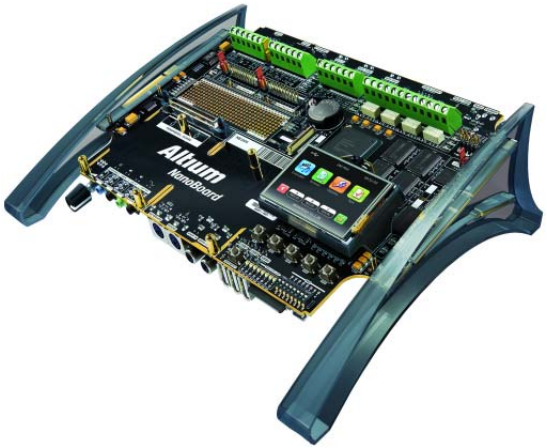
Mini Projet : **Génération d'une trame DMX512 par FPGA sous environnement ALTIUM** Durée estimée : 6 séances de 4h = 24h

Sommaire :






Annexes :



Les TPs ont été réalisés avec un PC équipé de la manière suivante

Configuration logicielle :

- Windowsw Seven Pro 64 bits.
- Altium Designer Version 13.2.5 (build 28368)
- Quartus II Web Edition 12.0 sp2

Mini Projet : **Génération d'une trame DMX512 par FPGA sous environnement ALTIUM** Durée estimée : 6 séances de 4h = 24h

**1 Objectif du projet :**

⇒ Réaliser l'acquisition et le traitement de consignes pour générer une trame DMX par un FPGA implanté sur la Nanoboard 3000.

Le support de l'étude est une console DMX portable de régie lumière pilotant des projecteurs à Led – voir les annexes à partir de la page 5

**2 Pré-requis :**

⇒ Savoir écrire un programme en langage C avec passage de paramètres, maîtriser l'algorithmique.
⇒ TPs de formation sur ALTIUM (spécifiés pour chaque activité)

**3 Nombre d'étudiants menant ce projet :**

⇒ 1 étudiant

**4 Cahier des charges du mini projet :**

⇒ **Mettre en œuvre sur la carte de prototypage rapide « Nano Board 3000 » pour générer une trame DMX.**

Vous développerez, dans un premier temps, sous l'environnement ALTIUM la programmation du FPGA afin d'y implanter un microprocesseur **TSK 3000** (mise en œuvre d'**IP**, **I**ntellectual **P**roperties).
Dans un deuxième temps vous programmerez, en langage C, le microprocesseur afin de réaliser l'acquisition des grandeurs analogiques images des consignes Rouge, Vert, Bleu et Dimmer pour ensuite les envoyer vers la liaison série Rs485 en respectant le protocole DMX.

⇒ Caractéristiques de l'acquisition numérique :
- CAN piloté par bus SPI à 2 MHz
- résolution du CAN à 8bits.

⇒ Entrée analogique
- Niveau d'entrée variant de 0 à 3.3V

⇒ Sortie numérique sur les bus RS485 intégré à la Nanoboard 3000

**5 Outil de test du mini projet :**

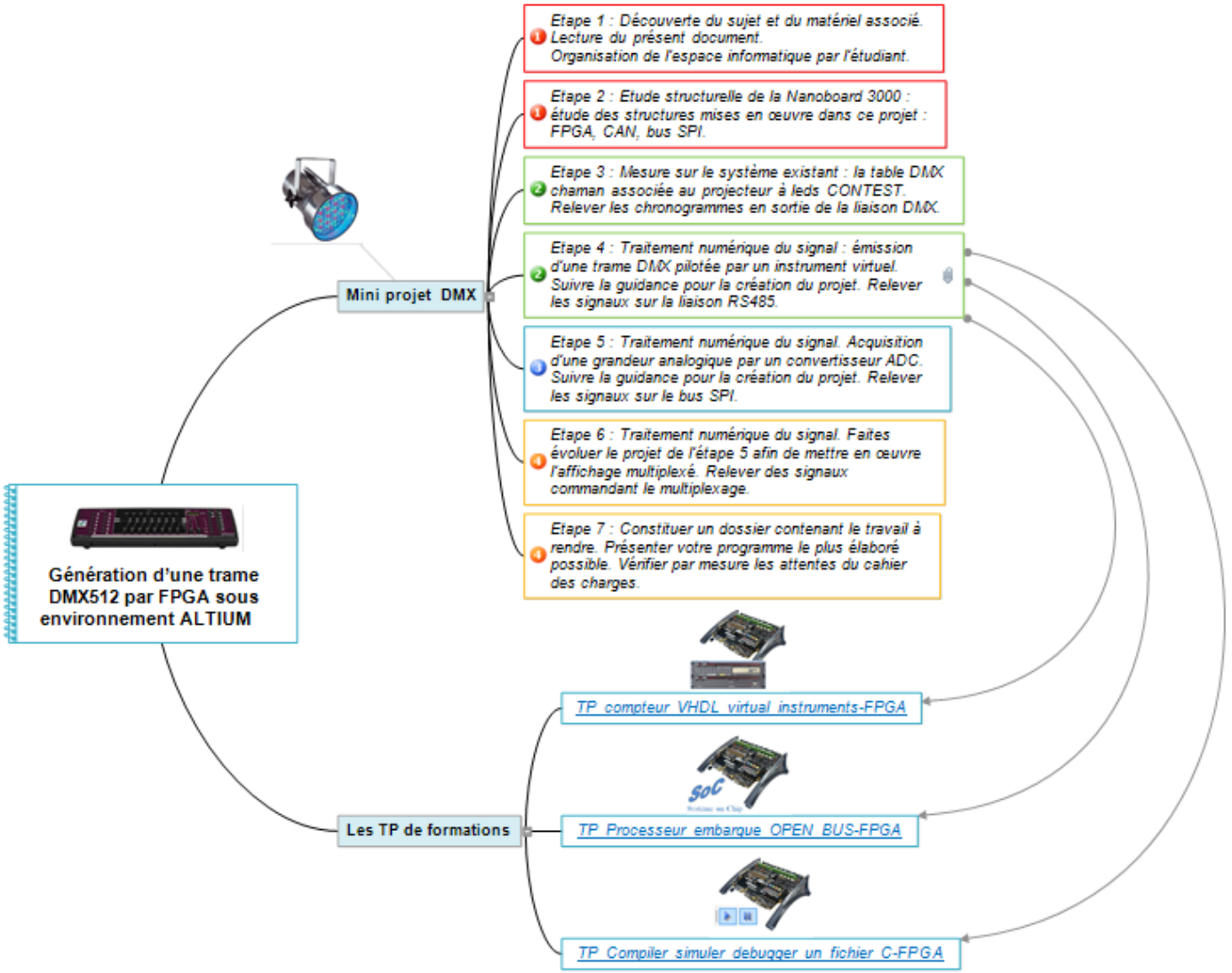
⇒ **Ecrire sous ALTIUM DESIGNER une application en langage C pour générer une trame DMX**
⇒ Vous développerez un banc de mesure permettant de tester votre application. Voir le synoptique Annexe 5.
⇒ Ce banc de mesure incorporera un PC équipé de **Windows 7 64 bits** + Altium Designer v13.2.5 + Quartus II Web Edition v 12.0 + Nanoboard 3000 qui permettra :
- de générer vers la liaison série RS485 une trame respectant le protocole DMX
- de numériser une grandeur analogique qui sera ensuite envoyée vers la liaison DMX
- de piloter un affichage multiplexé

⇒ Vous mettrez en place des grapheurs permettant de visualiser les signaux électriques avant et après le traitement.

**6 Ressources fournies :**

⇒ Activité 1 : Analyse structurelle des fonctions de la carte Nanoboard mises en œuvre durant ce projet.
⇒ Protocole DMX
⇒ La notice de la table DMX Chaman.
⇒ La notice du projecteur à leds PAR 38 RGB
⇒ Les annexes A1 à A9 du présent document.

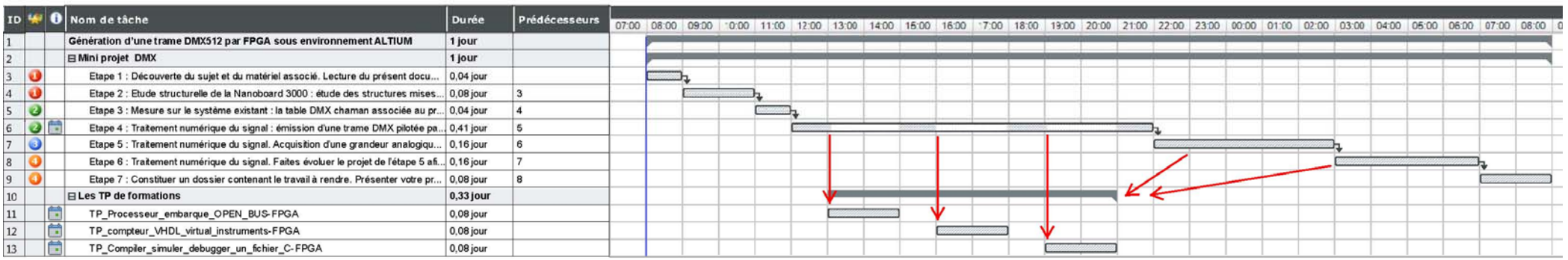
## 7- Parcours de formation du mini projet



Mini Projet : **Génération d'une trame DMX512 par FPGA sous environnement ALTIUM** Durée estimée : 6 séances de 4h = 24h

