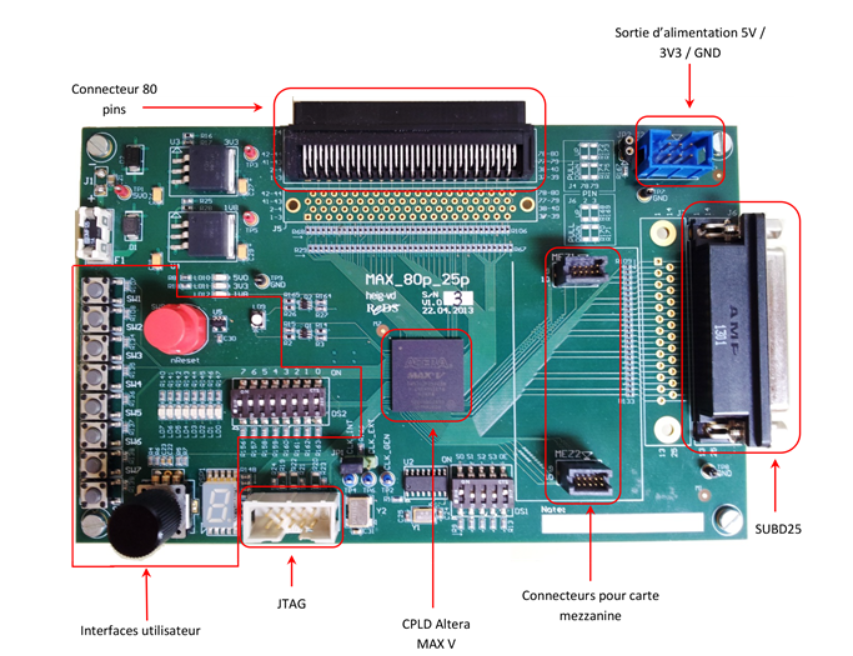
Commande PWM pour LED RGB

[CONCEPTION DE SYSTÈMES NUMÉRIQUES (CSN)](https://cyberlearn.hes-so.ch/course/view.php?id=14116#section-1) 

Auteur : Spinelli Isaia et

Gerardi tommy fdp

Prof : [Etienne](https://cyberlearn.hes-so.ch/user/view.php?id=104149&course=1) Messerli

Ing : Sébastien Masle

Date : 14.11.2019

Salle : A09

Classe : CSN

Table des matières

[Objectifs - 3 -](#_Toc24633404)

[Principe physique - 3 -](#_Toc24633405)

[Applications - 3 -](#_Toc24633406)

[Contrôle de la luminosité - 3 -](#_Toc24633407)

[Principe de fonctionnement d’un PWM - 3 -](#_Toc24633408)

[Spécification du système de commande d’une Led RGB - 4 -](#_Toc24633409)

[Entrées/sorties du système - 4 -](#_Toc24633410)

[Description du fonctionnement - 4 -](#_Toc24633411)

[Commentaire - 4 -](#_Toc24633412)

[Décomposition du système - 5 -](#_Toc24633413)

[Réalisation des 3 générateurs PWM (partie 1) - 6 -](#_Toc24633414)

[Analyse du fonctionnement - 6 -](#_Toc24633415)

[Décomposition optimisée - 6 -](#_Toc24633416)

[Schéma bloc - 6 -](#_Toc24633417)

[Tables des fonctions synchrones - 6 -](#_Toc24633418)

[Block1 - 6 -](#_Toc24633419)

[Block 2 - 6 -](#_Toc24633420)

[Descriptions VHDL - 6 -](#_Toc24633421)

[Block 1 - 6 -](#_Toc24633422)

[Block 2 - 6 -](#_Toc24633423)

[Bock 1 - 6 -](#_Toc24633424)

[Bock 2 - 6 -](#_Toc24633425)

[Regroupement des blocs - 6 -](#_Toc24633426)

[Vérification du fonctionnement - 6 -](#_Toc24633427)

[Synthétisation - 6 -](#_Toc24633428)

[Quantité de logique - 6 -](#_Toc24633429)

[Réalisation du module de conversion (partie 2) - 7 -](#_Toc24633430)

[Analyse du fonctionnement - 7 -](#_Toc24633431)

[Description VHDL - 7 -](#_Toc24633432)

[Synthétisation - 7 -](#_Toc24633433)

