

Enseignement CE 211 TP MICROCONTROLEUR STM8S GENERALITES

1 GENERALITES

Les applications pratiques de l'enseignement CE211 permettront de développer des applications mettant en œuvre le microcontrôleur **STM8S105C6** dans un environnement comportant plusieurs cartes ainsi que des équipements extérieurs.

Les développements logiciels se feront en langage C et vous feront découvrir un environnement de développement – IDE – comme **Integrated Development Environment** – fourni par la société ST Microelectronics : le **ST Visual Develop** qui est installé sur tous les PC des salles de TP.

Les documents nécessaires pour les TP sont rassemblés dans le classeur rouge présent sur chaque poste de travail. Le contenu de ce classeur sera complété au fur et à mesure en fonction des besoins liés aux sujets traités en TP.

La consultation de ces documents sous-entend qu'aucun document ne soit sorti des pochettes en plastique dans lesquelles ils se trouvent.

Les exercices pratiques portent sur la mise en œuvre des GPIO, des Timers et de l'UART du STM8S. La maîtrise du langage C pour des applications embarquées pilotées par le STM8 ainsi que l'utilisation du debugger de **ST Visual Develop** font aussi partie des apprentissages des TP.

Concernant le langage C vous respecterez les différentes règles de codage, de nommage et de formatage qui vous seront communiquées. Le C norme ANSI est utilisé.

Les travaux effectués ne donneront pas lieu à des comptes rendus, cependant vous remplirez votre **cahier de TP personnel** (détails plus loin) en suivant les indications qui vous seront données. C'est ce document ainsi que le travail effectué qui serviront de base à l'évaluation de chaque élève.

2 LA CARTE CIBLE

On appelle carte cible la structure physique sur laquelle se trouve le microcontrôleur chargé d'exécuter les programmes que vous aurez développés.

Cette structure a été définie spécifiquement pour les TP 2A et est originale dans la mesure où elle se compose de 2 cartes :

- **La carte de développement** qui contient le **STM8S105C6** avec son interface permettant le raccordement, par liaison USB, au PC de développement. C'est la **STM8S-DISCOVERY**.
Documentation : UM0817 du site st.com
- **La carte d'accueil**, sur laquelle la précédente est enfichée et qui contient différents dispositifs d'affichage dont l'afficheur 7 segments TIL321, des boutons-poussoirs, des embases de connectique, un buzzer, un potentiomètre linéaire, des circuits à interface i2C, etc. Seule d'adjonction de la **STM8S-DISCOVERY** permet d'exploiter les éléments de **la carte d'accueil**.

Un schéma de principe de la carte d'accueil sera présenté en séance. Il sera complété ultérieurement en fonction des sujets de TP.

3 EQUIPEMENTS NECESSAIRES

Pour réaliser les TP les équipements suivants seront mis à disposition sur chaque poste :

- PC avec l'environnement de développement **ST Visual Development** et le compilateur **COSMIC CXSTM8_4K**.
- Carte **STM8S Discovery** enfichée sur la **carte d'accueil**
- **Câble USB** permettant le téléchargement du code exécutable entre le PC et la **STM8S Discovery**.

- Le module d'entrées-sorties comportant 8 leds et 8 clés et 2 nappes avec connecteurs DIL16. Les 8 leds sont interfacées par des circuits logiques alimentés en 5 V. Les 8 clés sont interfacées par des anti-rebonds constitués de portes NAND. Raccordement du connecteur OUT sur l'embase identifiée CN10 (port C), raccordement du connecteur IN sur l'embase identifiée CN20 (port B).
- Cordon d'alimentation 5 v pour le module d'entrées-sorties.
- 2 boutons-poussoirs à 3 plots destinés à être raccordés sur les embases CN11 et CN12. Les anti-rebonds se trouvent sur la carte d'accueil, ceux-ci sont en service lorsque les cavaliers bleus [JP 401 et JP 403] sont en position 2-3.
- Câble (référence 29) de 3 conducteurs pour l'exploitation de la liaison série asynchrone RS232. A raccorder sur l'embase CN30.

