a33                                                              <=
std_logic_vector(to_signed(integer(mat_a33*(2.0**fixed_point)),mat_coef_width));

  ------------------------------------------------------------------------
  -- Ce process génère 100 vecteurs de test aléatoires et les applique aux entrées
  -- du module à vérifier au front descendant de l'horloge.
  ------------------------------------------------------------------------
  process(clk)
    constant limit : positive := 104;
    variable tampon : line; -- pointeur vers objet de type string
    variable seed1 : positive := 1;
    variable seed2 : positive := 2;
    variable seed3 : positive := 2;
    variable aleatoire1 : real := 0.0;
    variable aleatoire2 : real := 0.0;
    variable aleatoire3 : real := 0.0;
    variable varR : integer := -1;
    variable varG : integer := -1;
    variable varB : integer := -1;
    variable compte : natural range 0 to limit := 0;

  begin
    if falling_edge(clk) then
      if(compte = limit) then
        report "Simulation terminée" severity failure;
      else
```

```
      uniform(seed1, seed2, aleatoire1);       -- 0.0 < aleatoire1 < 1.0
      aleatoire1 := floor(aleatoire1 * 1023.0);-- 0.0 <= aleatoire1 <= 1023.0
      varR := integer(aleatoire1);             -- 0 <= varR <= 1023

      uniform(seed2, seed3, aleatoire2);       -- 0.0 < aleatoire < 1.0
      aleatoire2 := floor(aleatoire2 * 1023.0);-- 0.0 <= aleatoire <= 1023.0
      varG := integer(aleatoire2);             -- 0 <= varG <= 1023

      uniform(seed1, seed3, aleatoire3);       -- 0.0 < aleatoire < 1.0
      aleatoire3 := floor(aleatoire3 * 1023.0);-- 0.0 <= aleatoire <= 1023.0
      varB := integer(aleatoire3);             -- 0 <= varB <= 1023

      -- Application des vecteurs de test aux entrées du module à vérifier.
      red_in <= std_logic_vector(to_unsigned(1023-varR,color_width));
      green_in <= std_logic_vector(to_unsigned(1023-varG,color_width));
      blue_in <= std_logic_vector(to_unsigned(1023-varB,color_width));

      write(tampon, now, unit => ns);
      write(tampon, " (R,G,B) = (" & integer'image(varR) & "," &
              integer'image(varG) & "," & integer'image(varB) & ")" );
      writeline(output, tampon); -- écriture à la console

      compte := compte + 1;
    end if;
  end if;
     end process;

-------------------------------------------------------------------------------
-- Ce process retarde la valeur red_in de trois coups d'horloge.
-------------------------------------------------------------------------------
process(clk)
begin
  if rising_edge(clk) then
    red_d1 <= std_logic_vector(to_unsigned(1023,color_width)) - red_in;
    red_d2 <= red_d1;
    red_d3 <= red_d2;
  end if;
end process;

-------------------------------------------------------------------------------
-- Ce process retarde la valeur green_in de trois coups d'horloge.
-------------------------------------------------------------------------------
process(clk)
begin
  if rising_edge(clk) then
    green_d1 <= std_logic_vector(to_unsigned(1023,color_width)) - green_in;
    green_d2 <= green_d1;
    green_d3 <= green_d2;
  end if;
end process;

-------------------------------------------------------------------------------
-- Ce process retarde la valeur blue_in de trois coups d'horloge.
-------------------------------------------------------------------------------
process(clk)
begin
  if rising_edge(clk) then
    blue_d1 <= std_logic_vector(to_unsigned(1023,color_width)) - blue_in;
    blue_d2 <= blue_d1;
    blue_d3 <= blue_d2;
  end if;
end process;
```

```
-------------------------------------------------------------------------------
-- Ce process calcul la valeur attendue de la composante R.
-------------------------------------------------------------------------------
process(clk)
  variable varR : real := 0.0;
begin
  if rising_edge(clk) then
    varR := real(to_integer(unsigned(red_d3))) * mat_a11+
            real(to_integer(unsigned(green_d3))) * mat_a12+
            real(to_integer(unsigned(blue_d3))) * mat_a13;
    if (varR < 0.0) then
      calc_R <= 0.0;
    else
      calc_R <= varR;
    end if;
  end if;
end process;

-------------------------------------------------------------------------------
-- Ce process calcul la valeur attendue de la composante G.
-------------------------------------------------------------------------------
process(clk)
  variable varG : real := 0.0;
begin
  if rising_edge(clk) then
    varG := real(to_integer(unsigned(red_d3))) * mat_a21+
            real(to_integer(unsigned(green_d3))) * mat_a22+
            real(to_integer(unsigned(blue_d3))) * mat_a23;
    if (varG < 0.0) then
      calc_G <= 0.0;
    else
      calc_G <= varG;
    end if;
  end if;
end process;

-------------------------------------------------------------------------------
-- Ce process calcul la valeur attendue de la composante B.
-------------------------------------------------------------------------------
process(clk)
  variable varB : real := 0.0;
begin
  if rising_edge(clk) then
    varB := real(to_integer(unsigned(red_d3))) * mat_a31+
            real(to_integer(unsigned(green_d3))) * mat_a32+
            real(to_integer(unsigned(blue_d3))) * mat_a33;
    if (varB < 0.0) then
      calc_B <= 0.0;
    else
      calc_B <= varB;
    end if;
  end if;
end process;

-------------------------------------------------------------------------------
-- Ce process compare les valeurs des sorties du module à vérifier avec les
-- valeurs attendues.
-------------------------------------------------------------------------------
process(clk)
  variable varOutR : real := 0.0;
  variable varOutG : real := 0.0;
  variable varOutB : real := 0.0;
  variable compte : natural range 0 to 4 := 0;
```

```
  begin
    if falling_edge(clk) then

      case state is
               -- Décalage de 4 coups d'horloge pour tenir compte de la latence
               -- de 4 clk du module à vérifier.
      when '0' =>
        if compte = 4 then
          state <= '1';
        else
          compte := compte+1;
        end if;
      -- Vérification des sorties du module à vérifier et génétration des
      -- messages d'erreur si la différence entre les sorties du module
      -- et les valeurs attendues est supérieur à une unité.
      when '1' =>
        varOutR := real(to_integer(unsigned(red_out)));
        varOutG := real(to_integer(unsigned(Green_out)));
        varOutB := real(to_integer(unsigned(blue_out)));

        assert (abs(Calc_R - varOutR) <= 0.5)
          report "Erreur : red_calc="&real'image(Calc_R)&" et red_out="
                   &real'image(varOutR)&"." severity error;
        assert (abs(Calc_G - varOutG) <= 0.5)
          report "Erreur : green_calc="&real'image(Calc_G)&" et green_out="
                   &real'image(varOutG)&"." severity error;
        assert (abs(Calc_B - varOutB) <= 0.5)
          report "Erreur : blue_calc="&real'image(Calc_B)&" et blue_out="
                   &real'image(varOutB)&"." severity error;
      when others =>
        state <= '0';
      end case;

    end if;
  end process;

end arch;
```

ANNEXE 9 – Code VHDL de 'sensor_control.vhd'

```
--------------------------------------------------------------------------------
-- Titre         : Sensor control
-- Projet        : Capteur d'images
--------------------------------------------------------------------------------
-- Fichier       : sensor_control.vhd
-- Auteur        : Mohamed Sebbar
--------------------------------------------------------------------------------
-- Description   : Circuits de génération des signaux de synchronisation, de
--                 contrôle et d'acquisition pour le capteur d'images.
--------------------------------------------------------------------------------
-- Historique de modification :
-- 2009/06/04 : Génération des signaux de synchronisation par J.-P. David.
-- 2010/05/03 : Transformation en composante SOPC pour NIOS, ajout de machines
--              à états et ajout des circuits de contrôle et d'acquisition par
--              M. Sebbar.
-- 2011/04/05 : Modification pour le prototype E du capteur d'images par
--              M. Sebbar.
--------------------------------------------------------------------------------
library ieee ;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity sensor_control is
  generic (
    image_height : integer := 90; -- Nombre de lignes de l'image
    image_width  : integer := 120 -- Nombre de colonnes de l'image
  );

  port(
    -- Côté NIOS
    reset_n     : in std_logic;
    clk         : in std_logic;
    addr        : in std_logic_vector(3 downto 0);
    writedata   : in std_logic_vector(31 downto 0);
    readdata    : out std_logic_vector(31 downto 0);
    wr          : in std_logic;
    rd          : in std_logic;
    waitrequest : out std_logic;

    -- Côté ADCs
    clk_20MHz  : in std_logic;
    adc_clk    : out std_logic;
    adc_data   : in std_logic_vector(9 downto 0);
    adc_oe1    : out std_logic;
    adc_oe2    : out std_logic;

    -- Carte DE2-70
    out_on_off : in std_logic;

    -- Signaux pour la synchronisation verticale du capteur d'images
    rn_row    : out std_logic;
    clk_row   : out std_logic;
    clk2_row  : out std_logic;
    d_row     : out std_logic;
    d2_row    : out std_logic;
    v_grst    : out std_logic;
    v_grwsel  : out std_logic;
```