**Diagramme de Gantt : planification du travail sur les 6 séances de 4h ,:**

| ID | Nom de tâche | Durée | Prédécesseurs |
|---|---|---|---|
| 1 | **Génération d'une trame DMX512 par FPGA sous environnement ALTIUM** | **1 jour** | |
| 2 | **Mini projet DMX** | **1 jour** | |
| 3 | Etape 1 : Découverte du sujet et du matériel associé. Lecture du présent docu... | 0,04 jour | |
| 4 | Etape 2 : Etude structurelle de la Nanoboard 3000 : étude des structures mises... | 0,08 jour | 3 |
| 5 | Etape 3 : Mesure sur le système existant : la table DMX chaman associée au pr... | 0,04 jour | 4 |
| 6 | Etape 4 : Traitement numérique du signal : émission d'une trame DMX pilotée pa... | 0,41 jour | 5 |
| 7 | Etape 5 : Traitement numérique du signal. Acquisition d'une grandeur analogiqu... | 0,16 jour | 6 |
| 8 | Etape 6 : Traitement numérique du signal. Faites évoluer le projet de l'étape 5 afi... | 0,16 jour | 7 |
| 9 | Etape 7 : Constituer un dossier contenant le travail à rendre. Présenter votre pr... | 0,08 jour | 8 |
| 10 | **Les TP de formations** | **0,33 jour** | |
| 11 | TP_Processeur_embarque_OPEN_BUS-FPGA | 0,08 jour | |
| 12 | TP_compteur_VHDL_virtual_instruments-FPGA | 0,08 jour | |
| 13 | TP_Compiler_simuler_debugger_un_fichier_C-FPGA | 0,08 jour | |




*Repère des documents de travail : Activités 1 à 4*

*Les TP de formations s'insèrent dans l'étude du mini projet*

**8 Travail à rendre :**

1 : Trois projets ALTIUM sous forme numérique :

⇒ Projet ALTIUM 1 : envoi d'une trame DMX à partir de consignes d'un instrument virtuel

⇒ Projet ALTIUM 2 : acquisition d'une grandeur analogique

⇒ Projet ALTIUM 3 : affichage multiplexé de la grandeur analogique

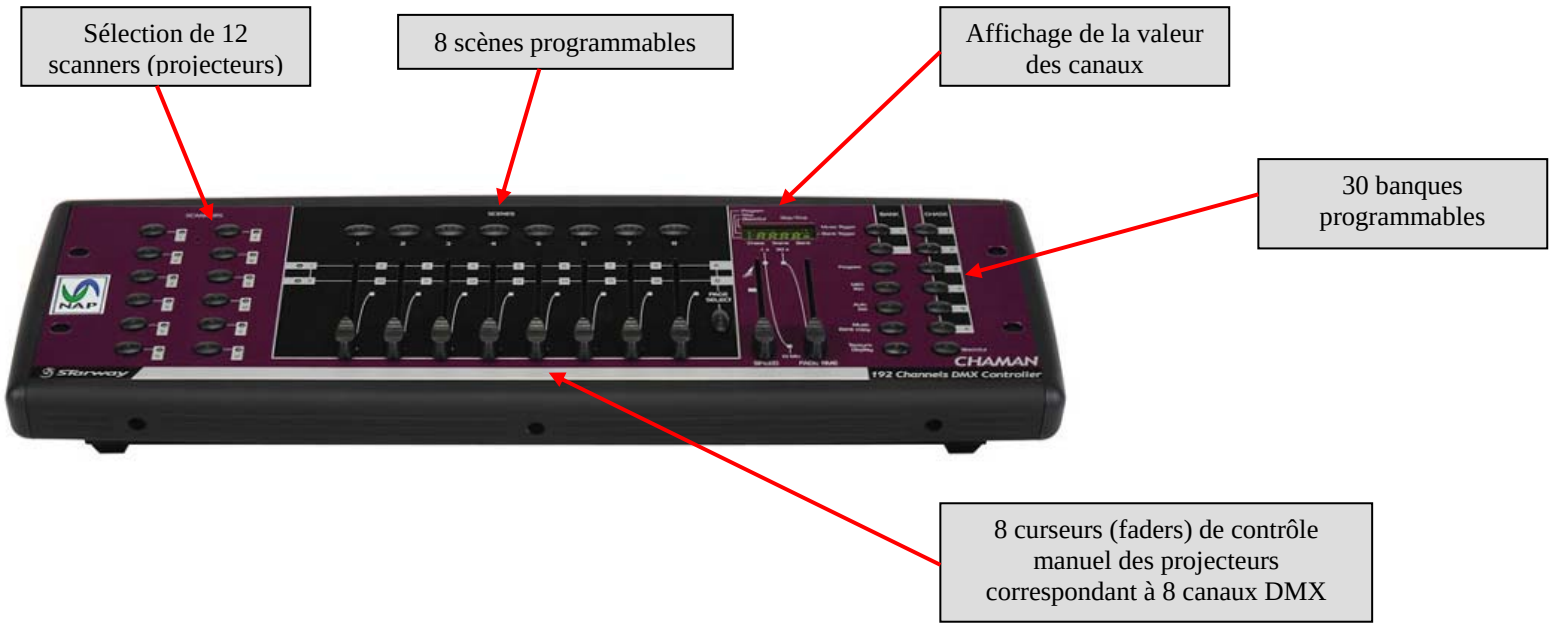
2 : Les chronogrammes acquis mettant en évidence les traitements numériques (envoi trame DMX, signaux SPI pilotant le CAN et les signaux de l'affichage multiplexé. Ces chronogrammes, concaténés dans un fichier WORD, doivent être commentés.

3 : Vous présenterez à l'enseignant votre programme le plus élaboré possible lors d'une mise en œuvre du banc de mesure décrit en annexe 5.

4 :Le diagramme de Gantt ci-dessus complété.

## ANNEXES : INFORMATIONS COMPLEMENTAIRES AU CAHIER DES CHARGES :

A1 : Présentation de la table DMX Chaman : (Extrait thème Bac STI 2012 – Régie lumière)




Le contrôleur Chaman, de la société Starway, est une console DMX portable conçue pour des professionnels de l'éclairage qui animent des spectacles dans des clubs de petite envergure, des bars ou autres installations fixes.

Cette console est facilement programmable et offre un accès rapide aux projecteurs pilotés par le protocole DMX. Pour réaliser un jeu de lumière, la console peut commander jusqu'à 12 projecteurs (scanners) et dispose de 16 canaux DMX pour transmettre à distance un ordre aux projecteurs.
Elle peut faire défiler les séquences manuellement ou automatiquement ou bien les synchroniser sur l'entrée audio, garantissant ainsi un spectacle dynamique et net. Il est également possible d'appeler des séquences pré-programmées ou des programmes spécifiques via l'entrée MIDI. Pour réaliser un jeu de lumière, la console peut commander jusqu'à 12 projecteurs (scanners). Chaque appareil peut avoir 16 canaux DMX.

**Extraits des spécifications de la Table DMX Chaman :**

- Commande de 12 projecteurs de 16 canaux DMX (192 canaux : 12 x 16) ;
- Pré-programmation possible de séquences dans 30 banques de 8 scènes ;
- curseurs (faders) de contrôle manuel des projecteurs (8 canaux DMX) ;
- 2 curseurs (faders) de contrôle de la vitesse de défilement des séquences pré-programmées ;
- 1 touche de sélection de page (Page Select) permettant d'obtenir 16 canaux DMX (2 x 8 canaux) ;
- Mode manuel ou automatique de défilements des séquences ;
- Affichage (4 digits) des valeurs des canaux DMX et des différents modes de fonctionnement ;
- Affichage par leds des scanners sélectionnés et de la page sélectionnée (Page A ou Page B) ;
- Commande Master de Blackout (mise en pause des sorties : extinction des projecteurs) ;
- Microphone intégré pour déclenchement musical des séquences pour un spectacle dynamique ;
- Télécommande MIDI, via des changements de programmes ou des notes ;
- Dimensions (mm) : 482 x 132 x 73 ;
- Poids : 2,5 Kgs ;
- Connexions : Sortie DMX (RS 485) : connecteur XLR 3 broches Femelles ;
- Entrée MIDI : connecteur DIN 5 broches Femelle ;
- Entrée DC 9V~12V, 300mA min.

A2 : Présentation du projecteur à leds CONTEST : (Extrait thème Bac STI 2012 – Régie lumière)


Les systèmes d'éclairage à leds sont de plus en plus utilisés de nos jours : éclairage domestique, éclairage urbain (décorations de Noël), phares de voitures…

Le projecteur étudié, **led 38 RGB**, fait partie de la gamme des projecteurs *« PAR à leds RGB et blanc »* de la société Contest. Il est conçu pour la production d'effets de lumière et est utilisé dans des spectacles lumineux.

Les projecteurs à leds comportent de nombreux avantages :

- pas d'ampoules à changer : 100 000 heures de fonctionnement garanties soit plus de 11 ans en continu !
- consomment peu, donc économiques et écologiques, c'est dans l'ère du temps.
- ne chauffent pas.
- légers, silencieux : pas besoin de ventilateurs pour refroidir les lampes.

Mais ils présentent aussi quelques inconvénients :

- la puissance lumineuse reste faible et peut être insuffisante pour certains spectacles.
- le système n'étant pas motorisé, la lumière est toujours émise dans la même direction.