[Quantité de logique - 7 -](#_Toc24633434)

[Contrôle des niveaux - 7 -](#_Toc24633435)

[Réalisation - 7 -](#_Toc24633436)

[Description VHDL - 8 -](#_Toc24633437)

[Synthétisation - 8 -](#_Toc24633438)

[Vérification du système complet et intégration - 9 -](#_Toc24633439)

[Vérification du système « led\_rgb\_top.vhd » - 9 -](#_Toc24633440)

[Synthétisation - 9 -](#_Toc24633441)

[Quantité logique - 9 -](#_Toc24633442)

[Intégration - 9 -](#_Toc24633443)

[Test du système - 9 -](#_Toc24633444)

[Système de commande - 9 -](#_Toc24633445)

[Fréquence de fonctionnement du PWM - 9 -](#_Toc24633446)

[Validation - 9 -](#_Toc24633447)

# Objectifs

Nous souhaitons réaliser une commande d’une LED-RGB permettant de contrôler l’intensité et la couleur via des signaux binaires. Nous allons utiliser le principe du PWM pour contrôler l’intensité.

Le principe de la modulation de largeur d’impulsion PWM (Pulse Width Modulation) est de contrôler la valeur moyenne d'un signal en modulant la largeur d'activation d'un signal binaire ('1' / '0') pendant une période définie. Dans la partie test, nous pourrons expérimenter le principe du mélange additif des couleurs « RGB »

## Principe physique

L’affichage couleur d’un point lumineux est basé sur le principe du mélange additif des couleurs primaires ( Rouge « R » , Vert « G » , Bleu « B » ). Soit X : une lumière de couleur quelconque, alors :

X = Ir . R + Ig . G + Ib . B

Où ( Ir, Ig, Ib) sont les intensités lumineuses de chacune des lumières de couleurs primaires (RGB)

## Applications

* Un écran couleur, est constitué des milliers de points lumineux. L’intensité et la couleur de chaque point sont contrôlées par le niveau des signaux vidéo (RGB) à afficher.
* L’affichage LCD d’un téléphone mobile de la nouvelle génération est rétro-illuminé par 3 LED (RGB) afin de pouvoir régler la couleur de cette lumière (Back-Light).
* Des éléments décoratifs sont actuellement en vente (LED Tower, RGB Led projector)

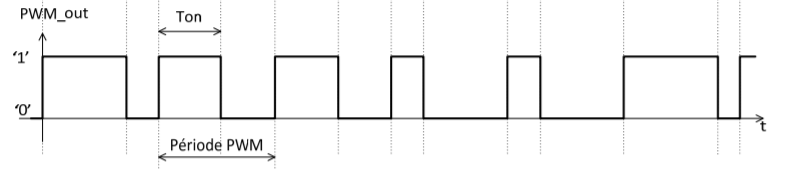
## Contrôle de la luminosité

Pour contrôler ces 3 LED (RGB), afin de pouvoir définir la couleur voulue, deux méthodes sont possibles :

* **Analogique** à l’aide de 3 signaux analogiques (r, g, b), le cas des écrans à Tube à Rayon Cathodique (CRT)
* **Numérique** : Solution robuste, simple, et à rendement énergétique très élevé !

## Principe de fonctionnement d’un PWM

Un PWM permet de contrôler via un signal binaire une information analogique. La modulation de largeur d'impulsions (PWM) est une technique couramment utilisée pour synthétiser des signaux continus à l'aide d'un signal tout ou rien numérique ('1' et '0'). La variation de la tension continue est obtenue en modifiant le rapport cyclique entre la durée à '1' et à '0' du signal digital.



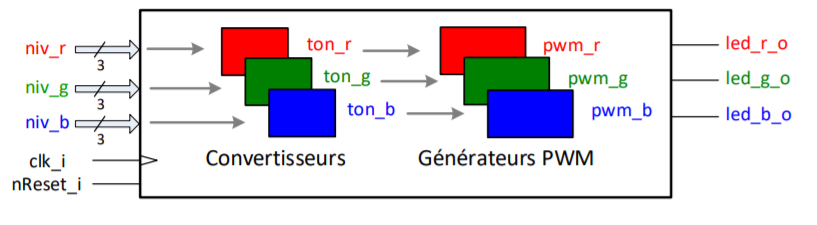
Nous pouvons définir 2 paramètres qui spécifient un générateur PWM, soit :

* La fréquence du PWM (période PWM).
* La résolution du PWM, soit le nombre de bits du signal de commande du niveau (Ton).