```
    -- Signaux pour la synchronisation horizontale du capteur d'images
    rn_col    : out std_logic;
    clk_col   : out std_logic;
    d_col     : out std_logic;
    vn        : out std_logic;
    vp        : out std_logic;

    -- Signaux pour les afficheurs 7 segments
    hex7 : out std_logic_vector(6 downto 0);
    hex6 : out std_logic_vector(6 downto 0);
    hex5 : out std_logic_vector(6 downto 0);
    hex4 : out std_logic_vector(6 downto 0);
    hex3 : out std_logic_vector(6 downto 0);
    hex2 : out std_logic_vector(6 downto 0);
    hex1 : out std_logic_vector(6 downto 0);
    hex0 : out std_logic_vector(6 downto 0)
  );
end sensor_control;

architecture behav of sensor_control is

component color_conversion is
  generic (
    mat_coef_width : integer := 9;
    fixed_point    : integer := 7;
    color_width    : integer := 10
  );
  port(
    reset_n  : in std_logic;
    clk      : in std_logic;

    -- Composantes RGB d'entrées
    red_in   : in std_logic_vector(color_width-1 downto 0);
    green_in : in std_logic_vector(color_width-1 downto 0);
    blue_in  : in std_logic_vector(color_width-1 downto 0);

    -- Coefficients de la matrice de conversion 3x3
    mat_a11  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a12  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a13  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a21  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a22  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a23  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a31  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a32  : in std_logic_vector(mat_coef_width-1 downto 0);
    mat_a33  : in std_logic_vector(mat_coef_width-1 downto 0);

    -- Composantes RGB converties
    red_out  : out std_logic_vector(color_width-1 downto 0);
    green_out: out std_logic_vector(color_width-1 downto 0);
    blue_out : out std_logic_vector(color_width-1 downto 0)
  );
end component;

-- FIFO dual clock
component FIFO0
  port(
    data    : in std_logic_vector(31 downto 0);
    rdclk   : in std_logic;
    rdreq   : in std_logic;
    wrclk   : in std_logic;
    wrreq   : in std_logic;
    q       : out std_logic_vector(31 downto 0);
```

```
    rdempty : out std_logic;
    wrfull  : out std_logic
  );
end component;

component bintobcd is
  port(
    bin : in std_logic_vector(15 downto 0);
    bcd : out std_logic_vector(15 downto 0)
  );
end component;

component decod7seg is
  port(
    i : in std_logic_vector(3 downto 0);
    o : out std_logic_vector(6 downto 0)
  );
end component;

constant global_width   : integer := 32;
constant mat_coef_width : integer := 12;--9;
constant fixed_point    : integer := 10;--7;
constant color_width    : integer := 10;

signal gate_reset_time  : std_logic_vector(global_width-1 downto 0);
signal integration_time : std_logic_vector(global_width-1 downto 0);
signal vgrwsel_width    : std_logic_vector(global_width-1 downto 0);
signal mat_a11 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a12 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a13 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a21 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a22 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a23 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a31 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a32 : std_logic_vector(mat_coef_width-1 downto 0);
signal mat_a33 : std_logic_vector(mat_coef_width-1 downto 0);

signal nbr_col : std_logic_vector(global_width-1 downto 0);
signal nbr_row : std_logic_vector(global_width-1 downto 0);

signal s_clk : std_logic;
signal s_row : std_logic_vector(global_width-1 downto 0);
signal s_col : std_logic_vector(global_width-1 downto 0);

signal s_red_out : std_logic_vector(color_width-1 downto 0);
signal s_green_out : std_logic_vector(color_width-1 downto 0);
signal s_blue_out : std_logic_vector(color_width-1 downto 0);

signal s_col_i1 : std_logic;
signal s_col_i2 : std_logic;
signal s_col_i3 : std_logic;
signal s_col_i4 : std_logic;
signal s_col_i5 : std_logic;
signal s_row_i1 : std_logic;
signal s_row_i2 : std_logic;

signal s_bcd_row : std_logic_vector(15 downto 0);
signal s_bcd_col : std_logic_vector(15 downto 0);

signal s_active : std_logic;
signal s_start : std_logic;
```

```
type state1_type is (INIT_S1, WAIT1_S1, WAIT2_S1, WAIT3_S1, WAIT4_S1, STATE1_S1,
STATE2_S1, STATE3_S1);
signal state1 : state1_type;

type state2_type is (INIT_S2, WAIT1_S2, WAIT2_S2, WAIT3_S2, WAIT4_S2, WAIT5_S2,
WAIT6_S2, WAIT7_S2, WAIT8_S2, STATE1_S2);
signal state2 : state2_type;

type state3_type is (INIT1_S3, INIT2_S3, INIT3_S3, STATE1_S3);
signal state3 : state3_type;

signal s_counter  : std_logic_vector(15 downto 0);
signal s_counter1 : std_logic_vector(global_width-1 downto 0);
signal s_counter2 : std_logic_vector(global_width-1 downto 0);
signal s_counter3 : std_logic_vector(global_width-1 downto 0);
signal s_counter4 : std_logic_vector(15 downto 0);
signal s_calcul1  : std_logic_vector(2*global_width-1 downto 0);
signal s_calcul2  : std_logic_vector(2*global_width-1 downto 0);
signal s_calcul3  : std_logic_vector(2*global_width-1 downto 0);

signal s_fifo_data  : std_logic_vector(31 downto 0);
signal s_fifo_wr    : std_logic;
signal s_fifo_full  : std_logic;
signal s_fifo_empty : std_logic;

signal s_adc_data1 : std_logic_vector(9 downto 0);
signal s_adc_data2 : std_logic_vector(9 downto 0);
signal s_adc_data3 : std_logic_vector(9 downto 0);
signal s_adc_oe    : std_logic_vector(1 downto 0);

signal s_registre : std_logic_vector(2 downto 0);

begin

-- Cette FIFO est remplie par state_machine2 (domaine d'horloge 20MHz/3)
-- et vidée par le NIOS à travers le bus Avalon (domaine d'horloge 50Mz)
FIFO0_inst : FIFO0
  port map(
    data    => s_fifo_data,
    wrclk   => clk_20MHz,
    wrreq   => s_fifo_wr,
    wrfull  => s_fifo_full,
    rdclk   => clk,
    rdreq   => rd,
    q       => readdata,
    rdempty => s_fifo_empty
  );

color_conversion_inst : color_conversion
  generic map(
    mat_coef_width => mat_coef_width,
    fixed_point    => fixed_point,
    color_width    => color_width
  )
  port map(
    reset_n  => reset_n,
    clk      => clk,
    red_in   => s_adc_data1,
    green_in => s_adc_data2,
    blue_in  => s_adc_data3,
    mat_a11  => mat_a11,
    mat_a12  => mat_a12,
    mat_a13  => mat_a13,
```

```
    mat_a21  => mat_a21,
    mat_a22  => mat_a22,
    mat_a23  => mat_a23,
    mat_a31  => mat_a31,
    mat_a32  => mat_a32,
    mat_a33  => mat_a33,
    red_out  => s_red_out,
    green_out=> s_green_out,
    blue_out => s_blue_out
  );


waitrequest <= rd and s_fifo_empty;

nbr_col <= vgrwsel_width + image_width;
nbr_row <= integration_time + std_logic_vector( to_unsigned(image_height,global_width)
) + 1;
s_calcul1 <= integration_time * ( std_logic_vector(
to_unsigned(image_width,global_width) ) + vgrwsel_width );
s_calcul2 <= (integration_time + std_logic_vector(
to_unsigned(image_height,global_width) ) ) *
             ( std_logic_vector( to_unsigned(image_width,global_width) ) +
vgrwsel_width );
s_calcul3 <= (integration_time * ( std_logic_vector(
to_unsigned(image_width,global_width) ) +
             vgrwsel_width ) )  + ( std_logic_vector(
to_unsigned(image_width,global_width) ) );

s_clk <= clk_20MHz;

adc_clk <= s_registre(2);

(adc_oe1, adc_oe2) <= s_adc_oe;

-- Ce process génère les deux signaux adc_oe1 et adc_oe2 pour l'activation à tour de
-- rôle des sorties des trois ADCs sur la carte de conversion A/N.
process(s_registre(0), s_registre(1), s_registre(2))
begin
  if s_registre(2)='1' then
    s_adc_oe <= "00";
  elsif s_registre(1)='1' then
    s_adc_oe <= "01";
  elsif s_registre(0)='1' then
    s_adc_oe <= "10";
  else
    s_adc_oe <= "11";
  end if;
end process;

-- Ce process permet au NIOS de modifier les valeurs de différents registres à travers
-- le bus Avalon
update_parameters : process(clk, reset_n)
begin
  if (reset_n = '0') then
    s_start <= '0';
    gate_reset_time  <= std_logic_vector(to_unsigned(180,global_width));
    integration_time <= std_logic_vector(to_unsigned(200,global_width));
    vgrwsel_width    <= std_logic_vector(to_unsigned(2,global_width));
    mat_a11 <= (others => '0');
    mat_a12 <= (others => '0');
    mat_a13 <= (others => '0');
    mat_a21 <= (others => '0');
    mat_a22 <= (others => '0');
```