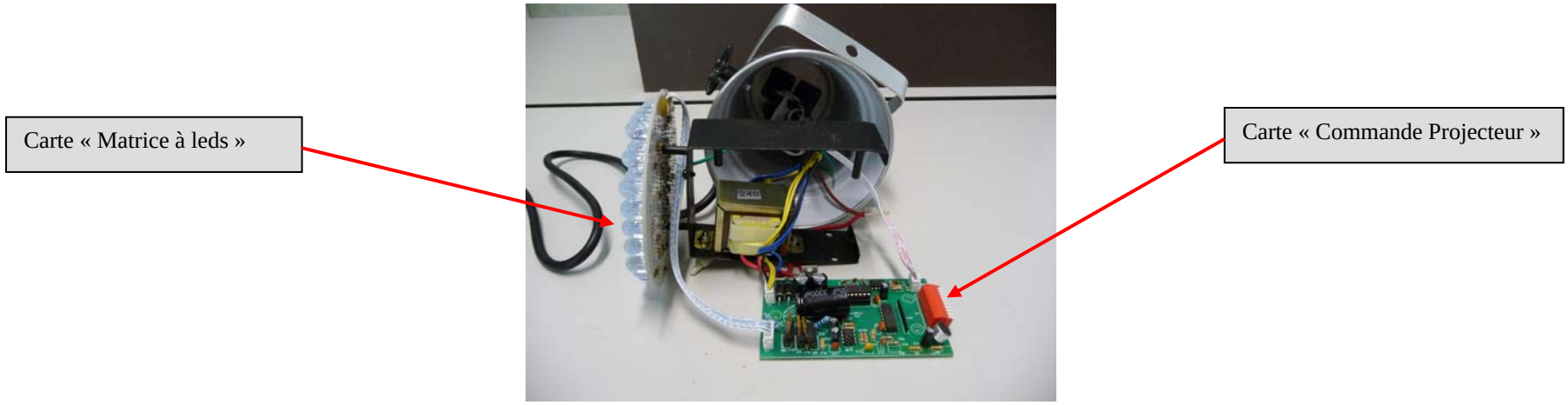
Une matrice à leds constituée de leds rouges, vertes et bleues permet, par synthèse additive, de disposer d'un nombre important de couleurs.

Un réglage de l'intensité lumineuse émise est possible, ainsi que du mode d'éclairage : statique, clignotant ou stroboscopique.

Le projecteur est contrôlable soit manuellement par une console DMX selon la norme DMX512, soit de façon autonome en mode musical (les couleurs changent au rythme du son ambiant) ou en mode automatique. L'affectation des différents modes et le choix de l'adresse DMX se fait via le DipSwitch situé sur la face arrière de l'appareil.

La table DMX peut transmettre les commandes à distance au projecteur depuis 4 canaux DMX et permet de gérer : la couleur, l'intensité lumineuse, le mode de fonctionnement.

Le projecteur est constitué de deux cartes :



**Extraits des spécifications du projecteur « led 38 RGB » :**

- Le projecteur est constitué de 75 leds : 24 rouges, 24 vertes, 27 bleues ;
- Leds 10 mm ; 100 000 heures de fonctionnement ;
- Ouverture 30° ;
- Plusieurs modes de fonctionnement :
- Mode DMX : Pilotage de 4 canaux DMX (Rouge/Vert/Bleu/Dimmer et Strobe) ;
- Mode Musical : Autonome en mode musical pour un spectacle dynamique ;
- Mode Automatique ;
- Choix du mode et adressage par 10 DipSwitches ;
- Microphone intégré pour déclenchement musical des séquences ;
- Dimensions (mm) : 188 x 154 x 135 – Poids : 1 Kgs ;
- Consommation : 9 Watts ;
- Connexions : Entrée DMX (RS 485) : connecteur XLR 3 broches Mâles ;
  Sortie DMX (RS 485) : connecteur XLR 3 broches Femelles ;
  Entrée Secteur : ~230VAC / 50Hz / Fusible 500mA ;

A3: Adressage DMX : (Extrait thème Bac STI 2012 – Régie lumière)

***Explications de base pour comprendre comment fonctionne le principe de codage DMX, l'affectation des canaux et l'assignation de projecteurs ou d'appareils commandés en DMX.***

**Principe de base**
Le DMX 512 est un signal de commande numérique qui permet de faire passer dans un seul câble 512 canaux d'informations. Ces informations comportent chacune 255 niveaux (de 0 à 255 ou de 0 à 100%).

Pour vous aidez, vous pouvez vous représenter un canal DMX comme un fader (potentiomètre rectiligne). Chaque projecteur (ou bloc, ou machine à fumée, etc..., pour simplifier, on parlera de projecteur) utilise un certain nombre de ces canaux. Chaque canal correspondant à une fonction ou à une commande spécifique.

Prenons par exemple le projecteur à Leds PAR 38 RGB. Il utilise 4 canaux DMX. Ces canaux sont répartis de cette façon :

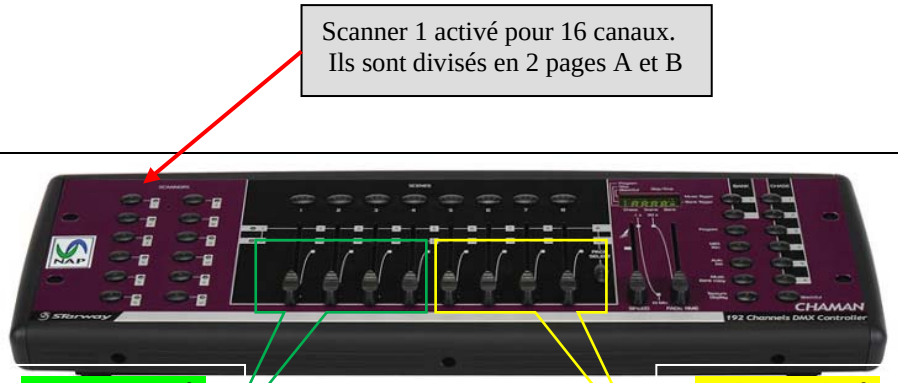
| Canaux | DMX | Contrôle |
|---|---|---|
| Canal 1 | 000 - 255 | Rouge |
| Canal 2 | 000 - 255 | Vert |
| Canal 3 | 000 - 255 | Bleu |
| Canal 4 | 000 - 189 | Dimmer |
| | 190 - 250 | Clignotement |
| | 251 - 255 | Aucun changement |

Cela veut dire concrètement, que sur une console DMX, on utilisera 4 faders (potentiomètres rectilignes) pour son pilotage. Chacun des faders va contrôler une fonction, suivant l'ordre décrit ci-dessus.

Dans le cas d'un projecteur à leds PAR 38 RGB adressé en "1" par exemple, si vous agissez sur le fader n°3 de la console, vous allez contrôler la couleur Bleu (éclairage minimum si vous êtes à 0%, éclairage maximum si vous êtes à 100%, éclairage moyen si vous êtes à 50%).

## Adressage

L'adressage DMX permet d'affecter chaque projecteur à des canaux définis sur la trame des 512 canaux du signal DMX. Ainsi, sur l'ensemble d'une ligne DMX, vous pouvez assigner 128 Projecteurs à Leds PAR 38 RGB pouvant répondre tout à fait indépendamment les uns des autres (512 divisé par 4). Attention, la table Chaman ne commande que 192 canaux maximum.




L'adresse d'un projecteur doit correspondre au numéro du premier canal qu'il va utiliser. Ainsi, si vous adressez un Projecteur à Leds PAR 38 RGB en "00000001", il va répondre sur les faders de 1 à 4 sur votre console DMX. Si vous l'adressez en "00000101", il va répondre sur les faders de 5 à 8 de votre console DMX (voir illustration).

Le plus souvent, on adresse les projecteurs les uns derrières les autres. Le premier étant en "1", le second est donc en "5", le troisième en "9", le quatrième en "13", et ainsi de suite.

Adresse 1

Adresse 5

## Le DIPSWITCH

Le projecteur à Leds PAR 38 RGB s'adresse soit par un petit pavé "DIPSWITCH.

Dans le cas du DIPSWITCH, voilà comment procéder :


Un DIPSWITCH se compose en général de 10 switchs qui agissent en position ON / OFF. Le ON étant déterminé par le sens de la flèche, ou même souvent indiqué en toutes lettres.
Dans le cas du Projecteur à Leds PAR 38 RGB, les switchs 9 et 10 définissent les modes de fonctionnement (Mode automatique, Mode impulsion sonore, Assignation DMX)

Mode Automatique : DIP#9 OFF et DIP #10 OFF

Sélectionnez la vitesse d'enchaînement de couleur et la durée de chaque étape avec les dipswitches 7 et 8.

Sélectionnez le mode couleur statique avec les dipswitches 1 à 3 (1 pour le rouge, 2 pour le vert et 3 pour le bleu). Sélectionnez la vitesse de clignotement de la lumière avec les dipswitches 4 à 6.

Mode impulsion sonore : DIP#9 ON

Ce mode permet le changement de couleur du projecteur en fonction de l'impulsion sonore. Enclenchez le dip switch #9 pour activer le mode impulsion sonore.

Assignation DMX : DIP#10 ON

Si vous utilisez une télécommande DMX pour contrôler vos appareils, vous devez programmer les dipswitches de tous les appareils qui recevront le signal DMX.

Enclenchez le dipswitch #10 pour activer le mode DMX.

Enclenchez les dipswitches #1 à #8 pour sélectionner l'adresse DMX. L'appareil utilise 4 canaux DMX, veuillez donc assigner les projecteurs de 4 en 4 (projecteur n°1 en adresse 1, projecteur n°2 en adresse 5 , projecteur n°3 en adresse 9...).