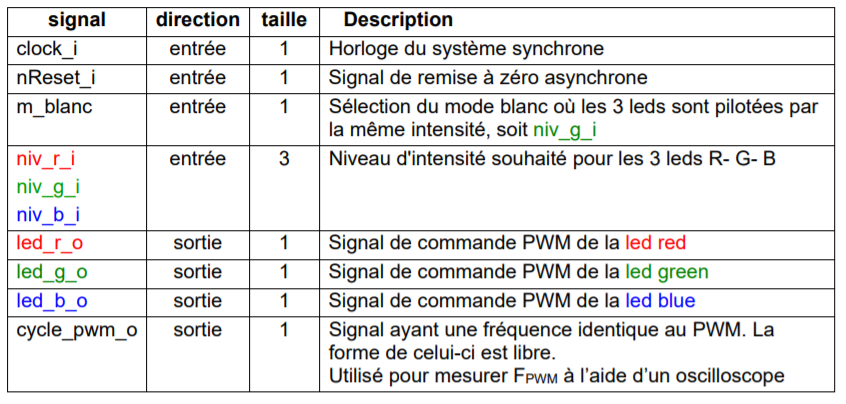
# Spécification du système de commande d’une Led RGB

Le système comprend 2 blocs, soit les convertisseurs et les PWM. L'intensité des leds n'étant pas linéaire en fonction de ton, nous allons utiliser des convertisseurs pour obtenir 8 niveaux d'intensité régulièrement répartit. Le second bloc comprend les 3 générateurs PWM, soit un pour chaque couleur.

Voici le schéma bloc du système :



## Entrées/sorties du système



## Description du fonctionnement

* L’intensité lumineuse ( niv\_r, niv\_g, niv\_b) de chaque couleur est définie par une valeur sur 3 bits de 0 à 7.
* Les convertisseurs permettent de détermine la valeur de ton pour les 8 niveaux d'intensité. Une table sera utilisée pour convertir les niveaux d’intensité (codée sur 3 bits) en une largeur d’impulsion codé sur 8 bits (table donnée ci-après).
* Les trois sorties led\_r, led\_g et led\_b sont commandées par un signal modulé en largeur d'impulsion (PWM) codé sur 8 bits. La largeur de l'impulsion peut varier de 0 à 255.

### Commentaire

Notre impression de luminosité n’est pas linéaire par rapport à la tension moyenne appliquée à une Led. Nous allons donc introduire, pour la réalisation, un convertisseur qui va permettre de fournir une suite de largeur d'impulsion, Ton pour le PWM, non linéaire. Cela permettra d'attribuer 8 valeurs de largeur d'impulsion permettant une progression plus ou moins linéaire de l'impression de luminosité de la LED.

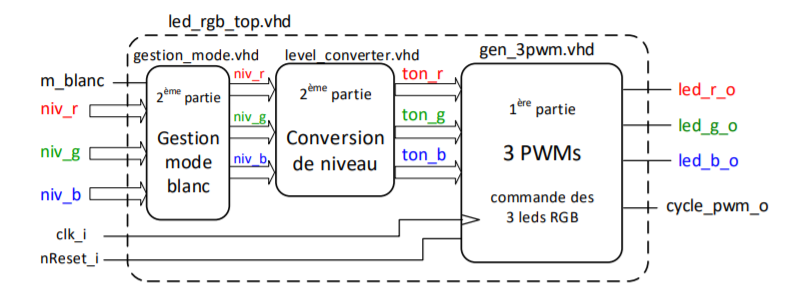
## Décomposition du système

Pour la réalisation pratique, nous allons décomposer le système en deux modules, soit :

* Un module de conversion des niveaux d'intensité en une valeur de Ton appliquée à la Led via le PWM.
* Un module composé de 3 PWMs.

Pour la réalisation, nous allons travailler en deux parties, soit :

* 1ère partie : conception et réalisation du module comprenant trois PWMs, un par Led (gen\_3pwm.vhd).
* 2ème partie : Conception et réalisation du module de conversion des niveaux d'intensité en une valeur de Ton et gestion du mode « blanc ».



# Réalisation des 3 générateurs PWM (partie 1)

Voici les spécifications des générateurs PWM à réaliser, soit :

• PWM à 256 niveaux, le signal de consigne, temps d'activation Ton, est sur 8 bits.