```
    mat_a23 <= (others => '0');
    mat_a31 <= (others => '0');
    mat_a32 <= (others => '0');
    mat_a33 <= (others => '0');
  elsif (clk'event and clk='1') then
    if (wr='1') then
      case addr is
      when "0000" =>
        s_start  <= writedata(0);
      when "0001" =>
        gate_reset_time  <= writedata;
      when "0010" =>
        integration_time <= writedata;
      when "0011" =>
        vgrwsel_width <= writedata;
      ---------------------------------
      when "0100" =>
        mat_a11 <= writedata(mat_coef_width-1 downto 0);
      when "0101" =>
        mat_a12 <= writedata(mat_coef_width-1 downto 0);
      when "0110" =>
        mat_a13 <= writedata(mat_coef_width-1 downto 0);
      when "0111" =>
        mat_a21 <= writedata(mat_coef_width-1 downto 0);
      when "1000" =>
        mat_a22 <= writedata(mat_coef_width-1 downto 0);
      when "1001" =>
        mat_a23 <= writedata(mat_coef_width-1 downto 0);
      when "1010" =>
        mat_a31 <= writedata(mat_coef_width-1 downto 0);
      when "1011" =>
        mat_a32 <= writedata(mat_coef_width-1 downto 0);
      when "1100" =>
        mat_a33 <= writedata(mat_coef_width-1 downto 0);
      ---------------------------------
      when others =>
        s_start  <= s_start;
      end case;
    end if;
  end if;
end process update_parameters;

-- Registre à décalage à droite de trois bits dont la valeur initiale est "100".
-- Le '1' circulant dans le registre est utilisé pour synchroniser les opérations
-- des autres process
registre_decalage : process(s_clk, reset_n)
begin
  if (reset_n = '0') then
    s_registre <= "100";
  elsif (s_clk'event and s_clk='1') then
    s_registre <= s_registre(0) & s_registre(2 downto 1);
  end if;
end process registre_decalage;

-- Cette machine à états permet d'acquérir les valeurs des trois ADCs sur la
-- carte de conversion A/N. L'acquisition est démarrée par le premier pulse de d_col.
-- L'attente au début permet de prendre en considération la latence des ADCs. Les
-- valeurs acquisitionnées sont envoyées au circuit color_conversion.
state_machine1 : process (s_clk, reset_n)
begin
  if (reset_n = '0') then
    state1 <= INIT_S1;
    s_counter <= (others => '0');
```

```
    s_adc_data1 <= (others => '0');
    s_adc_data2 <= (others => '0');
    s_adc_data3 <= (others => '0');
  elsif (s_clk'event and s_clk = '1') then
    case state1 is
    when INIT_S1 =>
      s_counter <= (others => '0');
      if (s_col_i4 = '1') then --d_col
        state1   <= WAIT1_S1;
      end if;
    when WAIT1_S1 =>
      if (s_registre(2) = '1') then
        state1   <= WAIT2_S1;
      end if;
    when WAIT2_S1 =>
      if (s_registre(2) = '1') then
        state1   <= WAIT3_S1;
      end if;
    when WAIT3_S1 =>
      if (s_registre(2) = '1') then
        state1   <= WAIT4_S1;
      end if;
    when WAIT4_S1 =>
      if (s_registre(2) = '1') then
        state1   <= STATE1_S1;
      end if;
--
    when STATE1_S1 =>
      if(s_registre(2) = '1') then
        s_adc_data1 <= adc_data;
        state1      <= STATE2_S1;
      end if;
    when STATE2_S1 =>
      if(s_registre(1) = '1') then
        s_adc_data2 <= adc_data;
        state1      <= STATE3_S1;
      end if;
    when STATE3_S1 =>
      if(s_registre(0) = '1') then
        s_adc_data3 <= adc_data;
        if(s_counter = image_width-1) then
          state1  <= INIT_S1;
        else
          state1  <= STATE1_S1;
        end if;
      end if;
    when others =>
      state1 <= INIT_S1;
    end case;
  end if;
end process state_machine1;

-- Cette machine à états permet de sauvegarder les valeurs calculées par le circuit
-- color_conversion dans la mémoire FIFO. L'opération est démarrée par le premier
-- pulse de d_col. L'attente au début permet de prendre en considération la latence
-- des ADCs et du circuit color_conversion.
state_machine2 : process (s_clk, reset_n)
begin
  if (reset_n = '0') then
    state2 <= INIT_S2;
    s_fifo_wr <= '0';
    s_counter4 <= (others => '0');
    s_fifo_data <= (others => '0');
```

```
  elsif (s_clk'event and s_clk = '1') then
    s_fifo_wr <= '0';
    case state2 is
    when INIT_S2 =>
      s_counter4 <= (others => '0');
      if (s_col_i4 = '1') then --d_col
        state2   <= WAIT1_S2;
      end if;
--
    when WAIT1_S2 =>
      if (s_registre(2) = '1') then
        state2   <= WAIT2_S2;
      end if;
    when WAIT2_S2 =>
      if (s_registre(2) = '1') then
        state2   <= WAIT3_S2;
      end if;
    when WAIT3_S2 =>
      if (s_registre(2) = '1') then
        state2   <= WAIT4_S2;
      end if;
    when WAIT4_S2 =>
      if (s_registre(2) = '1') then
        state2   <= WAIT5_S2;
      end if;
--
    when WAIT5_S2 =>
      state2   <= WAIT6_S2;
    when WAIT6_S2 =>
      state2   <= WAIT7_S2;
    when WAIT7_S2 =>
      state2   <= WAIT8_S2;
    when WAIT8_S2 =>
      state2   <= STATE1_S2;
--
    when STATE1_S2 =>
      s_fifo_wr <= '0';
      if(s_registre(1) = '1') then
        s_fifo_wr <= '1';
        s_counter4 <= s_counter4+1;
        s_fifo_data <= "11" & s_blue_out & s_green_out & s_red_out;
        if(s_counter4 = image_width-1) then
          state2  <= INIT_S2;
        else
          state2  <= STATE1_S2;
        end if;
      end if;
    when others =>
      state2 <= INIT_S2;
    end case;
  end if;
end process state_machine2;

-- Cette machine à états permet de générer les signaux de synchronisation pour
-- le capteur d'images. Cette opération est démarrée par le signal s_start.
state_machine3 : process(s_clk, reset_n)
  variable v_row : std_logic_vector(global_width-1 downto 0);
  variable v_col : std_logic_vector(global_width-1 downto 0);

begin
  if (reset_n = '0') then
    s_row <= (others => '0');
    s_col <= (others => '0');
```

```
    ----------------------------
    s_col_i1 <= '0'; -- clk_row, clk2_row
    s_col_i2 <= '0'; -- v_grst
    s_col_i3 <= '0'; -- v_grwsel, vn, vp
    s_col_i4 <= '0'; -- d_col
    ----------------------------
    s_row_i1 <= '0'; -- d_row
    s_row_i2 <= '0'; -- d2_row

    s_active <= '0';
    s_counter1 <= (others => '0');
    s_counter2 <= (others => '0');
    s_counter3 <= (others => '0');
    state3 <= INIT1_S3;

  elsif (s_clk'event and s_clk = '1') then
    case state3 is
    when INIT1_S3 =>
      if (s_start='1') then
       state3 <= INIT2_S3;
      end if;
      s_row <= (others => '0');
      s_col <= (others => '0');
      ----------------------------
      s_col_i1 <= '0'; -- clk_row, clk2_row
      s_col_i2 <= '0'; -- v_grst
      s_col_i3 <= '0'; -- v_grwsel, vn, vp
      s_col_i4 <= '0'; -- d_col
      ----------------------------
      s_row_i1 <= '0'; -- d_row
      s_row_i2 <= '0'; -- d2_row

      s_active <= '0';
      s_counter1 <= (others => '0');
      s_counter2 <= (others => '0');
      s_counter3 <= (others => '0');
    when INIT2_S3 =>
      s_row_i1 <= '1'; -- d_row
      if(s_counter1 = 63) then --63
        state3 <= INIT3_S3;
      else
        s_counter1 <= s_counter1 + 1;
      end if;
    when INIT3_S3 =>
      s_col_i1 <= '1'; -- clk_row
      s_col_i2 <= '1'; -- v_grst
      state3 <= STATE1_S3;
    when STATE1_S3 =>
      if(s_registre(2) = '1') then
        v_row := s_row;
        v_col := s_col + 1;
        if (s_col = nbr_col - 1) then
          v_col := (others => '0');
          v_row := s_row + 1;
          if (v_row = nbr_row) then
            v_row := (others => '0');
          end if;
        end if;

        s_row <= v_row;
        s_col <= v_col;
        ----------------------------------------
        -- clk_row, clk2_row
```