Pour adresser un projecteur, il suffit de coder en un nombre binaire l'adresse du premier canal DMX en plaçant les switchs sur ON ou OFF (la position ON correspond au niveau logique 1).

Exemple :

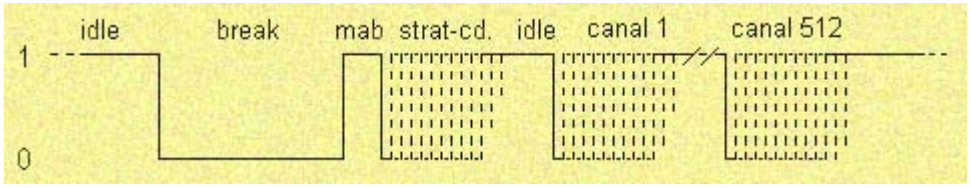
Pour un adressage DMX commençant par le canal 10, le code est :

canal 010 $\Rightarrow$ 0000 $1010_{binaire}$ $\Rightarrow$ switch 2 et 4 sur ON, les autres sur OFF.

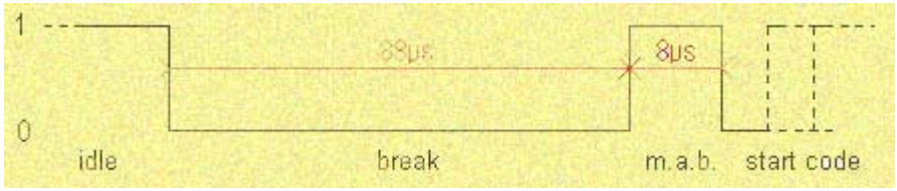
A4: Protocole DMX (Extrait thème Bac STI 2005 – Lyre Asservie SPOT 150)

Les données d'un bus DMX 512 sont transmises sous la forme d'une succession d'octets vers une liaison série différentielle RS485. Un octet est composé de 8 bits et peut avoir 256 états pour représenter une valeur d'intensité, de couleur, de position, etc…

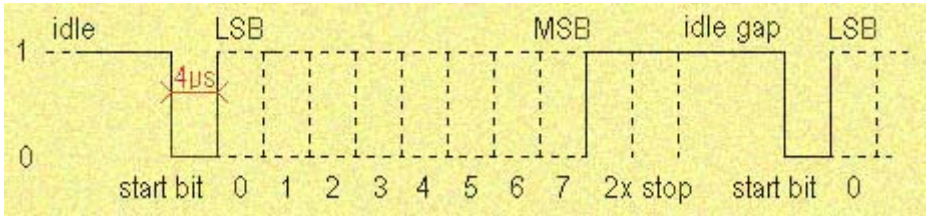
**Principe :**
Le DMX utilise un codage temporel où les informations sont transmises dans un ordre croissant. Un cycle commence par une initialisation (break + mark after break), puis suit un code indiquant la nature des informations (start-code), les 11 bits permettant la transmission du premier octet (1start-bit + 8 bits data + 2 stop-bits), puis les 11 bits du circuit 2, etc. ceci jusqu'au dernier canal. Certaines consoles n'ont pas les ressources suffisantes pour transmettre les signaux de façon continue et peuvent intercaler un temps de pause précédant les start-bits. Enfin comme il a déjà été dit, il reste tout à fait possible de restreindre le nombre de canaux à transmettre (entre 24 et 512).




Structure d'un bloc de données :
**Un break** (r.a.z.) de 88 µs minimum (durée de transmission de deux canaux). Il n'y a pas de maximum fixé par la norme mais certains équipements tolèrent mal les durées trop importantes (au delà de 200 ms).
**Une impulsion** ( mab) **Mark After Break** (état de travail) de 8 µs minimum (durée deux bits).
**Le start-code** (assimilable à un canal 0) indique la nature des informations transmises, null-start (valeur transmise 0 ) pour les données linéaires sur 8 bits : les gradateurs sont donc censés ignorer tout autre start-code compris entre 1 et 255. Les changeurs de couleurs ainsi que de nombreuses consoles et projecteurs automatisés utilisent aussi ce start-code '0', et restent compatibles avec une console dédiée à la commande de gradateurs. Les autres codes sont réservés pour un usage futur mais aujourd'hui certains fabricants tirent profit de cette possibilité afin d'optimiser leurs systèmes.
**Les données** sont présentées sur le bus de façon sérielle. Un bit a une durée de 4µs avec une tolérance de 2%.
**Des temps de repos** (idle) peuvent être intercalés entre les trames, caractérisés par un état haut de la ligne.



Composition de l'information transmise pour un canal:
**Un start-bit**, état bas, précède la transmission de l'octet
**LSB > MSB** : Du bit de poids le plus faible jusqu'au bit de poids le plus fort
**Deux bits de stop**, état haut, après la fin de l'octet
**44 µs minimum entre deux trames** (rappel : durée 1 bit 4µs )




Limites des temps de transmission :

L'intervalle séparant deux impulsions de remise à zéro (break + mark after break) doit être:

- d'au moins 1 196 µs (durée de la transmission de 24 circuits)
- au maximum d'une seconde (temps de repos inclus). En cas d'absence de signal, le récepteur doit maintenir son dernier état au minimum pendant cette durée d'une seconde.

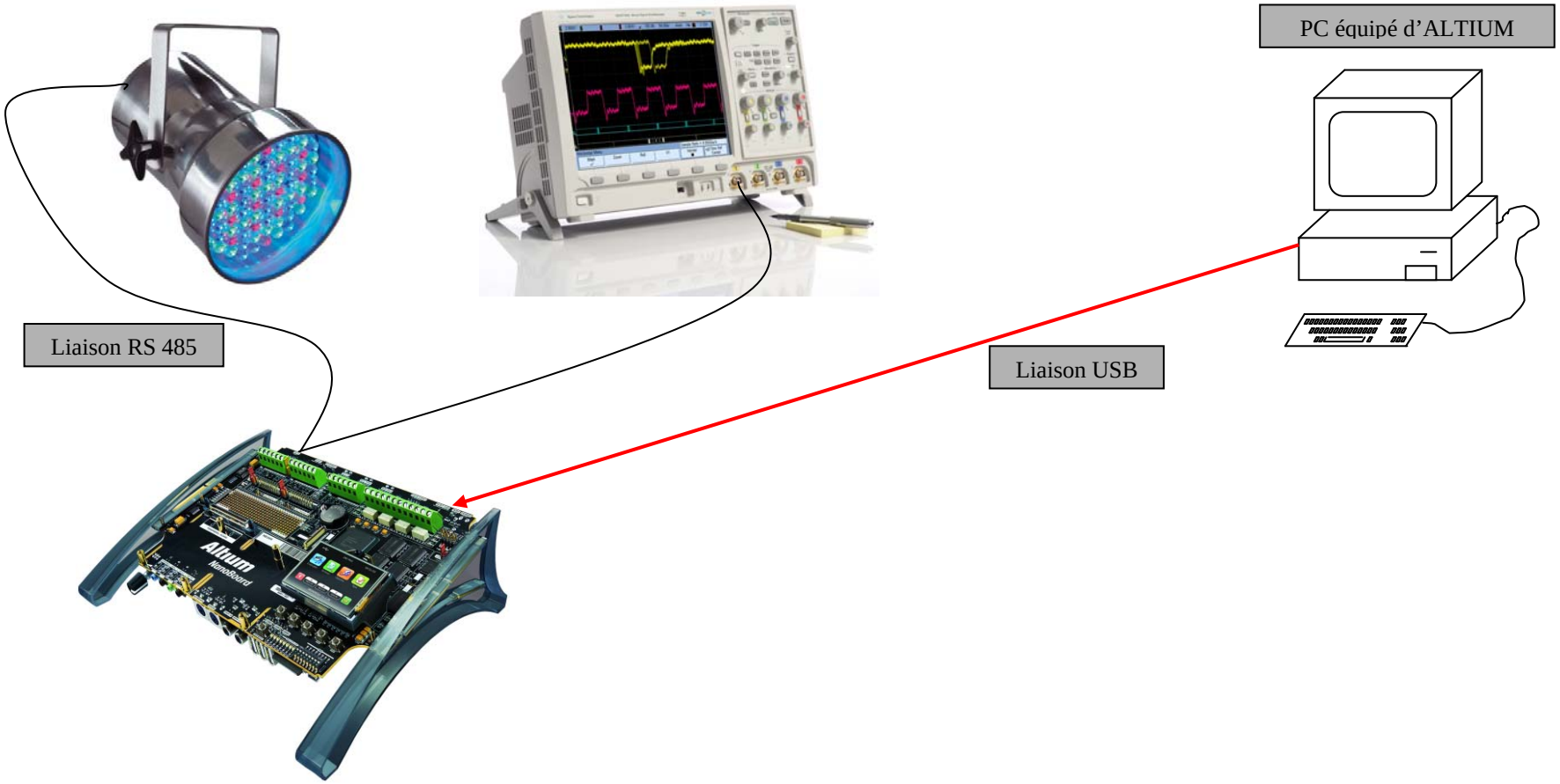
Calcul des temps de rafraîchissement :

- **Pour 512 canaux** on a une fréquence de rafraîchissement de 44 Hz
  durée de la trame (break :88µs) + (m.a.b. :8µs) + (start-code :44µs)+ (512 * 44µs)= 22668 µs d'où une fréquence de 44,1 Hz
- **Pour 24 canaux** on a une fréquence de rafraîchissement de 836 Hz
  durée de la trame (break :88µs) + (m.a.b. :8µs) + (start-code :44µs)+ (24 * 44µs)= 1196 µs d'où une fréquence de 836,1 H