• Période du PWM est de 1.024 ms soit une fréquence proche de 1 KHz (976 Hz).

Nous disposons d'une fréquence système à 1 MHz. Nous devons optimiser la quantité de logique nécessaire à notre solution. Celle-ci doit être optimisée dans notre cas, soit la réalisation de 3 PWMs fonctionnant à la même fréquence mais avec des niveaux différents.

## Analyse du fonctionnement

### Décomposition optimisée

On commence par décomposer en un seul générateur PWM :

|  |  |
| --- | --- |
| **Entrées** | **Sorties** |
| Ton\_x | Led\_x\_o |
| Clk\_i |  |
| Nreset\_i |  |

**Led\_x\_o** est la sortie désirée, un signal PWM modulable en fonction de Ton\_x associé.

**La clock** est de 1Mhz, donc il nous fait 1024 coups de clock pour avoir un temps de 1.024ms. Il nous faut donc un compteur 10 bits (1024). A chaque boucle de comptage notre période du signale PWM recommencera.

Ensuite, afin de gérer la sortie en fonction du **Ton**, il nous faut un comparateur pour mettre la sortie à 1 le temps que Ton est plus grand que notre compteur divisé par 4 (shift de 2 à droite). Une fois que le compteur shifté est plus grand que la valeur de Ton, la sortie passera à 0.

Si le **nReset** est à 0, il faudra reset le compteur et mettre la sortie à 0.

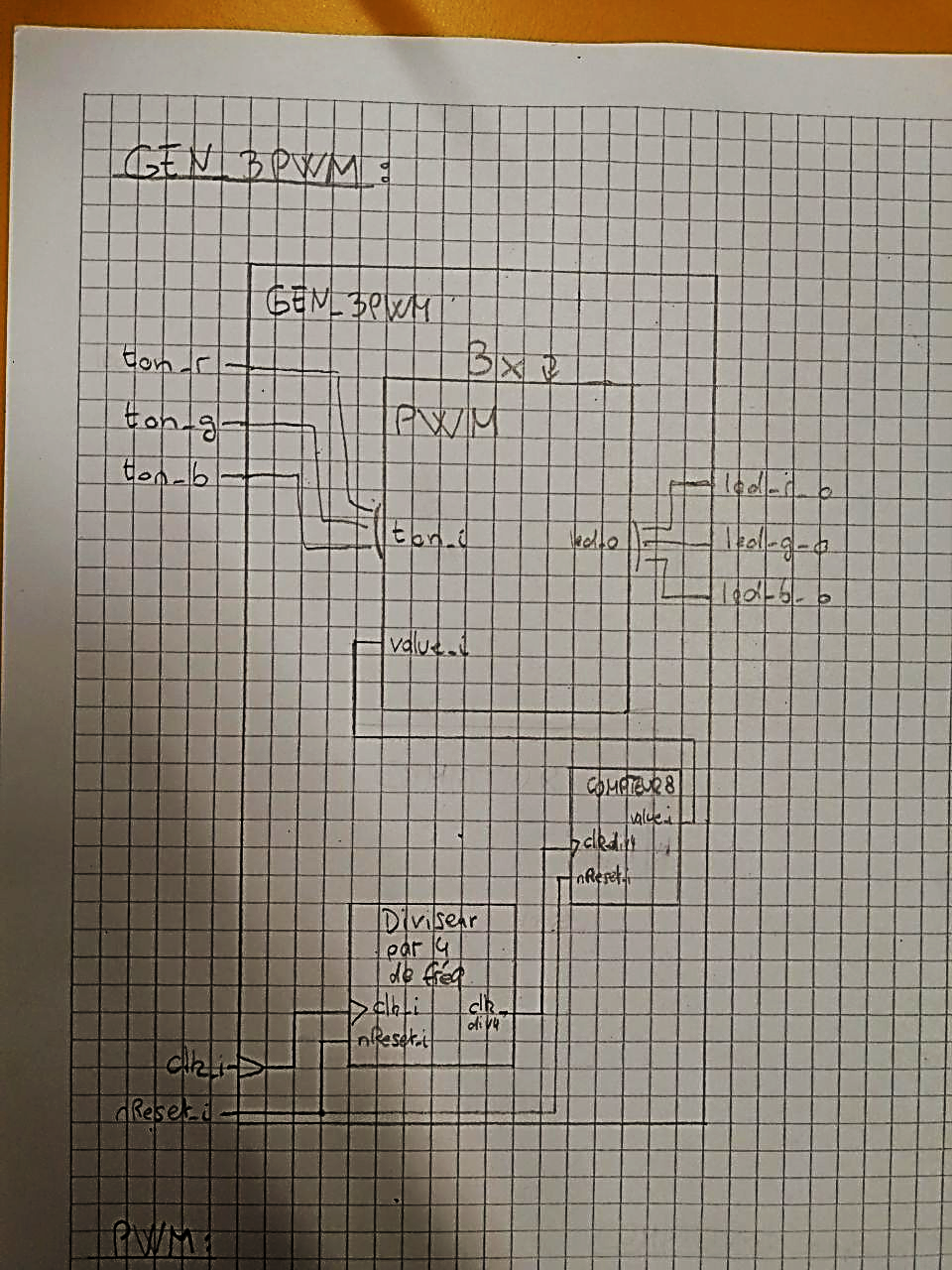
Il serait aussi possible d’utiliser un compteur 2 bits suivie d’un compteur 8 bits. Ceci nous permettrait de compter directement de 0 à 256. Cependant, nous préférons utiliser un seul compteur 10 bits car nous pensons que cela est plus optimisé.