```
        if (s_col = nbr_col-1) then s_col_i1 <= '1';
        elsif (s_col = '0'&nbr_col(global_width-1 downto 1)) then s_col_i1 <= '0';
        end if;
        -- v_grst
        if (s_col = nbr_col-1) then s_col_i2 <= '1';
        elsif (s_col = gate_reset_time-1) then s_col_i2 <= '0';
        end if;
        -- v_grwsel, vn, vp
        if (s_col = nbr_col-1 and s_active = '1') then s_col_i3 <= '1';
        elsif (s_col = vgrwsel_width-1) then s_col_i3 <= '0';
        end if;
        -- d_col
        if (s_col = vgrwsel_width-2 and s_active = '1') then s_col_i4 <= '1';
        elsif (s_col = vgrwsel_width-1) then s_col_i4 <= '0';
        end if;
        -- s_active
        if (s_counter2 = s_calcul1(global_width-1 downto 0)) then s_active <= '1';
        elsif (s_counter2 = s_calcul2(global_width-1 downto 0)) then s_active <= '0';
        end if;
        ---------------------------------------------
        -- d_row
        if (s_col = '0'&nbr_col(global_width-1 downto 1)) then
          if (s_row = 0) then s_row_i1 <= '0';
          end if;
        end if;
        -- d2_row
        if (s_counter3 = s_calcul3(global_width-1 downto 0) ) then s_row_i2 <= '1';
        elsif (s_row = integration_time+1) then s_row_i2 <= '0';
        end if;
        s_counter2 <= s_counter2 + 1;
        s_counter3 <= s_counter3 + 1;
        ---------------------------------------------
        if (s_row = nbr_row-1 and s_col = nbr_col-1) then
          state3 <= INIT1_S3;
        end if;
      end if;
    when others =>
      state3 <= INIT1_S3;
    end case;
  end if;
end process state_machine3;

-- Ce prosess permet d'activer et désactiver les sorties vers le capteur d'images
-- à l'aide du commutateur out_on_off.
process(out_on_off, s_col_i1, s_col_i2, s_col_i3, s_col_i4, s_row_i1, s_row_i2,
        s_registre(2))
begin
  clk_row   <= '0';
  clk2_row  <= '0';
  v_grst    <= '0';
  v_grwsel  <= '0';
  d_col     <= '0';
  d_row     <= '0';
  d2_row    <= '0';
  rn_row    <= '0';
  rn_col    <= '0';
  clk_col   <= '0';
  vn        <= '0';
  vp        <= '0';
  hex7      <= (others => '1');  --OFF
  hex6      <= (others => '1');  --OFF

  if (out_on_off='1') then
```

```
    clk_row  <= s_col_i1;
    clk2_row <= s_col_i1;
    v_grst   <= s_col_i2;
    v_grwsel <= s_col_i3;
    d_col    <= s_col_i4;
    d_row    <= s_row_i1;
    d2_row   <= s_row_i2;
    rn_row   <= '1';
    rn_col   <= '1';
    clk_col  <= s_registre(2);
    vn       <= s_col_i3;
    vp       <= not s_col_i3;
    hex7     <= (others => '0'); --ON
    hex6     <= (others => '0'); --ON
  end if;
end process;

bcd1 : bintobcd port map (
  s_row(15 downto 0),
  s_bcd_row
);

bcd2 : bintobcd port map (
  s_col(15 downto 0),
  s_bcd_col
);

disp5 : decod7seg port map(
  s_bcd_row(7 downto 4),
  hex5
);

disp4 : decod7seg port map(
  s_bcd_row(3 downto 0),
  hex4
);

disp3 : decod7seg port map(
  s_bcd_col(15 downto 12),
  hex3
);

disp2 : decod7seg port map(
  s_bcd_col(11 downto 8),
  hex2
);

disp1 : decod7seg port map(
  s_bcd_col(7 downto 4),
  hex1
);

disp0 : decod7seg port map(
  s_bcd_col(3 downto 0),
  hex0
);

end behav;
```

ANNEXE 10 – Code VHDL de ‘sensor_control_tb.vhd’

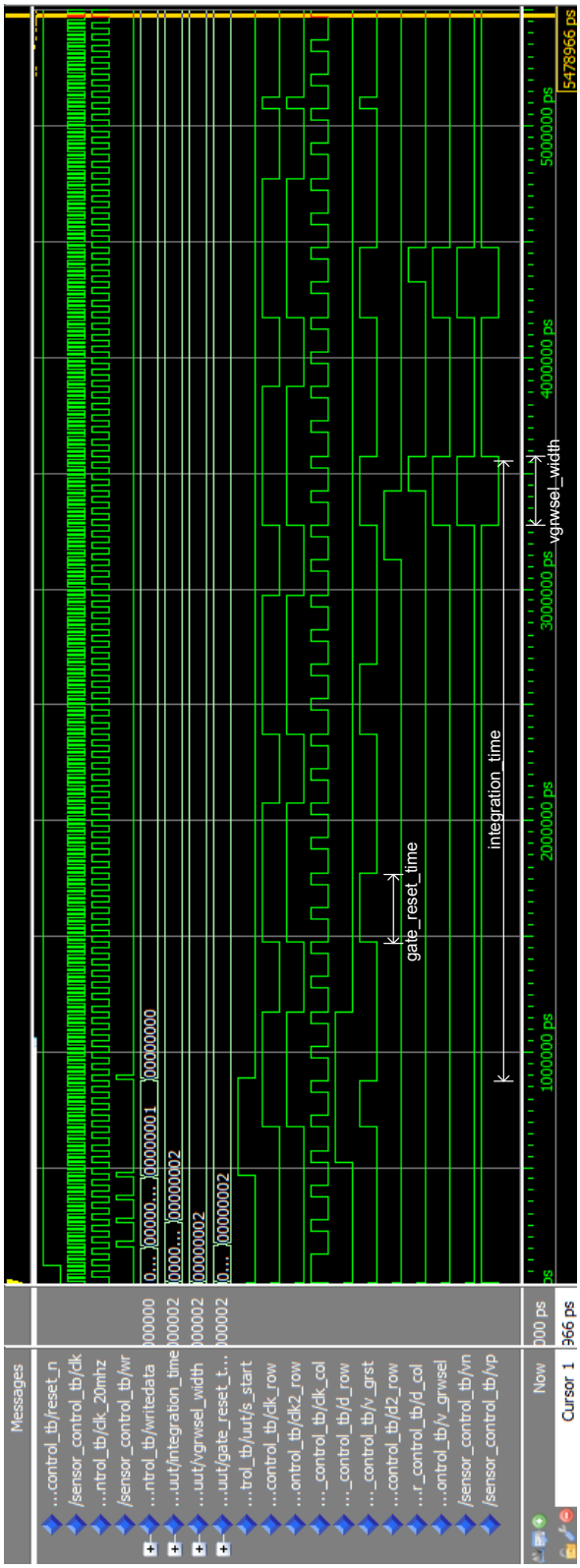
```
--------------------------------------------------------------------------------
-- Titre         : Sensor control Testbench
-- Projet        : Capteur d'images
--------------------------------------------------------------------------------
-- Fichier       : sensor_control_tb.vhd
-- Auteur        : Mohamed Sebbar
--------------------------------------------------------------------------------
-- Description   : Testbench pour le module sensor_control.
--------------------------------------------------------------------------------
-- Historique de modification :
-- 2010/05/03 : création
--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

-- Ce testbench test le module sensor_control.
entity sensor_control_tb is
end sensor_control_tb;

architecture test_bench of sensor_control_tb is

constant clk_periode        : time := 20 ns;
constant clk_20MHz_periode  : time := 50 ns;
constant image_height : integer := 2;
constant image_width  : integer := 4;

signal clk : std_logic := '0';
signal reset_n : std_logic := '0';
signal addr : std_logic_vector(3 downto 0) := (others => '0');
signal writedata : std_logic_vector(31 downto 0) := (others => '0');
signal wr   : std_logic := '0';
signal rd   : std_logic := '0';
signal out_on_off : std_logic := '0';
signal clk_20MHz : std_logic := '0';
signal adc_clk  : std_logic := '0';
signal adc_data : std_logic_vector(9 downto 0) := (others => '0');
signal adc_oe1  : std_logic;
signal adc_oe2  : std_logic;
signal rn_row   : std_logic;
signal clk_row  : std_logic;
signal clk2_row : std_logic;
signal d_row    : std_logic;
signal d2_row   : std_logic;
signal v_grst   : std_logic;
signal v_grwsel : std_logic;
signal rn_col   : std_logic;
signal clk_col  : std_logic;
signal d_col    : std_logic;
signal vn : std_logic;
signal vp : std_logic;



--------------------------------------------------------------------------------
-- Déclaration du module à vérifier.
--------------------------------------------------------------------------------
```

```
component sensor_control
  generic (
    image_height : integer := 90; -- Nombre de lignes de l'image
    image_width  : integer := 120 -- Nombre de colonnes de l'image
  );

  port(
    -- Côté NIOS
    reset_n     : in std_logic;
    clk         : in std_logic;
    addr        : in std_logic_vector(3 downto 0);
    writedata   : in std_logic_vector(31 downto 0);
    readdata    : out std_logic_vector(31 downto 0);
    wr          : in std_logic;
    rd          : in std_logic;
    waitrequest : out std_logic;

    -- Côté ADCs
    clk_20MHz  : in std_logic;
    adc_clk    : out std_logic;
    adc_data   : in std_logic_vector(9 downto 0);
    adc_oe1    : out std_logic;
    adc_oe2    : out std_logic;

    -- Carte DE2-70
    out_on_off : in std_logic;