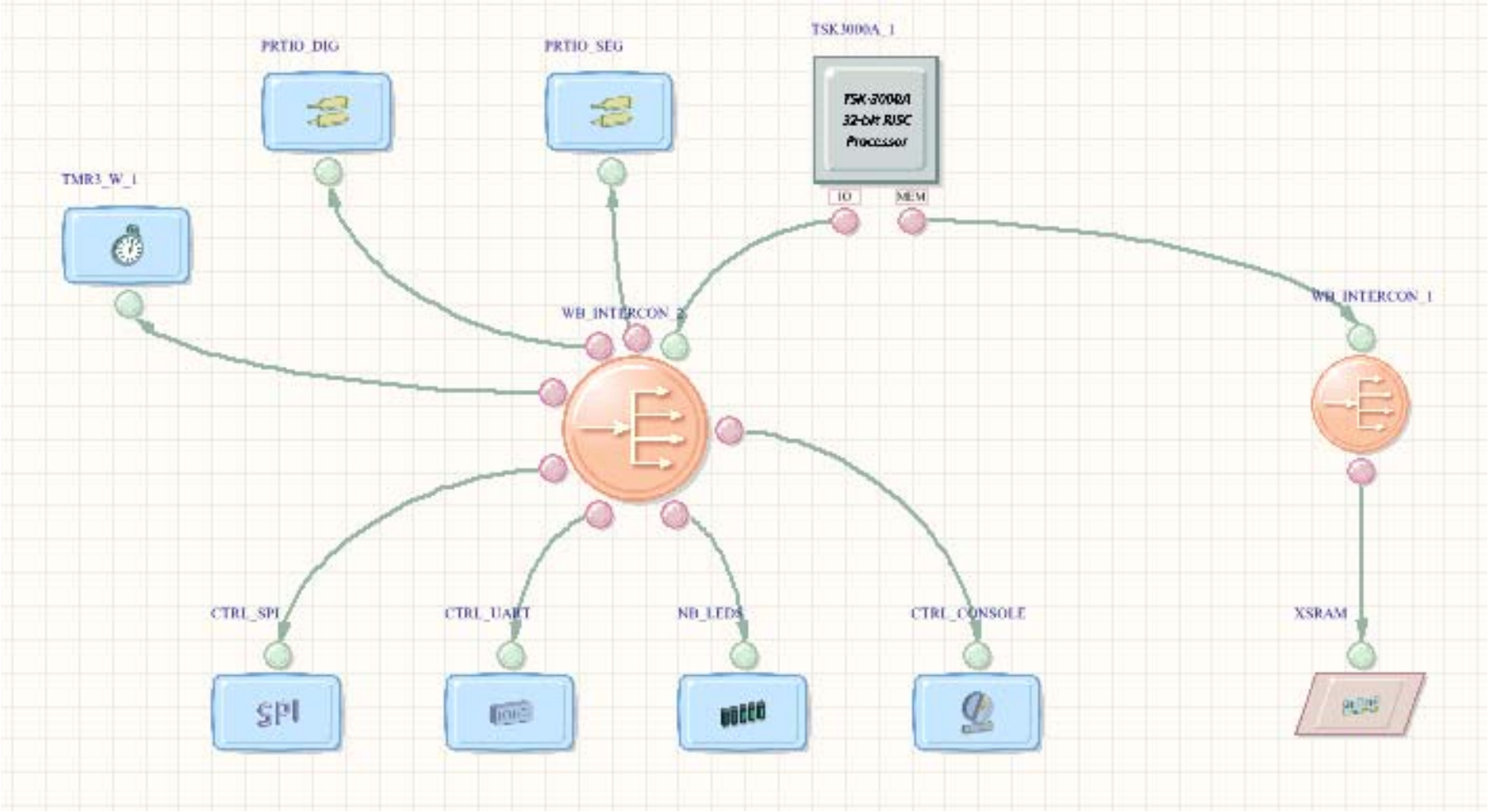
Tableau récapitulatif des durées :

| APPELLATIONS | DUREE TYPE (µs) | DUREE MIN. (µs) | DUREE MAX.(µs) |
|---|---|---|---|
| break | **88** | **88** | **200 000** |
| mark after break | **8** | **8** | **$10^6$** |
| 1 bit | **4** | **3,92** | **4,08** |
| Intervalle entre deux débuts de trame | **22 668** | **1 196** | **$10^6$** |

A5 : Synoptique du banc de mesure permettant de valider les résultats du mini projet.

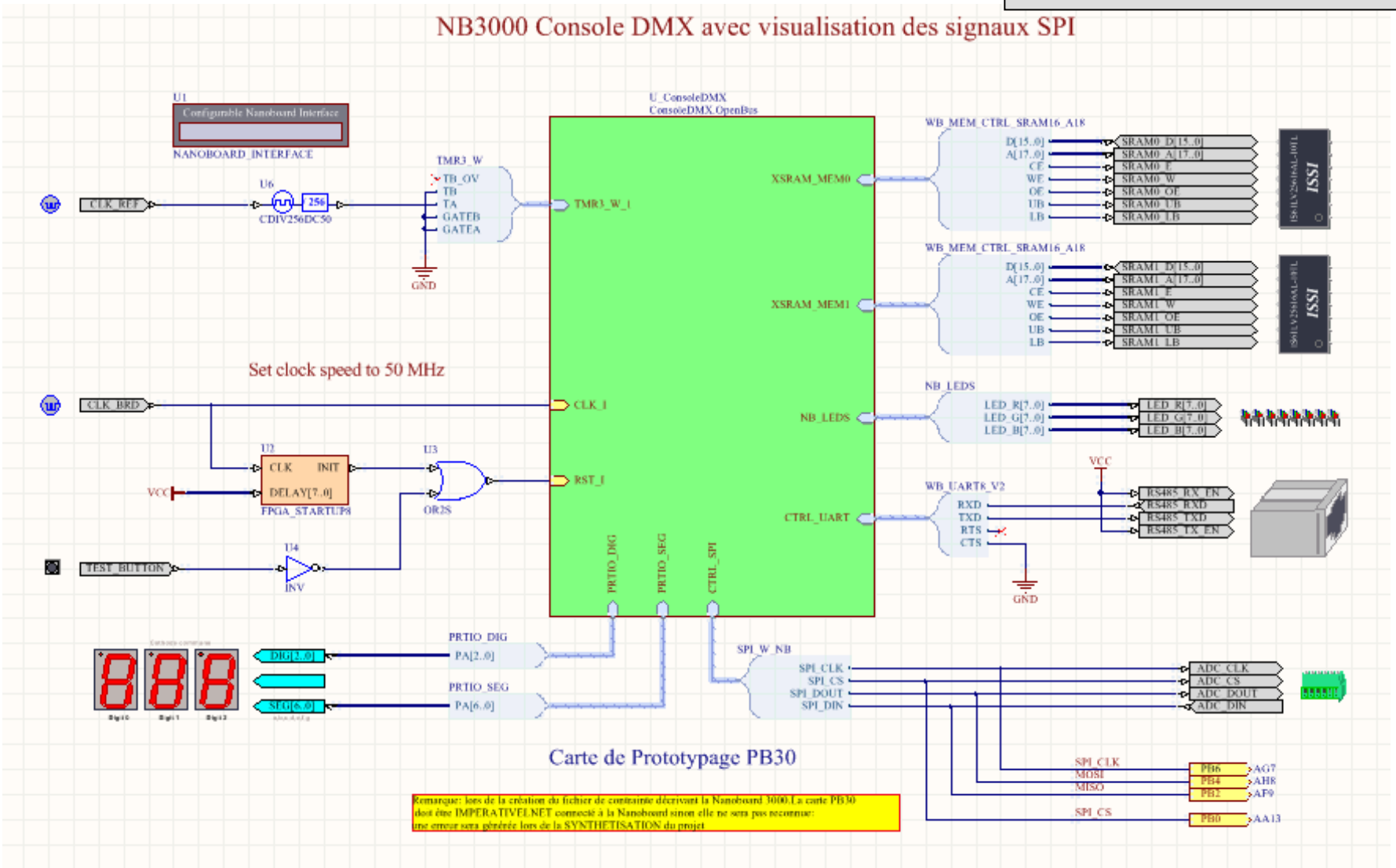


A6 : Schéma « Open  Bus » à mettre en place pour le projet.



Vous serez amené à saisir le schéma ci-dessous lors de l'étape de création du projet

A7 : Schéma « TOP » à mettre en place dans le projet :



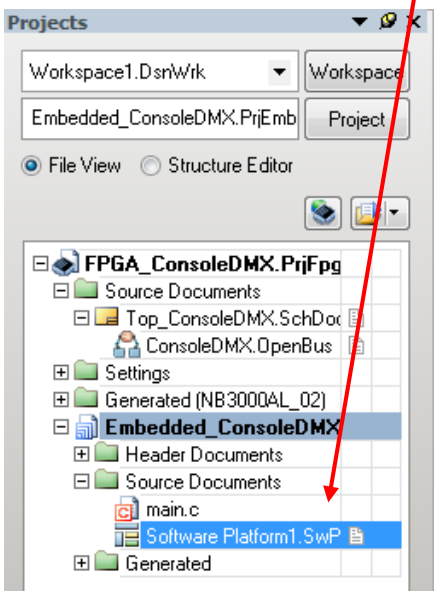
A8 : Présentation des principales fonctions **API** en langage C à mettre en œuvre durant le projet.