Après mur réflexion, nous avons pensé a remplacé le compteur 2 bits avec un simple diviseur de clock par 4 étant donné que nous n’avons pas besoin de l’information du compteur. Il suffira d’enchainer deux bascule T afin de diviser la clock par 4. De ce fait, nous utilisons un compteur 8 bits et deux bascules T, nous pensons que cette solution est plus optimisée que les précédentes.

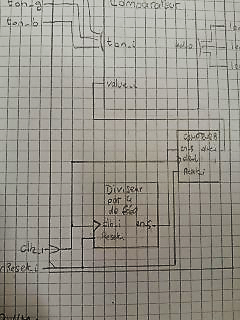
Au début nous avons directement connecté la sortie du diviseur de clock par 4 au compteur. Malheureusement, nous avons remarqué seulement après que cette solution n’est pas full synchrone car le compteur 8 bits doit être connecté à la clock du système. Donc, nous avons ajouté un enable généré par le diviseur par 4 afin d’être full synchrone.

## Schéma bloc

Voici notre schéma bloc général avant les corrections :



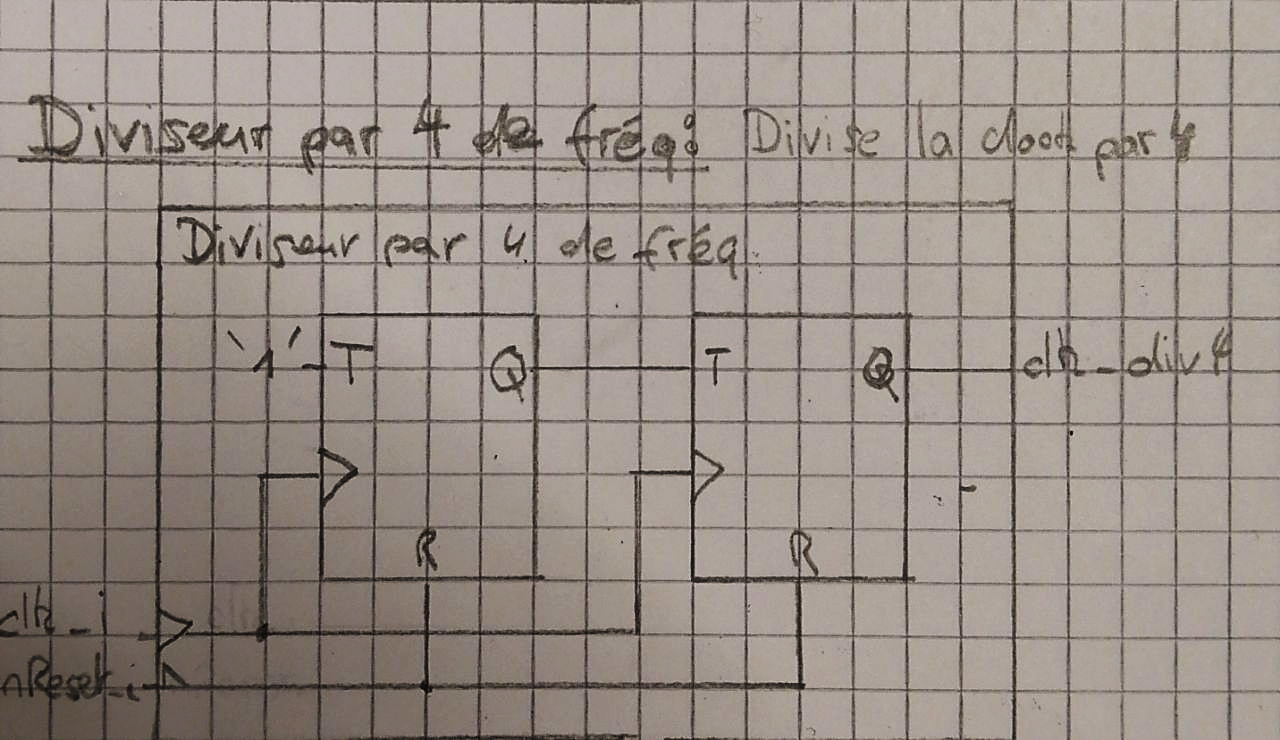
On peut voir que ce n’est pas full synchrone car le compteur 8 bits est synchrone sur le clock divisé par 4 généré par le diviseur de clock. Finalement, nous avons généré un enable qui permet de compter au bon moment pour synchrone avec le clock du système.