    -- Signaux pour la synchronisation verticale du capteur d'images
    rn_row    : out std_logic;
    clk_row   : out std_logic;
    clk2_row  : out std_logic;
    d_row     : out std_logic;
    d2_row    : out std_logic;
    v_grst    : out std_logic;
    v_grwsel  : out std_logic;

    -- Signaux pour la synchronisation horizontale du capteur d'images
    rn_col    : out std_logic;
    clk_col   : out std_logic;
    d_col     : out std_logic;
    vn        : out std_logic;
    vp        : out std_logic;

    -- Signaux pour les afficheurs 7 segments
    hex7 : out std_logic_vector(6 downto 0);
    hex6 : out std_logic_vector(6 downto 0);
    hex5 : out std_logic_vector(6 downto 0);
    hex4 : out std_logic_vector(6 downto 0);
    hex3 : out std_logic_vector(6 downto 0);
    hex2 : out std_logic_vector(6 downto 0);
    hex1 : out std_logic_vector(6 downto 0);
    hex0 : out std_logic_vector(6 downto 0)
  );
end component sensor_control;

begin

-------------------------------------------------------------------------------
-- instanciation du module à vérifier.
-------------------------------------------------------------------------------
UUT : sensor_control
  generic map(
    image_height => image_height,
```

```
    image_width  => image_width
  )
  port map(
    reset_n     => reset_n,
    clk         => clk,
    addr        => addr,
    writedata   => writedata,
    readdata    => open,
    wr          => wr,
    rd          => rd,
    waitrequest => open,
    clk_20MHz  => clk_20MHz,
    adc_clk    => adc_clk,
    adc_data   => adc_data,
    adc_oe1    => adc_oe1,
    adc_oe2    => adc_oe2,
    out_on_off => out_on_off,
    rn_row   => rn_row,
    clk_row  => clk_row,
    clk2_row => clk2_row,
    d_row    => d_row,
    d2_row   => d2_row,
    v_grst   => v_grst,
    v_grwsel => v_grwsel,
    rn_col   => rn_col,
    clk_col  => clk_col,
    d_col    => d_col,
    vn       => vn,
    vp       => vp,
    hex7 => open,
    hex6 => open,
    hex5 => open,
    hex4 => open,
    hex3 => open,
    hex2 => open,
    hex1 => open,
    hex0 => open
  );

-- Génération des horloges
clk       <= not clk after clk_periode/2;
clk_20MHz <= not clk_20MHz after clk_20MHz_periode/2;

-- Process de génération des signaux de test
test : process
begin
   reset_n      <= '0';
   out_on_off     <= '1';
   wait for 4*clk_periode;
   reset_n      <= '1';
   wait for 4*clk_periode;
   addr       <= "0001";
   writedata  <= std_logic_vector(to_unsigned(2,32)); --gate_reset_time
   wr         <= '1';
   wait for 1*clk_periode;
   wr         <= '0';
   wait for 4*clk_periode;
   addr       <= "0010";
   writedata  <= std_logic_vector(to_unsigned(2,32)); --integration_time
   wr         <= '1';
   wait for 1*clk_periode;
   wr         <= '0';
   wait for 4*clk_periode;
```

```
   addr       <= "0011";
   writedata  <= std_logic_vector(to_unsigned(2,32)); --vgrwsel_width
   wr         <= '1';
   wait for 1*clk_periode;
   wr         <= '0';
   wait for 4*clk_periode;
   addr       <= "0000";
   writedata  <= std_logic_vector(to_unsigned(1,32));
   wr         <= '1';
   wait for 1*clk_periode;
   wr         <= '0';
   wait for 20*clk_periode;
   addr       <= "0000";
   writedata  <= std_logic_vector(to_unsigned(0,32));
   wr         <= '1';
   wait for 1*clk_periode;
   wr         <= '0';
   wait for 50*clk_periode;
   rd         <= '1';
   wait;
end process;

end test_bench;
```

ANNEXE 11 – Simulation de ‘sensor_control’



ANNEXE 12 – Code source de ‘cpu_0.c’

```
/*********************************************************************************
 * Projet        : Capteur d'images
*********************************************************************************
 * Fichier       : cpu_0.c
 * Auteur        : Mohamed Sebbar
*********************************************************************************
 * Historique de modification :
 * 2010/05/06 : Création.
*********************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include "alt_types.h"
#include "system.h"
#include "altera_avalon_pio_regs.h"
#include "altera_up_avalon_video_pixel_buffer_dma.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"
#include "alt_tpo_lcd.h"
#include "touch_spi.h"
#include "button.h"
#include "image.h"
#include "sensor_control.h"

#define DEBUG_PRINT

TOUCH_HANDLE hTouch;
alt_up_pixel_buffer_dma_dev s_pixel_buf_dev;
alt_up_pixel_buffer_dma_dev *pixel_buf_dev = &s_pixel_buf_dev;
alt_up_char_buffer_dev s_char_buffer_dev;
alt_up_char_buffer_dev *char_buffer_dev = &s_char_buffer_dev;

// Les quatre boutons à afficher sur l'écran
button_struct s_button4;
button_struct *button4 = &s_button4;
button_struct s_button6;
button_struct *button6 = &s_button6;
button_struct s_button8;
button_struct *button8 = &s_button8;
button_struct s_button10;
button_struct *button10 = &s_button10;

int back_buffer = 1;
int exposure_time = 0;
int zoom_2x = 0;
int video_mode = 0;
int capture = 0;

// image de zoom 1x
image_struct s_image1x;
image_struct *image1x = &s_image1x;

// image de zoom 2x
```

```
image_struct s_image2x;
image_struct *image2x = &s_image2x;

void lcd_init();
void pen_detect();
void page0_draw();

int main()
{
  // Initialisation du module LCD.
  lcd_init();

  // Boutton 'Exposure'
  button4->position_x = MENU_START_X;
  button4->position_y = MENU_START_Y+3*BUTTON_DISTANCE_V;
  button4->pressed = 0;
  button4->rectangle_color = BUTTON_RECT_COLOR;

  // Boutton 'Capture'
  button6->position_x = MENU_START_X;
  button6->position_y = MENU_START_Y+5*BUTTON_DISTANCE_V;
  button6->pressed = 0;
  button6->rectangle_color = BUTTON_RECT_COLOR;

  // Boutton 'Zoom'
  button8->position_x = MENU_START_X;
  button8->position_y = MENU_START_Y+7*BUTTON_DISTANCE_V;
  button8->pressed = 0;
  button8->rectangle_color = BUTTON_RECT_COLOR;

  // Boutton 'Mode'
  button10->position_x = MENU_START_X;
  button10->position_y = MENU_START_Y+9*BUTTON_DISTANCE_V;
  button10->pressed = 0;
  button10->rectangle_color = BUTTON_RECT_COLOR;

  // Initialisation de image1x
  image1x->width = WIDTH;
  image1x->height = HEIGHT;
  image1x->red = malloc(image1x->width*image1x->height*sizeof(alt_u16));
  image1x->green = malloc(image1x->width*image1x->height*sizeof(alt_u16));
  image1x->blue = malloc(image1x->width*image1x->height*sizeof(alt_u16));

  // Initialisation de image2x
  image2x->width = 2*WIDTH;
  image2x->height = 2*HEIGHT;
  image2x->red = malloc(image2x->width*image2x->height*sizeof(alt_u16));
  image2x->green = malloc(image2x->width*image2x->height*sizeof(alt_u16));
  image2x->blue = malloc(image2x->width*image2x->height*sizeof(alt_u16));

  back_buffer = 0;
  page0_draw();
  // Dessiner le rectangle pour encadrer l'image
  alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, 68, 83, 189, 174,
0xFFFFFF, back_buffer);

  back_buffer = 1;
```

```
  page0_draw();
  // Dessiner le rectangle pour encadrer l'image
  alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, 68, 83, 189, 174,
0xFFFFFF, back_buffer);

  SENSOR_CONTROL_SET_RESET_TIME(50);
  SENSOR_CONTROL_SET_VGRWSEL_WIDTH(45);

  SENSOR_CONTROL_SET_MAT_A11(1.0);
  SENSOR_CONTROL_SET_MAT_A12(0.0);
  SENSOR_CONTROL_SET_MAT_A13(0.0);
  SENSOR_CONTROL_SET_MAT_A21(0.0);
  SENSOR_CONTROL_SET_MAT_A22(1.0);
  SENSOR_CONTROL_SET_MAT_A23(0.0);
  SENSOR_CONTROL_SET_MAT_A31(0.0);
  SENSOR_CONTROL_SET_MAT_A32(0.0);
  SENSOR_CONTROL_SET_MAT_A33(1.0);
/**
  SENSOR_CONTROL_SET_MAT_A11(0.33757);
  SENSOR_CONTROL_SET_MAT_A12(0.73759);
  SENSOR_CONTROL_SET_MAT_A13(-0.10183);
  SENSOR_CONTROL_SET_MAT_A21(0.80957);
  SENSOR_CONTROL_SET_MAT_A22(-0.07984);
  SENSOR_CONTROL_SET_MAT_A23(0.27027);
  SENSOR_CONTROL_SET_MAT_A31(0.63579);
  SENSOR_CONTROL_SET_MAT_A32(-0.60609);
  SENSOR_CONTROL_SET_MAT_A33(0.70163);
/**/