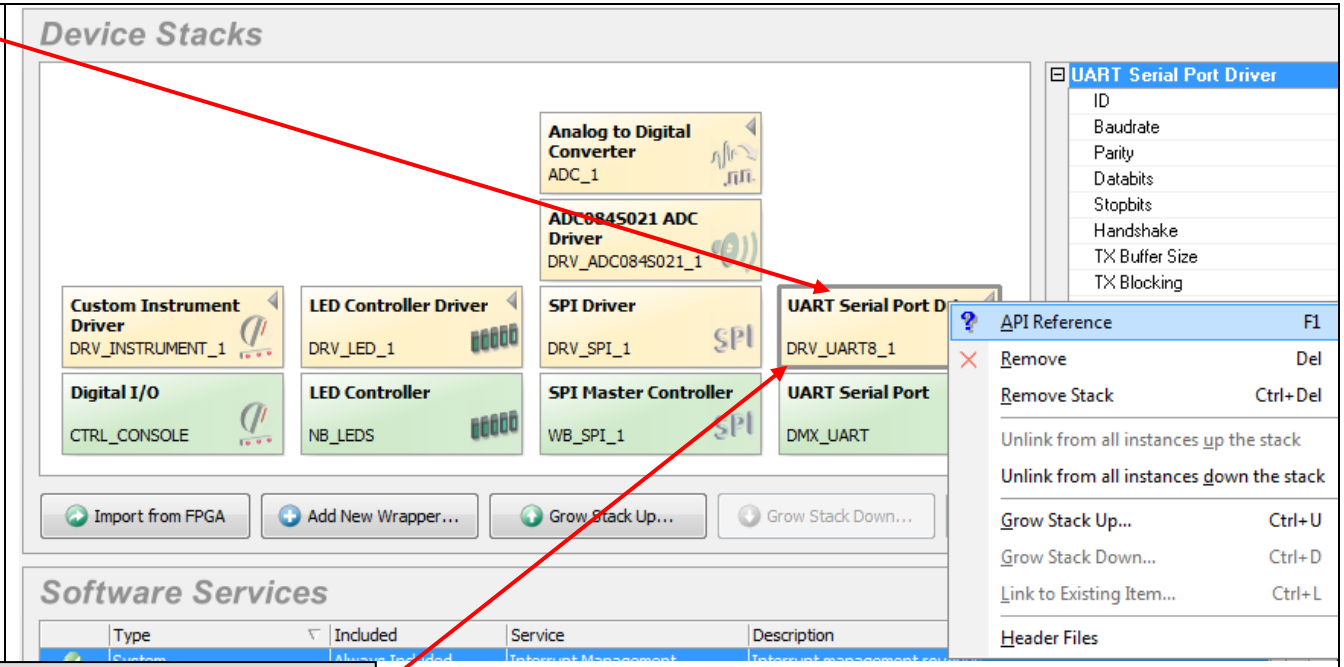
**API** : **A**pplication **P**rogramming **I**nterface
Une **API** est un ensemble de fonctions, procédures ou classes mises à disposition par une bibliothèque logicielle qui permet d'interfacer deux entités.
Dans notre cas les **API** mises à notre disposition nous permettent de dialoguer avec les composants placés sur la carte Nanoboard.
Nous nous intéresserons plus particulièrement aux API permettant de :

- ⇒ mettre en œuvre le bus SPI (pour paramétrer le CNA et acquérir une grandeur analogique).
- ⇒ mettre en œuvre la liaison série RS485 (pour transférer les données numériques vers la liaison série RS485).
- ⇒ mettre en œuvre l'affichage multiplexé sur la carte fille PB30.

Où trouver les API disponibles au sein d'un projet FPGA sous ALTIUM ?

Les **API** dédiées à un composant sont accessibles par un clic droit sur l'icône du composant dans le fichier **SwPlatform**.




Pour le port série, l'identifiant est « DRV_UART8_1 »

API : Universal Serial Port Driver

**Syntax**

```
#include <drv_uart8.h>
uart8_t * uart8_open(unsigned int id);
```

This function initializes both a UART8 core and its driver. You should call it only once per instantiation.

| Parameter | Description |
|---|---|
| id | valid driver id |

**Returns:** NULL if invalid id is passed, pointer to initialized driver otherwise

Fichier d'entête

Prototype de la fonction « uart_open ». Cette fonction permet d'ouvrir le port de la liaison série.

Exemple :

```
// Ouverture du périphérique port série UART
    drv_uart8_1  = uart8_open(DRV_UART8_1);
```

Identifiant du driver du port série

**Syntax**

```
#include <drv_uart8.h>
void uart8_set_parameters(uart8_t *restrict drv, uint32_t baudrate,
uart8_parity_t parity, uint8_t wordlen, uint8_t stopbits);
```

| Parameter | Description |
|---|---|
| drv | uart8 driver pointer |
| baudrate | communication speed to be used (in bps) |
| parity | parity to be used |
| wordlen | word length to be used, minimal 5 and maximal 8 bits per serial word |
| stopbits | number of stopbits, must be 1 or 2 |

**Returns:**

Prototype de la fonction « uart_set_parameters ». Cette fonction permet de paramétrer la liaison série à une vitesse de transmission de 250 kBauds/ s avec 2 bits de stops, pas de parité.

Exemple :

```
// 250kbauds, pas de parité, 8 bits de données et 2 bits de stops
entre l'envoi de 2 caractères
    uart8_set_parameters(drv_uart8_1,250000, UART8_NO_PARITY,8,2);
```

**Syntax**

```
#include <drv_uart8.h>
int uart8_putbreak(uart8_t *restrict drv, uint8_t duration);
```

Have the UART8 generate a breakcondition of `length` bits long. The break-instruction is added to the transmit buffer, but there can only be one pending break-instruction at a time. This function can be blocking or non-blocking depending on the plugin settings.

| Parameter | Description |
|---|---|
| drv | uart8 driver pointer |
| duration | length of break in bits |

**Returns:** 0 if succes, -1 if there is no space or already a pending break

Prototype de la fonction « uart8_putbreak ». Cette fonction permet d'envoyer un break d'une longueur de « n » Tbits vers la liaison série.

Exemple :

```
// Envoi du break d'une durée de 22 Tbits
uart8_putbreak(drv_uart8_1, 22);
```

**Syntax**

```
#include <drv_uart8.h>
int uart8_putchar(uart8_t *restrict drv, int val);
```

Put a single byte `val` in the transmit buffer (or in the core if the buffer size is zero). This function can be blocking or non-blocking depending on the plugin settings.

| Parameter | Description |
|---|---|
| drv | uart8 driver pointer |
| val | byte to transmit. |

**Returns:** 0 if succes, -1 if no space.

Prototype de la fonction « uart8_putchar ». Cette fonction permet d'envoyer un caractère vers la liaison série

Exemple :

```
// Envoi du caractère null
uart8_putchar(drv_uart8_1,0);
```

**Syntax**

```
#include <drv_uart8.h>
size_t uart8_transmit_buf_free(uart8_t *restrict drv);
```

Test if transmitter has free space left to write more data. If there is no transmit buffer configured, the maximum returned value will be 1 if any data at all can be written to the core.

| Parameter | Description |
|---|---|
| drv | uart8 driver pointer |

**Returns:** Amount of free data available in the transmit buffer.

Prototype de la fonction « uart8_transmit_buf_free ». Cette fonction permet de tester si le buffer d'émission est vide.

Exemple :

```
// Test que le buffer d'émission est vide
while (!uart8_transmit_buf_free(drv_uart8_1));
```

API : LED Controler Driver

**Syntax**

```
#include <drv_led.h>
led_t* led_open(int id);
```

This function opens an instance of an LED Controller driver. The LED Controller device's properties are stored in the configuration data returned. This function should only be called once per driver instance id.

| Parameter | Description |
|---|---|
| id | Valid driver id (defined in devices.h) |

**Returns:** configuration data pointer if successful initialization, NULL otherwise.

Fichier d'entête

Prototype de la fonction « led_open ». Cette fonction permet d'ouvrir le port led.

Exemple :

```
// ouverture du périphérique DRV_LED
leds = led_open(DRV_LED_1);
```

Identifiant du driver du contrôler de leds

**Function: led_turn_all_off**

**Syntax**

```
#include <drv_led.h>
void led_turn_all_off(led_t *restrict led_drv);
```

Turn off all LEDs.

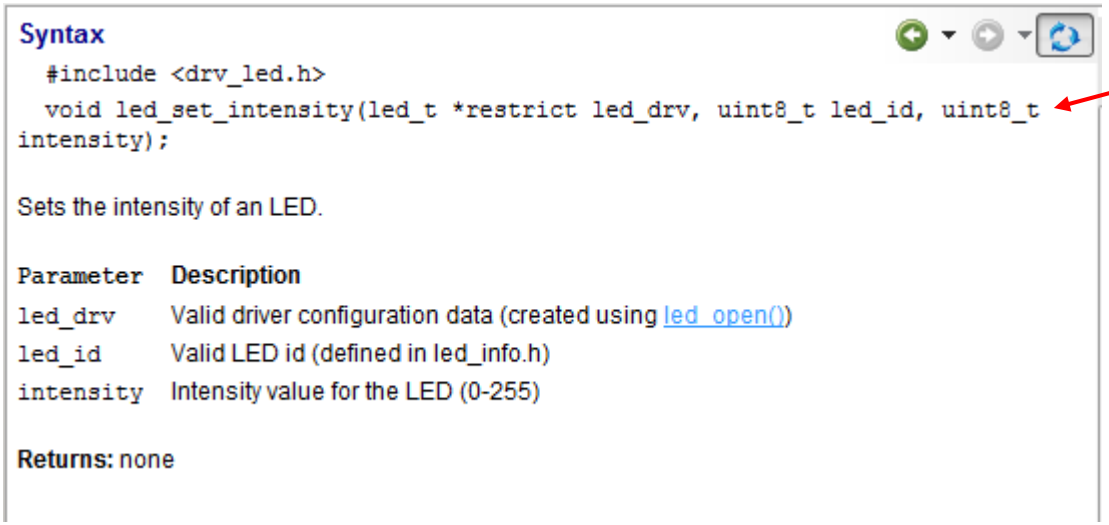
| Parameter | Description |
|---|---|
| led_drv | Valid driver configuration data (created using led_open()) |

**Returns:** none

Prototype de la fonction « led_turn_all_off ». Cette fonction permet d'éteindre toutes les leds.