Chaque bloc est détaillé ci-dessous.

### Diviseur

Le but de ce block est de diviser la clock du système (1Mhz) par 4 avec deux bascules T (T=’1’) afin d’avoir un fréquence 256 fois plus élevé que la fréquence du signal PWM souhaitée. (976 Hz)



1

**0**

#### Table des fonctions synchrones

Voici la table des fonctions synchrone d’une simple bascule T (T=’1’). Afin de diviser par 4 la clock source, nous enchainerons 2 bascules comme ceci.

|  |  |  |  |
| --- | --- | --- | --- |
| **T** | **Reg\_pres** | **Ref\_fut** | **Fonction** |
| 1 | / | not(reg\_pres) | Inverse (Diviseur de clock par 2) |
| 0 | / | = reg\_pres | Maintiens |

Après avoir remarqué que notre système n’était pas full synchrone, nous avons ajouté une porte AND avec Q0 et Q1 afin de générer un signal en\_s qui permettra au compteur 8 bits de compter.

Finalement, en ouvrant les fichiers fournis, nous avons remarqué qu’il fallait faire des systèmes génériques. De ce fait, le principe de bascule T n’est pas facile à rendre générique donc nous avons décidé de revenir sûr la première solution en utilisant un simple compteur 2 bits afin d’être générique simplement. Ce compteur généra une sorite « en » qui permettra au compteur de compté un clock sur N (4).

#### Description VHDL

### Compteur

Ce bloc est un simple compteur 8 bits synchrone sur la clock du système. Il compte une fois sur 4 du coup de clock grâce au enable généré avec notre diviseur présenté préférablement. Ce compteur est nécessaire afin d’avoir une résolution de 256 pas sur le signal PWM. La valeur du compteur 8 bits sera utilisé afin de contrôler la largeur d’impulsion du signal PWM en fonction de Ton.

#### Table des fonctions synchrones

|  |  |  |  |
| --- | --- | --- | --- |
| **En\_s** | **Cpt\_pres** | **Cpt\_fut** | **Fonction** |
| 1 | / | = cpt\_pres + 1 | Comptage |
| 0 | / | = cpt\_pres | Maintien |

Nous avons commencé par connecter l’entrée clk du compteur avec la clock du diviseur 4. Après avoir remarqué que cette solution n’était pas full synchrone, nous avons ajouté une entrée « en » afin de compter seulement un fois sur 4 clock. Maintenant, notre entré clk est bien connecté avec la clock du système.