  #ifdef DEBUG_PRINT
    printf("Hello from Nios II!\n");
  #endif

  while(1) {
    pen_detect();

    if(capture == 1) {
      SENSOR_CONTROL_CAPTURE();
      image_read(image1x);

      // Effacer image
      alt_up_pixel_buffer_dma_draw_box(pixel_buf_dev, 8, 38, 249, 219, 0,
back_buffer);

      if(zoom_2x == 1) {
        // Dessiner le rectangle pour encadrer l'image
        alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, 8, 38, 249, 219,
0xFFFFFF, back_buffer);
        image_zoom2x(image1x, image2x);
        image_draw(pixel_buf_dev, image2x, 9, 39, back_buffer);
      }else {
        // Dessiner le rectangle pour encadrer l'image
        alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, 68, 83, 189,
174, 0xFFFFFF, back_buffer);
        image_draw(pixel_buf_dev, image1x, 69, 84, back_buffer);
      }
      if (video_mode == 0)
```

```
        capture = 0;

      alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);
      while(alt_up_pixel_buffer_dma_check_swap_buffers_status(pixel_buf_dev)
== 1);
    }
  }

  return 0;
}

/*****************************************************************************
 * Cette fonction initialise le module LCD.
*****************************************************************************/
void lcd_init()
{
  int result;                 // Résultat de l'initialisation du LCD.
  alt_tpo_lcd s_lcd;          // Structure pour la configuration de
                              // l'écran LCD.
  alt_tpo_lcd *lcd = &s_lcd; // Pointeur vers la structure.

  // Spécification de l'adresse de base de chaque bus PIO de
  // communiction tels que définits dans system.h
  lcd->scen_pio = LCD_I2C_EN_BASE;
  lcd->scl_pio  = LCD_I2C_SCL_BASE;
  lcd->sda_pio  = LCD_I2C_SDAT_BASE;

  #ifdef DEBUG_PRINT
    printf("Intitializing LCD control registers & gamma curve...");
  #endif

  // Vérifier le Chip ID du module LCD, écrire la courbe Gamma et
  // la résolution 800x480.
  result = alt_tpo_lcd_init(lcd, 800, 480);
  #ifdef DEBUG_PRINT
    if(result) {
      printf("Failed.\n");
    } else {
      printf("Success.\n");
    }
  #endif

/*****************************************************************************/

  #ifdef DEBUG_PRINT
    printf("Initializing touchscreen...");
  #endif

  hTouch = Touch_Init(TOUCH_PANEL_SPI_BASE, TOUCH_PANEL_PEN_IRQ_N_BASE,
TOUCH_PANEL_PEN_IRQ_N_IRQ);
  Touch_EmptyFifo(hTouch);
  #ifdef DEBUG_PRINT
    if(!hTouch)
      printf("Failed.\n");
    else
      printf("Success.\n");
  #endif
```

```
/*****************************************************************************/

  // Ouverture de 'Pixel Buffer'
  pixel_buf_dev = alt_up_pixel_buffer_dma_open_dev (PIXEL_BUFFER_DMA_NAME);
  #ifdef DEBUG_PRINT
    if ( pixel_buf_dev == NULL)
      printf ("Error: could not open pixel buffer device.\n");
    else
      printf ("Opened pixel buffer device.\n");
  #endif

  // Ouverture de 'Character Buffer'
  char_buffer_dev =
alt_up_char_buffer_open_dev(CHAR_BUFFER_WITH_DMA_AVALON_CHAR_CONTROL_SLAVE_NAM
E);
  #ifdef DEBUG_PRINT
    if ( char_buffer_dev == NULL)
      printf ("Error: could not open character buffer device.\n");
    else
      printf ("Opened character buffer device.\n");
  #endif

  alt_up_pixel_buffer_dma_change_back_buffer_address(pixel_buf_dev,
0x00100000);
}

/*****************************************************************************
 * Cette fonction détermine l'action à faire en fonction des coordonnées (x,y)
 * du point où l'écran a été touché.
*****************************************************************************/
void pen_detect()
{
  int x = 0, y = 0;

  // L'écran n'a pas été touché
  if (!Touch_GetXY(hTouch, &x,&y))
    return;

  // Le bouton 'Exposure' a été touché
  if(x>=BUTTON4_X_MIN && x<=BUTTON4_X_MAX && y>=BUTTON4_Y_MIN &&
y<=BUTTON4_Y_MAX) {
    if(x>=BUTTON4_X_MIN+(BUTTON_WIDTH/2) && x<=BUTTON4_X_MAX &&
y>=BUTTON4_Y_MIN && y<=BUTTON4_Y_MAX) {
      if(exposure_time<8)
        exposure_time++;
      else
        exposure_time = 0;
    }else {
      if(exposure_time>0)
        exposure_time--;
      else
        exposure_time = 8;
    }
    #ifdef DEBUG_PRINT
      printf("Button4 press detected\n");
    #endif
```

```
    button4->pressed = 1;
    button_draw(pixel_buf_dev, char_buffer_dev, button4, 0);
    usleep(200000);
    button4->pressed = 0;
    button_draw(pixel_buf_dev, char_buffer_dev, button4, 0);

    switch (exposure_time) {
      case 0 : strcpy(button4->text,"Exposure: 1s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*IMAGE_WIDTH));
             break;
      case 1 : strcpy(button4->text,"Exposure: 1/2s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*2*IMAGE_WIDTH));
             break;
      case 2 : strcpy(button4->text,"Exposure: 1/4s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*4*IMAGE_WIDTH));
             break;
      case 3 : strcpy(button4->text,"Exposure: 1/8s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*8*IMAGE_WIDTH));
             break;
      case 4 : strcpy(button4->text,"Exposure: 1/10s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*10*IMAGE_WIDTH));
             break;
      case 5 : strcpy(button4->text,"Exposure: 1/20s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*20*IMAGE_WIDTH));
             break;
      case 6 : strcpy(button4->text,"Exposure: 1/40s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*40*IMAGE_WIDTH));
             break;
      case 7 : strcpy(button4->text,"Exposure: 1/80s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*80*IMAGE_WIDTH));
             break;
      case 8 : strcpy(button4->text,"Exposure: 1/100");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*100*IMAGE_WIDTH));
             break;
      default : ;
             #ifdef DEBUG_PRINT
               printf("Unexpected case value: pen_down_handler->switch-
>exposure_time.\n");
             #endif
    }/**/
    #ifdef DEBUG_PRINT
      printf("exposure_time : %d\n",exposure_time);
    #endif

  }
  // Le bouton 'Capture' a été touché
  else if(x>= BUTTON6_X_MIN && x <= BUTTON6_X_MAX && y>= BUTTON6_Y_MIN && y <=
BUTTON6_Y_MAX) {
```

```
    #ifdef DEBUG_PRINT
      printf("Button6 press detected\n");
    #endif
    button6->pressed = 1;
    button_draw(pixel_buf_dev, char_buffer_dev, button6, 0);
    usleep(200000);
    button6->pressed = 0;
    button_draw(pixel_buf_dev, char_buffer_dev, button6, 0);
    capture = 1;

  }
  // Le bouton 'Zoom' a été touché
  else if(x>= BUTTON8_X_MIN && x <= BUTTON8_X_MAX && y>= BUTTON8_Y_MIN && y <=
BUTTON8_Y_MAX) {
    #ifdef DEBUG_PRINT
      printf("Button8 press detected\n");
    #endif
    button8->pressed = 1;
    button_draw(pixel_buf_dev, char_buffer_dev, button8, 0);
    usleep(200000);
    button8->pressed = 0;
    button_draw(pixel_buf_dev, char_buffer_dev, button8, 0);

    if(zoom_2x == 0) {
      zoom_2x = 1;
      strcpy(button8->text,"Zoom 2x");
      button_update_text(char_buffer_dev, button8, 0);
      // Effacer image
      alt_up_pixel_buffer_dma_draw_box(pixel_buf_dev, 8, 38, 249, 219, 0,
back_buffer);
      alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, 8, 38, 249, 219,
0xFFFFFF, back_buffer);
      image_zoom2x(image1x, image2x);
      image_draw(pixel_buf_dev, image2x, 9, 39, back_buffer);

      alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);
      while(alt_up_pixel_buffer_dma_check_swap_buffers_status(pixel_buf_dev)
== 1);

    }else {
      zoom_2x = 0;
      strcpy(button8->text,"Zoom 1x");
      button_update_text(char_buffer_dev, button8, back_buffer);
      // Effacer image
      alt_up_pixel_buffer_dma_draw_box(pixel_buf_dev, 8, 38, 249, 219, 0,
back_buffer);
      alt_up_pixel_buffer_dma_draw_rectangle(pixel_buf_dev, 68, 83, 189, 174,
0xFFFFFF, back_buffer);
      image_draw(pixel_buf_dev, image1x, 69, 84, back_buffer);

      alt_up_pixel_buffer_dma_swap_buffers(pixel_buf_dev);
      while(alt_up_pixel_buffer_dma_check_swap_buffers_status(pixel_buf_dev)
== 1);