4 DOCUMENTS UTILES

- Pour les 2 premières séances : chapitre 11 sur les GPIO du RM0016 du site **st.com**
- Document décrivant la structure GPIO et les définitions des différents bits d'un registre
- Schéma électrique de la partie interfaçage de l'afficheur 7 segments TIL321A
- Schéma de principe des éléments concernés par les exercices des TP

5 CAHIER DE TP PERSONNEL

Le rôle du cahier de TP est triple :

- il vous oblige à être rigoureux dans les développements que vous faites, à consigner les résultats obtenus ainsi que les difficultés que vous rencontrerez
- il vous permettra de retrouver des méthodes/remarques/résultats ultérieurement lors de **la séance d'évaluation de fin de semestre.**
- il servira aussi à l'enseignant pour l'évaluation des travaux que vous avez effectués.

Vous consignerez dans **votre cahier de TP personnel** :

- les explications fournies par l'enseignant au cours de la séance.
- pour chaque exercice : les algorithmes et organigrammes qui ont conduit à l'écriture du code
- les noms du(es) fichier(s) source contenant le code de l'exercice, ainsi que le nom du projet.
- les différentes méthodes, techniques ou remarques qui vous auront permis de mener à bien les différentes manipulations et d'aboutir aux résultats.
- les résultats obtenus et la façon dont vous avez validé chaque développement
- ce que vous avez appris à l'occasion des exercices effectués
- les difficultés rencontrées et les solutions apportées pour y remédier
- une conclusion et le bilan de la séance : exercices terminés, il reste, je prévois de revenir pour terminer l'exercice ..

Bien évidemment pour chaque séance vous rappellerez le thème des exercices et vous mentionnerez la date.

En aucun cas le cahier ne contient le code source en langage C.

6 METHODE DE DEVELOPPEMENT

Pour des développements en langage C pour microcontrôleurs 8 bits.

Ecrire du code/des instructions ne représente qu'une petite partie du temps consacré au développement.

Bien que le code source soit essentiel pour l'application celui-ci ne peut exister qu'après une phase d'analyse rigoureuse des besoins de l'application.

Les 7 étapes du développement :

- **1** - Formuler de façon claire et précise les différentes actions de l'application demandée en ayant rappelé le rôle des équipements raccordés au microcontrôleur.
- **2** - Etablir l'organigramme ou bien l'algorithme ou bien la séquence des actions que l'application doit effectuer. Cette phase d'analyse est la partie la plus importante du développement.
- **3** - Définir clairement les noms des variables (globales et locales) ou des tableaux dont vous aurez besoin. Leurs noms seront choisis de façon à être en rapport leur rôle, par exemple : *compteur_impulsions_BP_ext_1* ou *tab_ASCII_dates[26]*.
Définir les mots de remplacements correspondant aux constantes dont vous aurez besoin.
Par exemple : *GPIO_PIN_7* en remplacement de la valeur \$80 ou 0x80
ou *GPIO_PIN_0* en remplacement de la valeur \$01 ou 0x01
- **4** - Ecrire le code source en langage C en utilisant soit l'éditeur de texte de l'IDE soit le bloc notes. En respectant les règles de codage et de formatage qui seront précisées.
- **Le code doit être facilement lisible et compréhensible par toute autre personne devant poursuivre ou reprendre le développement.**
- Le début du fichier comportera obligatoirement un **bloc commentaires** mentionnant le nom du fichier, sa date de création, l'objet de l'application, les noms des auteurs puis les dates des modifications et la date de validation.
- Ne pas oublier d'inclure le fichier **stm8s.h** au début de votre fichier source.
Ce fichier ainsi que les 3 fichiers **stm8s_conf.h**, **stm8s_type.h** et **stm8s_GPIO.h** sont à récupérer dans **L:\informatique\TP informatique\A163\CE211\STM8** et à placer dans la partie include files du projet.
- **5** - Procéder à l'opération de compilation puis de génération du code exécutable au moyen de l'IDE. En cas d'erreur n'oubliez pas de consigner dans le cahier de TP la cause de l'erreur et la correction effectuée.
- **6** - Effectuer le téléchargement dans la mémoire Flash du STM8S10C6 de la **STM8S Discovery** puis faire exécuter le code.
- **7** - Vérifier le fonctionnement de l'application en le comparant à la description faite à l'étape 1. Pour cela il est peut-être nécessaire de définir des valeurs de test. Définissez-les.
Consigner les résultats dans le cahier de TP et justifier le bon fonctionnement de l'application.
En cas de non satisfaction détailler le problème et expliquer la solution mise en œuvre. Si la solution a nécessité de modifier les écrits de l'étape 2 consignez-la dans le cahier de TP.
Après correction du fichier source reprenez à partir de l'**étape 5**