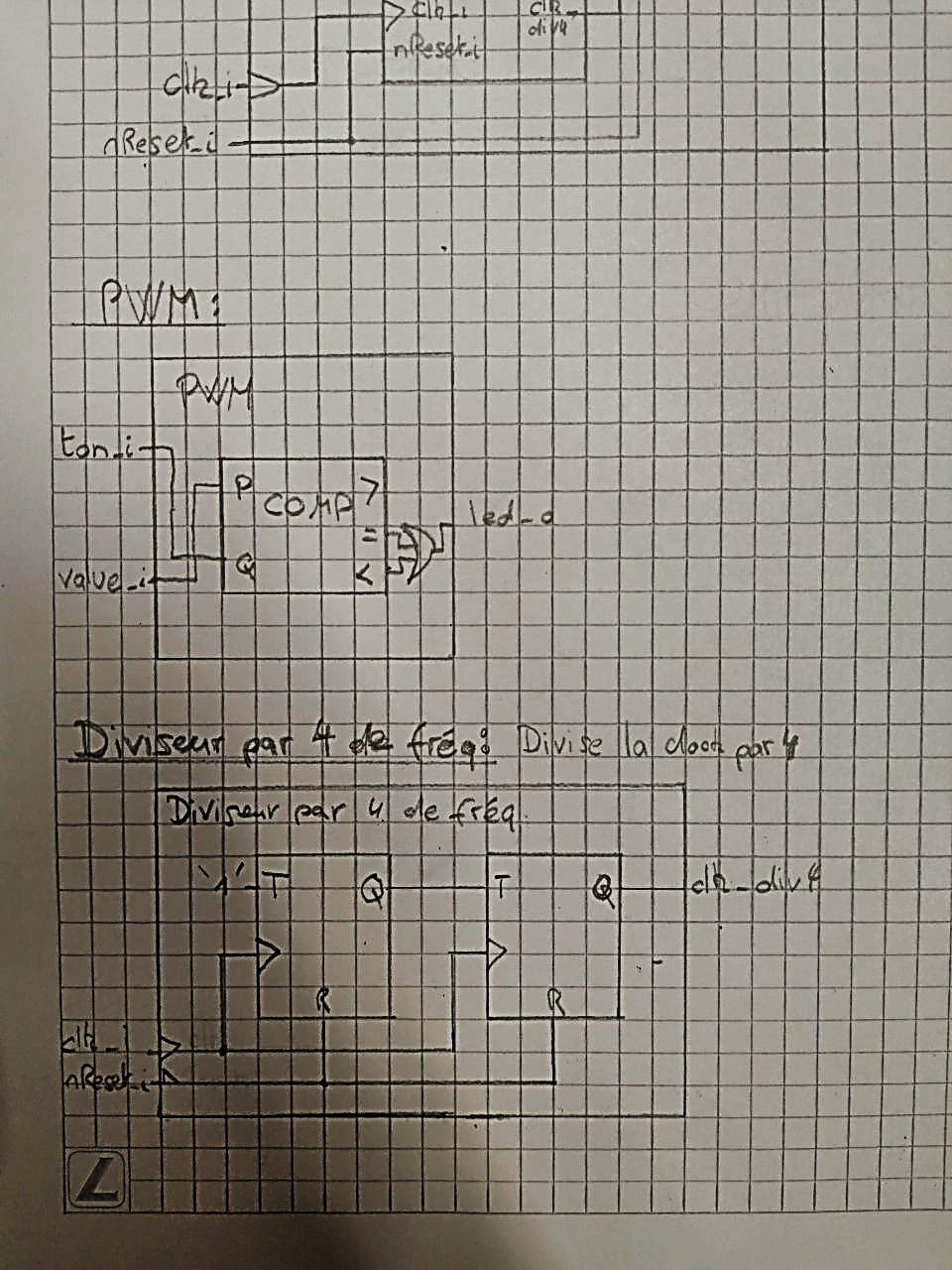
Finalement, nous devons faire un compteur générique. Par défaut nous avons mis un compteur 8 bits afin d’avoir une résolution de 256 comme demandé.

#### Description VHDL

### Comparateurs

Ce bloc permet de générer le signal PWM ainsi que réguler la largeur d’impulsion (Ton) du signal PWM en fonction de l’entrée Ton. Etant donné qu’il faut générer 3 signales PWM, nous avons besoin de 3 blocs comme celui-ci.

Ce bloc est uniquement composé d’un comparateur 8 bits. En effet, en comparant la valeur du compteur 8bits (value\_i) et la valeur Ton\_i donnée, le compteur modulera la largeur d’impulsion (Ton) du signal. Tant que la valeur du compteur 8 bits est égal ou plus petit que la valeur Ton\_i, la sortie reste à 1. Une fois que la valeur est supérieure, le sorite passe à 0.