    }
    #ifdef DEBUG_PRINT
      printf("zoom_2x : %d\n",zoom_2x);
```

```
    #endif

  }
  // Le bouton 'Mode' a été touché
  else if(x>= BUTTON10_X_MIN && x <= BUTTON10_X_MAX && y>= BUTTON10_Y_MIN && y
<= BUTTON10_Y_MAX) {
    #ifdef DEBUG_PRINT
      printf("Button10 press detected\n");
    #endif

    button10->pressed = 1;
    button_draw(pixel_buf_dev, char_buffer_dev, button10, 0);
    usleep(200000);
    button10->pressed = 0;
    button_draw(pixel_buf_dev, char_buffer_dev, button10, 0);

    if(video_mode == 0) {
      video_mode = 1;
      strcpy(button10->text,"Video mode");
      button_update_text(char_buffer_dev, button10, back_buffer);
    }else {
      video_mode = 0;
      strcpy(button10->text,"Image mode");
      button_update_text(char_buffer_dev, button10, back_buffer);
    }

    #ifdef DEBUG_PRINT
      printf("Video mode : %d\n",video_mode);
    #endif

  }
  #ifdef DEBUG_PRINT
    else
      printf("x: %d, y: %d\n", x, y);
  #endif
}

/*****************************************************************************
 * Cette fonction efface l'écran et affiche le titre et les boutons.
*****************************************************************************/
void page0_draw()
{
  char p_char[30];

  // Effacer l'écran et vider le tampon des charctères
  alt_up_pixel_buffer_dma_clear_screen (pixel_buf_dev, back_buffer);
  alt_up_char_buffer_clear(char_buffer_dev);

  // Afficher le titre
  strcpy(p_char, "Image Sensor Project");
  alt_up_char_buffer_string(char_buffer_dev, p_char, 15, 0);

  // Dessiner trois lignes horizentales en rouge, vert et bleu
  alt_up_pixel_buffer_dma_draw_line(pixel_buf_dev, 0, 14, 399, 14, 0xFF0000,
back_buffer);
  alt_up_pixel_buffer_dma_draw_line(pixel_buf_dev, 0, 16, 399, 16, 0x00FF00,
back_buffer);
```

```
  alt_up_pixel_buffer_dma_draw_line(pixel_buf_dev, 0, 18, 399, 18, 0x0000FF,
back_buffer);

  // Afficher boutton 'Exposure'
  switch (exposure_time) {
    case 0 : strcpy(button4->text,"Exposure: 1s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*IMAGE_WIDTH));
             break;
    case 1 : strcpy(button4->text,"Exposure: 1/2s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*2*IMAGE_WIDTH));
             break;
    case 2 : strcpy(button4->text,"Exposure: 1/4s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*4*IMAGE_WIDTH));
             break;
    case 3 : strcpy(button4->text,"Exposure: 1/8s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*8*IMAGE_WIDTH));
             break;
    case 4 : strcpy(button4->text,"Exposure: 1/10s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*10*IMAGE_WIDTH));
             break;
    case 5 : strcpy(button4->text,"Exposure: 1/20s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*20*IMAGE_WIDTH));
             break;
    case 6 : strcpy(button4->text,"Exposure: 1/40s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*40*IMAGE_WIDTH));
             break;
    case 7 : strcpy(button4->text,"Exposure: 1/80s");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*80*IMAGE_WIDTH));
             break;
    case 8 : strcpy(button4->text,"Exposure: 1/100");
             button_update_text(char_buffer_dev, button4, back_buffer);
             SENSOR_CONTROL_SET_INTEG_TIME(SENSOR_FREQ/(3*100*IMAGE_WIDTH));
             break;
    default :
             strcpy(button4->text,"Exposure: 1s");
             #ifdef DEBUG_PRINT
               printf("Unexpected case value : page0_draw->switch-
>exposure_time.\n");
             #endif
  }
  button_draw(pixel_buf_dev, char_buffer_dev, button4, back_buffer);

  // Afficher boutton 'Capture'
  strcpy(button6->text, "Capture");
  button_draw(pixel_buf_dev, char_buffer_dev, button6, back_buffer);

  // Afficher boutton 'Zoom'
  if(zoom_2x == 0)
    strcpy(button8->text,"Zoom 1x");
```

```
  else
    strcpy(button8->text,"Zoom 2x");
  button_draw(pixel_buf_dev, char_buffer_dev, button8, back_buffer);

  // Afficher boutton 'Mode'
  if(video_mode == 0)
    strcpy(button10->text,"Image mode");
  else
    strcpy(button10->text,"Video mode");
  button_draw(pixel_buf_dev, char_buffer_dev, button10, back_buffer);
}
```

ANNEXE 13 – Code source de ‘image.h’

```
/*****************************************************************************
 * Projet        : Capteur d'images
*****************************************************************************
 * Fichier       : image.h
 * Auteur        : Mohamed Sebbar
*****************************************************************************
 * Historique de modification :
 * 2010/05/06 : Création.
*****************************************************************************/

#ifndef IMAGE_H_
#define IMAGE_H_

#include "alt_types.h"
#include "altera_up_avalon_video_pixel_buffer_dma.h"

#define HEIGHT    90
#define WIDTH     120

// structure de données pour sauvegarder les images
typedef struct image_struct
{
  int height;
  int width;
  alt_u16 *red;
  alt_u16 *green;
  alt_u16 *blue;
} image_struct;

void image_draw(alt_up_pixel_buffer_dma_dev *pixel_buffer, image_struct
*image, int x_offset, int y_offset, int backbuffer);
void image_zoom2x(image_struct *image_in, image_struct *image_out);
void image_read(image_struct *image_out);

#endif /*IMAGE_H_*/
```

ANNEXE 14 – Code source de ‘image.c’

```
/*********************************************************************************
 * Projet         : Capteur d'images
**********************************************************************************
 * Fichier        : image.c
 * Auteur         : Mohamed Sebbar
**********************************************************************************
 * Historique de modification :
 * 2010/05/06 : Création.
**********************************************************************************/

#include <stdio.h>
#include "alt_types.h"
#include "io.h"
#include "image.h"

/*********************************************************************************
 * Cette fonction affiche l'image dans la position (x_offset, y_offset)
**********************************************************************************/
void image_draw(alt_up_pixel_buffer_dma_dev *pixel_buffer, image_struct
*image, int x_offset, int y_offset, int backbuffer)
{
  unsigned int addr;
  register unsigned int x, y;
  register unsigned int color;
  alt_u8 red = 0;
  alt_u8 green = 0;
  alt_u8 blue = 0;

  for (x = 0; x<image->width; x++) {
    for (y = 0; y<image->height; y++) {

      // Adresse du pixel en cours
      addr = (((x+x_offset) & pixel_buffer->x_coord_mask) << pixel_buffer-
>x_coord_offset);
      addr |= (((y+y_offset) & pixel_buffer->y_coord_mask) << pixel_buffer-
>y_coord_offset);
      addr |= 0x80000000;

      blue = (alt_u8)(image->blue[y*image->width+x]>>2);
      green = (alt_u8)(image->green[y*image->width+x]>>2);
      red = (alt_u8)(image->red[y*image->width+x]>>2);

      // Couleur du pixel en cours
      color = (red<<16) | (green<<8) | blue;

      if (backbuffer == 0)
        IOWR_32DIRECT(pixel_buffer->buffer_start_address, addr, color);
      else
        IOWR_32DIRECT(pixel_buffer->back_buffer_start_address, addr, color);

    }
  }
}
```

```
/*****************************************************************************
 * Cette fonction applique un zoom 2x sur l'image d'entrée
*****************************************************************************/
void image_zoom2x(image_struct *image_in, image_struct *image_out)
{
  int x, y;
  int x_in, y_in;

  for (x = 0; x<image_out->width; x++) {
    for (y = 0; y<image_out->height; y++) {
      x_in = x>>1;
      y_in = y>>1;
      image_out->red[y*image_out->width+x] = image_in->red[y_in*image_in-
>width+x_in];
      image_out->green[y*image_out->width+x] = image_in->green[y_in*image_in-
>width+x_in];
      image_out->blue[y*image_out->width+x] = image_in->blue[y_in*image_in-
>width+x_in];
    }
  }
}

/*****************************************************************************
 * Cette fonction lit l'image capturée via le bus Avalon
*****************************************************************************/
void image_read(image_struct *image_out)
{
  int x, y;
  alt_u32 data;

  for (y = 0; y<image_out->height; y++) {
    for (x = 0; x<image_out->width; x++) {
      data = IORD(0x80000000 | SENSOR_CONTROL_0_BASE, 0);
      image_out->red[y*image_out->width+x] = (alt_u16)data & 0x03FF;
      image_out->green[y*image_out->width+x] = (alt_u16)(data >> 10) & 0x03FF;
      image_out->blue[y*image_out->width+x] = (alt_u16)(data >> 20) & 0x03FF;
    }
  }
}
```

ANNEXE 15 – Code source de 'button.h'