7 INFORMATIONS GENERALES

Afin de permettre une utilisation aisée et une bonne compréhension des fichiers source les indications suivantes devront être respectées :

- tous vos fichiers se trouveront dans le répertoire **MonSTM8** que vous aurez créé sur le bureau du PC
- dans ce répertoire il y aura un dossier par thème de TP : GPIO, TIMER et UART.
- dans chaque dossier se trouvera le **Workspace** et les projets correspondant aux différents exercices.
- les noms des projets et des fichiers source contenant le main() auront un mot en commun et rappelleront la nature de l'exercice. Exemple : projet **exo1** fichier : **echo.c**

- n'utilisez qu'exceptionnellement des valeurs numériques explicites dans le corps des fonctions, les **#define** et **enum** sont indispensables. Exemple : **#define GPIO_MODE_SORTIE 0xFF**
- développez une fonction spécifique, même simple, chaque fois qu'une action précise peut être identifiée, chaque fonction sera décrite dans son en-tête.
- lors des modifications de bits dans un registre de contrôle ou de sortie il ne faut pas affecter les autres bits non concernés par l'action que vous voulez faire.
Exemple : **GPIOD->ODR = GPIOD->ODR | GPIO_PIN_7;**
- respectez scrupuleusement les noms des bits et des registres définis par ST, ils ont été définis dans le fichier **stm8s.h** et les fichiers **.h** associés aux périphériques que vous utilisez.
- incluez le fichier **stm8s.h** au début de votre fichier source. (récupérez ce fichier et le fichier **stm8s_conf.h** dans L:\informatique\TP informatique\A163\CE211\STM8
- lors de la séquence d'initialisation des registres il n'est pas utile d'accéder aux registres de configuration non utilisés. Leur contenu par défaut les rend « inactifs ».

L'exécution d'un programme se fait sous contrôle du debugger ST dont le fonctionnement sera présenté pendant la séance pratique.

Ce debugger est une aide précieuse pour la mise au point des programmes, il permet :

- de suivre l'exécution du programme instruction par instruction ou ligne par ligne en langage C
- de visualiser les instructions en assembleur,
- d'accéder aux contenus des registres du processeur et des emplacements mémoire, et d'en modifier les contenus
- d'accéder aux différents registres des ports et des périphériques et d'en modifier les contenus
- de faire des exécutions avec des points d'arrêt sur adresse (breakpoint)
- etc ...

Le projet, au sens de ST, est un fichier contenant **les noms** de tous les fichiers nécessaires à l'application.

Le fichier contenant le main : **echo.c** contient obligatoirement :

- **#include "stm8s.h"**

Si nécessaire: - **#include "xxxx.h"**

Le fichier contenant vos fonctions : **mes_fonctions.c** contient :

- **#include "stm8s.h"**
- **#include "mes_fonctions.h"** qui contient :
 - les **#define** des constantes nécessaires au fichier **mes_fonctions.c**

REMARQUE : seuls des fichiers .h peuvent être inclus dans un fichier.c

Carte d'accueil pour développements avec les microcontrôleurs STM8S

Caractéristiques et affectation des 6 Ports (septembre 2016)

Le microcontrôleur utilisé est le **STM8S105C6T6** dans un boîtier LQFP de 48 broches dont **38 lignes d'I/O** groupées en 6 ports de tailles inégales.