Exemple :

```
// Extinction des leds
led_turn_all_off(drv_led_1);
```



**Syntax**

```
#include <drv_led.h>
void led_set_intensity(led_t *restrict led_drv, uint8_t led_id, uint8_t
intensity);
```

Sets the intensity of an LED.

| Parameter | Description |
|---|---|
| led_drv | Valid driver configuration data (created using led_open()) |
| led_id | Valid LED id (defined in led_info.h) |
| intensity | Intensity value for the LED (0-255) |

**Returns:** none

Prototype de la fonction « led_set_intensity ». Cette fonction permet de commander le niveau d'éclairement des leds.

Exemple :

```
// Envoi d'une consigne issue d'un tableau vers les leds RVB
led_set_intensity(drv_led_1, i, DMX_Table[i]);
```

API : Custom Instrument Driver

**Syntax**

```
#include <drv_instrument.h>
instrument_t * instrument_open(unsigned int id);
```

This function opens an instance of a Custom Instrument driver. The properties of the Custom Instrument or Digital I/O device are stored in the configuration data returned.

| Parameter | Description |
|---|---|
| id | Valid driver ID (see devices.h) |

**Returns:** pointer to configuration data for the specified device, or NULL for invalid id

Fichier d'entête

Prototype de la fonction « instrument_open ». Cette fonction permet d'ouvrir instrument

Exemple :

```
// Ouverture du périphérique virtuel instrument INTRUMENT
drv_instrument_1 = instrument_open(DRV_INSTRUMENT_1);
```

**Syntax**

```
#include <drv_instrument.h>
uint32_t instrument_get_value(instrument_t *restrict drv, unsigned int signal_id);
```

This function reads a value from a specific output signal in the Custom Instrument or Digital I/O device. Note that if the specified signal ID is invalid, the function will return 0. Use the signal IDs in the generated header file instruments.h, which are in the format WrapperInstanceName_SignalName.

| Parameter | Description |
|---|---|
| drv | Driver configuration data |
| signal_id | Signal to retrieve the value from; must be from 0 to (drv->output_signal_count - 1); valid signal IDs can be found in instruments.h |

**Returns:** the value of the signal if the signal ID is valid, or 0 otherwise.

Prototype de la fonction « instrument_get_value ». Cette fonction permet de lire la consigne donnée par l'instrument virtuel.

Exemple :

```
// Lecture d'une consigne donnée par l'instrument virtuel et stockage du résultat dans un tableau
DMX_Table[i]=instrument_get_value(drv_instrument_1, i);
```

API : ADC084S021 ADC Driver

**Function: adc084s021_open**

**Syntax**

```
#include <drv_adc084s021.h>
adc084s021_t* adc084s021_open(unsigned int id);
```

This function initializes the adc084s021 device driver.

| Parameter | Description |
|---|---|
| id | valid driver id |

**Returns:** Driver pointer if succesful - NULL otherwise

Fichier d'entête

Prototype de la fonction « adc084s021_open ». Cette fonction permet d'ouvrir le port du convertisseur ADC.

Exemple :

```
// Ouverture du périphérique port convertisseur ADC084S021
drv_adc084s021_1=adc084s021_open(DRV_ADC084S021_1);
```

**Function: adc084s021_read**

**Syntax**

```
#include <drv_adc084s021.h>
int adc084s021_read(adc084s021_t *restrict drv, int
next_channel);
```

This function reads a single value from the adc084s021 device.

| Parameter | Description |
|---|---|
| drv | pointer to opened device driver |
| next_channel | channel for the next read (0 .. 3) |

**Returns:** -1 on error, value read from ADC otherwise

Prototype de la fonction « adc084s021_read ». Cette fonction permet de lire la valeur numérique donnée par le convertisseur ADC.

Exemple :

```
// Lecture d'une consigne donnée par le convertisseur ADC
et stockage du résultat dans un tableau

DMX_Table[0]= (char) adc084s021_read(drv_adc084s021_1,0);
```

API : GPIO Port Driver

Fichier d'entête

## Function: ioport_open

### Syntax

```
#include <drv_ioport.h>
ioport_t* ioport_open(const int id);
```

This function opens an instance of an I/O Port driver. The I/O Port peripheral's properties are stored in the configuration data returned.

| Parameter | Description |
|---|---|
| id | a valid driver ID |

**Returns:** configuration data for the I/O Port driver

Prototype de la fonction « ioport_open ». Cette fonction permet d'ouvrir le port IO

Exemple :

```
// Ouverture des ports IO
    drv_ioport_1 = ioport_open(DRV_IOPORT_1);
    drv_ioport_2 = ioport_open(DRV_IOPORT_2);
```

### Syntax

```
#include <drv_ioport.h>
void ioport_set_value(ioport_t *restrict drv, const uint8_t port, const uint32_t value);
```

This function writes a value to a specific port number in the I/O Port. Note that if the specified port number is invalid, the function will do nothing. Note that if the data width of the I/O Port peripheral is actually lower than 32 bits, the value to be written will be truncated.

| Parameter | Description |
|---|---|
| drv | I/O Port driver configuration data |
| port | the port number to write the value to; must be from 0 to (drv->port_count - 1) |
| value | the value to be written to the port |

**Returns:** none

Prototype de la fonction « ioport_set_value ». Cette fonction permet d'écrire sur les ports IO

Exemple :

```
// Ecriture sur le port DIGIT : Sélection Digit Unité
ioport_set_value( drv_ioport_1, 0, UNITE);
// Ecriture sur le port SEGMENT: Envoi commande sur les 7 segments
ioport_set_value( drv_ioport_2, 0, BCD_7Seg_Table[Unite]);
```

API : TMR3 Dual Timer Driver

Fichier d'entête

## Function: tmr3_open

**Syntax**

```
#include <drv_tmr3.h>
tmr3_t * tmr3_open(int id);
```

| Parameter | Description |
|---|---|
| id | id of the device to open |

**Returns:** pointer to opened device or NULL when device could not be opend

Prototype de la fonction « tmr3_open ». Cette fonction permet d'ouvrir le port du convertisseur ADC.

Exemple :

```
// Ouverture du périphérique port Timer 3
    drv_tmr3_1 = tmr3_open(DRV_TMR3_1);
```

## Function: tmr3_timer_a_set_handler

**Syntax**

```
#include <drv_tmr3.h>
int tmr3_timer_a_set_handler(tmr3_t * drv, tmr3_handler handler, void *
data);
```

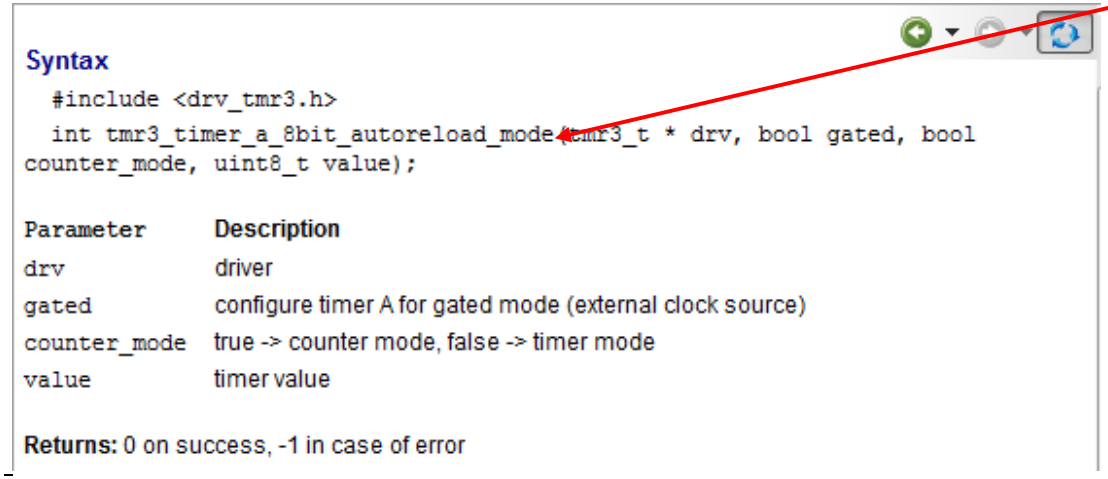
| Parameter | Description |
|---|---|
| drv | driver |
| handler | user handler to call when an interrupt for timer A occurs |
| data | application data to pass to the handler |

**Returns:** 0 on success, -1 in case of error

Prototype de la fonction « tmr3_a_set_handler». Cette fonction de gestion d'évènement qui fait le lien entre le débordement du timer3 et l'appel de la fonction d'interruption « handler_timer_a »