```
/*********************************************************************************
 * Projet        : Capteur d'images
**********************************************************************************
 * Fichier       : button.h
 * Auteur        : Mohamed Sebbar
**********************************************************************************
 * Historique de modification :
 * 2010/05/06 : Création.
*********************************************************************************/

#ifndef MENU_H_
#define MENU_H_

#include <string.h>
#include "altera_up_avalon_video_pixel_buffer_dma.h"
#include "altera_up_avalon_video_character_buffer_with_dma.h"
#include "alt_types.h"

#define BUTTON_HEIGHT         14
#define BUTTON_WIDTH          130 // espace pour 15 caractères
#define BUTTON_DISTANCE_V     16
#define BUTTON_COLOR          0x000000
#define BUTTON_PRESSED_COLOR  0xE6F092
#define BUTTON_RECT_COLOR     0xFFFFFF
#define BUTTON_TEXT_DEFAULT_X 34
#define MENU_START_X          264
#define MENU_START_Y          44

#define BUTTON1_X_MIN         MENU_START_X
#define BUTTON1_Y_MIN         MENU_START_Y
#define BUTTON1_X_MAX         (MENU_START_X+BUTTON_WIDTH)
#define BUTTON1_Y_MAX         (MENU_START_Y+BUTTON_HEIGHT)

#define BUTTON2_X_MIN         MENU_START_X
#define BUTTON2_Y_MIN         (MENU_START_Y+1*BUTTON_DISTANCE_V)
#define BUTTON2_X_MAX         (MENU_START_X+BUTTON_WIDTH)
#define BUTTON2_Y_MAX         (MENU_START_Y+1*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON3_X_MIN         MENU_START_X
#define BUTTON3_Y_MIN         (MENU_START_Y+2*BUTTON_DISTANCE_V)
#define BUTTON3_X_MAX         (MENU_START_X+BUTTON_WIDTH)
#define BUTTON3_Y_MAX         (MENU_START_Y+2*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON4_X_MIN         MENU_START_X
#define BUTTON4_Y_MIN         (MENU_START_Y+3*BUTTON_DISTANCE_V)
#define BUTTON4_X_MAX         (MENU_START_X+BUTTON_WIDTH)
#define BUTTON4_Y_MAX         (MENU_START_Y+3*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON5_X_MIN         MENU_START_X
#define BUTTON5_Y_MIN         (MENU_START_Y+4*BUTTON_DISTANCE_V)
#define BUTTON5_X_MAX         (MENU_START_X+BUTTON_WIDTH)
#define BUTTON5_Y_MAX         (MENU_START_Y+4*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON6_X_MIN         MENU_START_X
```

```
#define BUTTON6_Y_MIN          (MENU_START_Y+5*BUTTON_DISTANCE_V)
#define BUTTON6_X_MAX          (MENU_START_X+BUTTON_WIDTH)
#define BUTTON6_Y_MAX          (MENU_START_Y+5*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON7_X_MIN          MENU_START_X
#define BUTTON7_Y_MIN          (MENU_START_Y+6*BUTTON_DISTANCE_V)
#define BUTTON7_X_MAX          (MENU_START_X+BUTTON_WIDTH)
#define BUTTON7_Y_MAX          (MENU_START_Y+6*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON8_X_MIN          MENU_START_X
#define BUTTON8_Y_MIN          (MENU_START_Y+7*BUTTON_DISTANCE_V)
#define BUTTON8_X_MAX          (MENU_START_X+BUTTON_WIDTH)
#define BUTTON8_Y_MAX          (MENU_START_Y+7*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON9_X_MIN          MENU_START_X
#define BUTTON9_Y_MIN          (MENU_START_Y+8*BUTTON_DISTANCE_V)
#define BUTTON9_X_MAX          (MENU_START_X+BUTTON_WIDTH)
#define BUTTON9_Y_MAX          (MENU_START_Y+8*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON10_X_MIN          MENU_START_X
#define BUTTON10_Y_MIN          (MENU_START_Y+9*BUTTON_DISTANCE_V)
#define BUTTON10_X_MAX          (MENU_START_X+BUTTON_WIDTH)
#define BUTTON10_Y_MAX
(MENU_START_Y+9*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

#define BUTTON11_X_MIN          MENU_START_X
#define BUTTON11_Y_MIN          (MENU_START_Y+10*BUTTON_DISTANCE_V)
#define BUTTON11_X_MAX          (MENU_START_X+BUTTON_WIDTH)
#define BUTTON11_Y_MAX
(MENU_START_Y+10*BUTTON_DISTANCE_V+BUTTON_HEIGHT)

typedef struct button_struct
{
  int position_x;
  int position_y;
  int rectangle_color;
  char text[50];
  alt_u8  pressed;
} button_struct;

void button_draw(alt_up_pixel_buffer_dma_dev *pixel_buffer,
alt_up_char_buffer_dev *char_buffer, button_struct* button, int backbuffer);
void button_update_text(alt_up_char_buffer_dev *char_buffer, button_struct*
button, int backbuffer);

#endif /*MENU_H_*/
```

ANNEXE 16 – Code source de 'button.c'

```
/*******************************************************************************
 * Projet        : Capteur d'images
*******************************************************************************
 * Fichier       : button.c
 * Auteur        : Mohamed Sebbar
*******************************************************************************
 * Historique de modification :
 * 2010/05/06 : Création.
*******************************************************************************/

#include "button.h"

/*******************************************************************************
 * Cette fonction affiche le bouton passé en paramètre
*******************************************************************************/
void button_draw(alt_up_pixel_buffer_dma_dev *pixel_buffer,
alt_up_char_buffer_dev *char_buffer, button_struct* button, int backbuffer)
{
  alt_up_pixel_buffer_dma_draw_rectangle(pixel_buffer, button->position_x,
button->position_y, button->position_x+BUTTON_WIDTH, button-
>position_y+BUTTON_HEIGHT, button->rectangle_color, backbuffer);
  alt_up_char_buffer_string(char_buffer, button->text, BUTTON_TEXT_DEFAULT_X,
(int)(button->position_y+11)/8);
  if(button->pressed)
    alt_up_pixel_buffer_dma_draw_box(pixel_buffer, button->position_x+1,
button->position_y+1, button->position_x+BUTTON_WIDTH-1, button-
>position_y+BUTTON_HEIGHT-1, BUTTON_PRESSED_COLOR, backbuffer);
  else
    alt_up_pixel_buffer_dma_draw_box(pixel_buffer, button->position_x+1,
button->position_y+1, button->position_x+BUTTON_WIDTH-1, button-
>position_y+BUTTON_HEIGHT-1, BUTTON_COLOR, backbuffer);
}

/*******************************************************************************
 * Cette fonction met à jour le text du bouton
*******************************************************************************/
void button_update_text(alt_up_char_buffer_dev *char_buffer, button_struct*
button, int backbuffer)
{
  char pchar[50];

   // Effacer l'ancien text
  strcpy(pchar, "               ");
  alt_up_char_buffer_string(char_buffer, pchar, BUTTON_TEXT_DEFAULT_X,
(int)(button->position_y+11)/8);

  // Afficher le nouveau text
  alt_up_char_buffer_string(char_buffer, button->text, BUTTON_TEXT_DEFAULT_X,
(int)(button->position_y+11)/8);
}
```

ANNEXE 17 – Code source de 'sensor_control.h'

```
/************************************************************************
 * Projet        : Capteur d'images
************************************************************************
 * Fichier       : sensor_control.h
 * Auteur        : Mohamed Sebbar
************************************************************************
 * Historique de modification :
 * 2010/05/06 : Création.
************************************************************************/

#ifndef SENSOR_CONTROL_H_
#define SENSOR_CONTROL_H_

#include <math.h>
#include <unistd.h>
#include "alt_types.h"
#include "system.h"
#include "io.h"

#define SENSOR_FREQ         20000000
#define IMAGE_WIDTH         120
#define SHIFT_LENGTH        10

#define SENSOR_CONTROL_CAPTURE() \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 0, 1 ); \
  usleep(200); \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 0, 0 )

#define SENSOR_CONTROL_SET_RESET_TIME(reset_time) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 1, reset_time ); \
  usleep(200)

#define SENSOR_CONTROL_SET_INTEG_TIME(integration_time) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 2, integration_time ); \
  usleep(200)

#define SENSOR_CONTROL_SET_VGRWSEL_WIDTH(vgrwsel_width) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 3, vgrwsel_width ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A11(a_11) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 4,
(alt_32)(a_11*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A12(a_12) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 5,
(alt_32)(a_12*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A13(a_13) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 6,
(alt_32)(a_13*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)
```

```
#define SENSOR_CONTROL_SET_MAT_A21(a_21) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 7,
(alt_32)(a_21*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A22(a_22) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 8,
(alt_32)(a_22*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A23(a_23) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 9,
(alt_32)(a_23*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A31(a_31) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 10,
(alt_32)(a_31*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A32(a_32) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 11,
(alt_32)(a_32*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#define SENSOR_CONTROL_SET_MAT_A33(a_33) \
  IOWR( 0x80000000 | SENSOR_CONTROL_0_BASE, 12,
(alt_32)(a_33*pow(2,SHIFT_LENGTH)) ); \
  usleep(200)

#endif /*SENSOR_CONTROL_H_*/
```