Après avoir remarqué qu’il fallait rendre notre système générique nous avons juste fait que le comparateur soit générique avec une valeur par défaut de 8 pour être compatible avec la résolution de 256.

#### Table des fonctions synchrones

Ces blocs sont uniquement composés d’un comparateur qui est un simple composant sans clock donc non synchrone. Donc, il n’y a pas de table des fonctions synchrones à faire.

#### Description VHDL

## Regroupement des blocs

## Vérification du fonctionnement

À l'aide d'une simulation manuelle interactive avec console\_sim.vhd, la console REDS et le script « run\_ctrl\_gen\_3pwm\_sim.tcl » fourni, nous avons pu vérifier le fonctionnement du sous-système « gen\_3pwm.vhd » en testant plusieurs choses :

## Synthétisation

### Quantité de logique

# Réalisation du module de conversion (partie 2)

Pour cette partie, il y a 2 blocs à réaliser, soit :

* Conception et réalisation du module de conversion des niveaux d'intensité en une valeur de Ton. La table de conversion est donnée ci-dessous.
* Réalisation d’un bloc pour la gestion du mode blanc, soit :

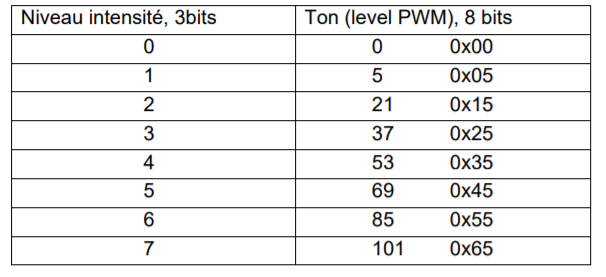
**m\_blanc actif (‘1’)**

Les trois leds sont pilotées par le niveau d’intensité vert (niv\_g). Ainsi la led RGB a une couleur blanche.

**m\_blanc inactif (‘0’)**

Les trois leds sont pilotées par leur niveau d’intensité respectif. Ainsi la led RGB a toute la palette de couleur possible.

Table de conversion entre le niveau d'intensité et le Ton du PWM pour la led :



## Analyse du fonctionnement

Ensuite, nous avons analyser le fonctionnement pour la conversion du niveau. Comme mentionné précédemment, la conversion est fixe. La table des conversions est plus haut. Il suffira d’utiliser des « when – else » comme un switch afin de faire la conversion.

## Description VHDL

## Synthétisation

### Quantité de logique

## Contrôle des niveaux

### Réalisation

Nous avons analysé le fonctionnement de la gestion du mode blanc :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entrées** | | | | **Sorties** | | |
| **M\_blanc** | **Niv\_r** | **Niv\_g** | **Niv\_b** | **Niv\_r** | **Niv\_g** | **Niv\_b** |
| 0 | R | G | B | R | G | B |
| 1 | R | G | B | G | G | G |

Donc si m\_blanc est actif, toutes les sorties prennent la valeur de niv\_g. Sinon, chaque sortie est pilotée par son niveau d’intensité respectif.

Nous pensons utiliser des simples « when-else » pour chaque sortie afin de tester l’entrée m\_blanc et d’affecter la bonne entrée à la sorit en fonction de la table ci-dessus.

### Description VHDL

### Synthétisation

# Vérification du système complet et intégration

## Vérification du système « led\_rgb\_top.vhd »

Nous avons testé le bon fonctionnement du fichier « led\_rgb\_top.vhd » avec une simulation automatique grâce au script « run\_led\_rgb\_top\_tb.tcl » fourni.

Voici le résultat :

On peut voir qu’il n’y a aucune erreur dans notre description.

## Synthétisation

Nous avons fait la synthèse et le placement routage de notre solution pour le led\_rgb\_top.vhd à l’aide du logiciel Quartus. Nous avons utilisé le top spécifique pour la carte Max-V fourni dans le projet, soit : maxv\_top.vhd.

### Quantité logique

Nous pouvons voir ici la quantité de logique de composant led\_rgb\_top qui est de XX sans la logique du maxv\_top.

## Intégration

Ensuite, nous avons intégré notre solution en programmant une maquette MaxV 80p-25p avec le fichier \*.pof généré lors du placement et routage.

## Test du système

### Système de commande

### Fréquence de fonctionnement du PWM

## Validation

Notre montage à bien été validé le … par … .