Exemple :

```
// Attribution de la gestion de l'interruption du Timer3 à un
handler timer_a
tmr3_timer_a_set_handler(drv_tmr3_1, handler_timer_a, NULL);
```



**Syntax**

```
#include <drv_tmr3.h>
int tmr3_timer_a_8bit_autoreload_mode(tmr3_t * drv, bool gated, bool
counter_mode, uint8_t value);
```

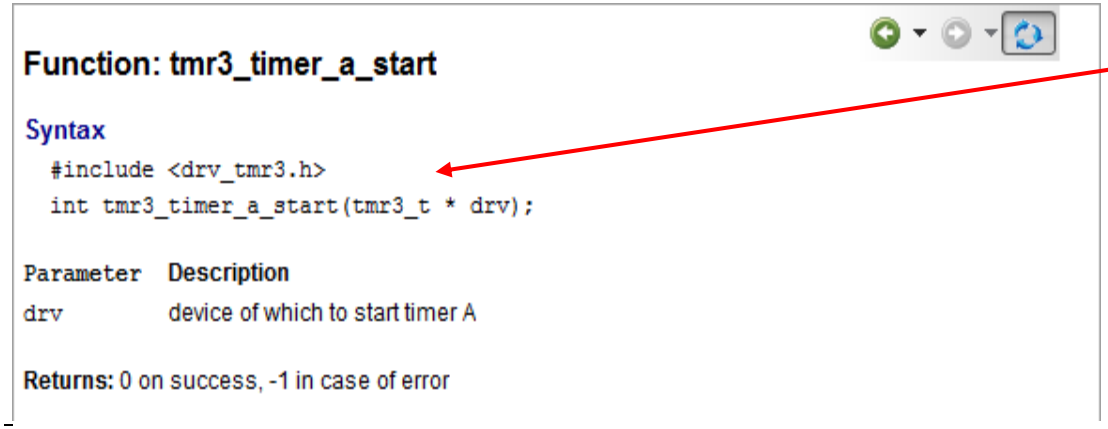
| Parameter | Description |
|---|---|
| drv | driver |
| gated | configure timer A for gated mode (external clock source) |
| counter_mode | true -> counter mode, false -> timer mode |
| value | timer value |

**Returns:** 0 on success, -1 in case of error

Prototype de la fonction « tmr3_a_8_autoreload_mode». Cette fonction permet de paramétrer le mode de fonctionnement du timer3 :
8 bist en auto rechargement
horloge interne
mode compteur = 128

Exemple :

```
// Timer 3 : Gate a non Activitée, Mode Compteur,
Compteur qui permet de diviser par 128
tmr3_timer_a_8bit_autoreload_mode(drv_tmr3_1, 0, 1,
128);
```



**Function: tmr3_timer_a_start**

**Syntax**

```
#include <drv_tmr3.h>
int tmr3_timer_a_start(tmr3_t * drv);
```

| Parameter | Description |
|---|---|
| drv | device of which to start timer A |

**Returns:** 0 on success, -1 in case of error

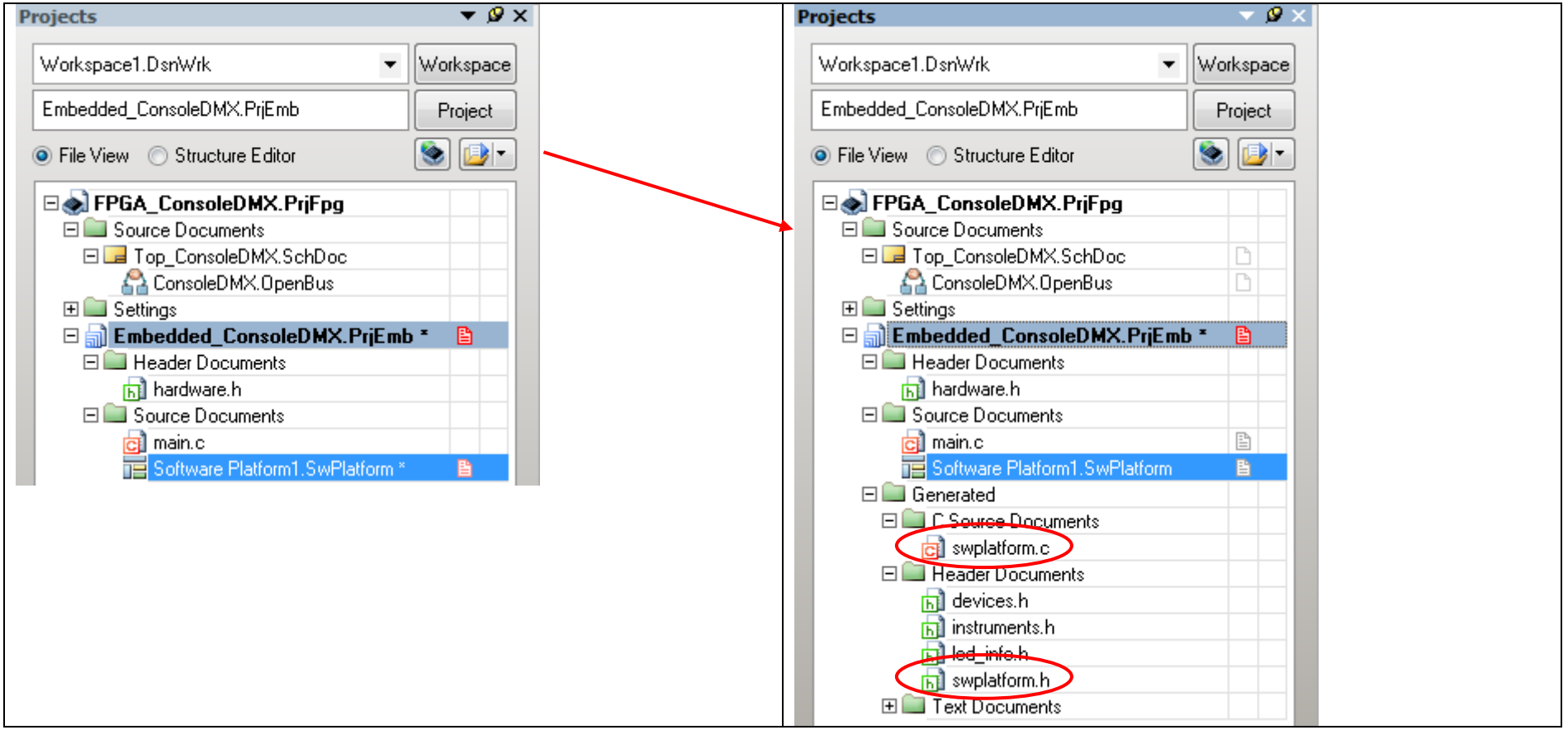
Prototype de la fonction «tmr3_timer_a_start ». Cette fonction permet d'activer le fonctionnement du timer3.

Exemple :

```
// Lancement du Timer3
tmr3_timer_a_start(drv_tmr3_1);
```

A9 : Générer et utiliser le fichier « swplatform.c » :

Après la compilation du projet Embarqué, des fichiers suivants sont générés :



Pour utiliser les fichiers générés, il faut les inclure dans le fichier top source « main.c » : par la déclaration suivante :
`#include « swplatform.h »`

Home | Devices | swplatform.h

```
#include "devices.h"

/* Extra project headers */
#include "instruments.h"
#include "ioport.h"
#include "led_info.h"

/* Sofware Services */
#include <interrupts.h>
#include <timers.h>
#include <timing.h>

/* Top Level Stack Items */
#include <drv_adc084s021.h>
#include <drv_instrument.h>
#include <drv_ioport.h>
#include <drv_led.h>
#include <drv_tmr3.h>
#include <drv_uart8.h>

/* Lower Level Stack Items */
#include <drv_spi.h>
#include <per_instrument.h>
#include <per_ioport.h>
#include <per_led.h>
#include <per_spi.h>
#include <per_tmr3.h>
#include <per_uart8.h>

/* Global variables to access Software Platform stacks */
extern led_t *        drv_led_1;
extern tmr3_t *       drv_tmr3_1;
extern uart8_t *      drv_uart8_1;
extern ioport_t *     drv_ioport_2;
extern instrument_t * drv_instrument_1;
extern ioport_t *     drv_ioport_1;
extern adc084s021_t * drv_adc084s021_1;

/* Initialize all stacks in the Software Platform */
extern void swplatform_init_stacks(void);
#endif
```

Home | Devices | swplatform.c

```
/*
 * Software Platform Generated File
 * --------------------------------
 */


#include "swplatform.h"

/* Global variables to access Software Platform stacks */
led_t *        drv_led_1;
tmr3_t *       drv_tmr3_1;
uart8_t *      drv_uart8_1;
ioport_t *     drv_ioport_2;
instrument_t * drv_instrument_1;
ioport_t *     drv_ioport_1;
adc084s021_t * drv_adc084s021_1;

/* Initialize all stacks in the Software Platform */
void swplatform_init_stacks(void)
{
    drv_led_1        = led_open(DRV_LED_1);
    drv_tmr3_1       = tmr3_open(DRV_TMR3_1);
    drv_uart8_1      = uart8_open(DRV_UART8_1);
    drv_ioport_2     = ioport_open(DRV_IOPORT_2);
    drv_instrument_1 = instrument_open(DRV_INSTRUMENT_1);
    drv_ioport_1     = ioport_open(DRV_IOPORT_1);
    drv_adc084s021_1 = adc084s021_open(DRV_ADC084S021_1);
}
```