(4 lignes sont affectées à des usages précis : 2 pour UART2, 1 pour la liaison SWIM et 1 pour la led verte ; il en reste donc 34)

PORT A => 6 lignes			PORT B			=> 8 lignes		8 sources IT
Nom des lignes	high sink	Fonction alternative	Affectation carte d'accueil pour TP		Nom des lignes	Fonction alternative	2 ^{ème} Fonction alternative	Affectation carte d'accueil pour TP
PA1 / OSCIN		OSC in Quartz 16MHz	Support DIL CN10 bit de droite		PB0 / AIN0	Analog input 0	TIM1_CH1N	TIL dp HDLX_D0
PA2 / OSCOUT		OSC out Quartz 16MHz	Embase CN 22. Extension		PB1 / AIN1	Analog input 1	TIM1_CH2N	TIL seg.g HDLX
PA3		Timer 2 channel 3	LED rouge 4		PB2 / AIN2	Analog input 2	TIM1_CH3N	TIL seg.f HDLX
PA4	HS		LED rouge 3		PB3 / AIN3/TIM1_ETR	Analog input 3	TIM1_ETR	TIL seg.e HDLX
PA5	HS		LED rouge 2		PB4 / AIN4	Analog input 4	I2C_SCL	TIL seg.d HDLX
PA6	HS		LED rouge 1		PB5 / AIN5	Analog input 5	I2C_SDA	TIL seg.c HDLX
					PB6 / AIN6	Analog input 6		TIL seg.b HDLX
					PB7 / AIN7	Analog input 7		TIL seg.a /CU

PORT C => 7 lignes			PORT D			=> 8 lignes		8 sources IT
Nom des lignes	high sink	Fonction alternative	Affectation carte d'accueil pour TP		Nom des lignes	high sink	2 ^{ème} Fonction alternative	Affectation carte d'accueil pour TP
PC1 / TIM1_CH1	HS	Timer 1 channel 1	Bouton poussoir 4		PD0 / TIM3_CH2	HS	Timer 3 channel 2	LED 1 Discovery
PC2 / TIM1_CH2	HS	Timer 1 channel 2	Bouton poussoir 3		PD1 / SWIM	HS	SWIM data interface	téléchargement
PC3 / TIM1_CH3	HS	Timer 1 channel 3	Bouton poussoir 2		PD2 / TIM3_CH1	HS	Timer 3 channel 1	Embase HE10 CN22 Extension
PC4 / TIM1_CH4	HS	Timer 1 channel 4	Bouton poussoir 1		PD3 / TIM2_CH2	HS	Timer 2 channel 2	HDLX Signal/BL
PC5 / SPI_SCK	HS	SPI Clock	Embase HE10		PD4 / TIM2_CH1	HS	Timer 2 channel 1	Buzzer #.
PC6 / SPI_MOSI	HS	SPI Master Out	CN31, entrées et sorties série SPI 8 bits		PD5 / UART2_TX		UART2 data transmit	Liaison série ...
PC7 / SPI_MISO	HS	SPI Master In			PD6 / UART_RX		UART2 data receive	... RS 232
					PD7 / TLI		Top level interrupt	BP1 ext. et signal /INT des circuits à interface I2C

PORT E => 7 lignes				7 sources IT
Nom des lignes	high sink	Fonction alternative	Affectation carte d'accueil pour TP	
PE0 / CLK_CCO	HS	Config. clock output	Commutateur INT1, alim pour TIL321 et test fréquence	
PE1 / I2C_SCL		I2C Clock	Embase HE10 CN31, RTC et capteur de température	
PE2 / I2C_SDA		I2C data	HDLX1 signal W/R du module 4	
PE3 / TIM1_BKIN		Timer 1 break input	digits de droite	
PE5 / SPI_NSS		SPI master select	HDLX2 signal W/R du module 4	
PE6 / AIN9		Analog input 9	digits de gauche	
PE7 / AIN8		Analog input 8	Potentiomètre # BP 2 ext. pour IT #	

[illegible]

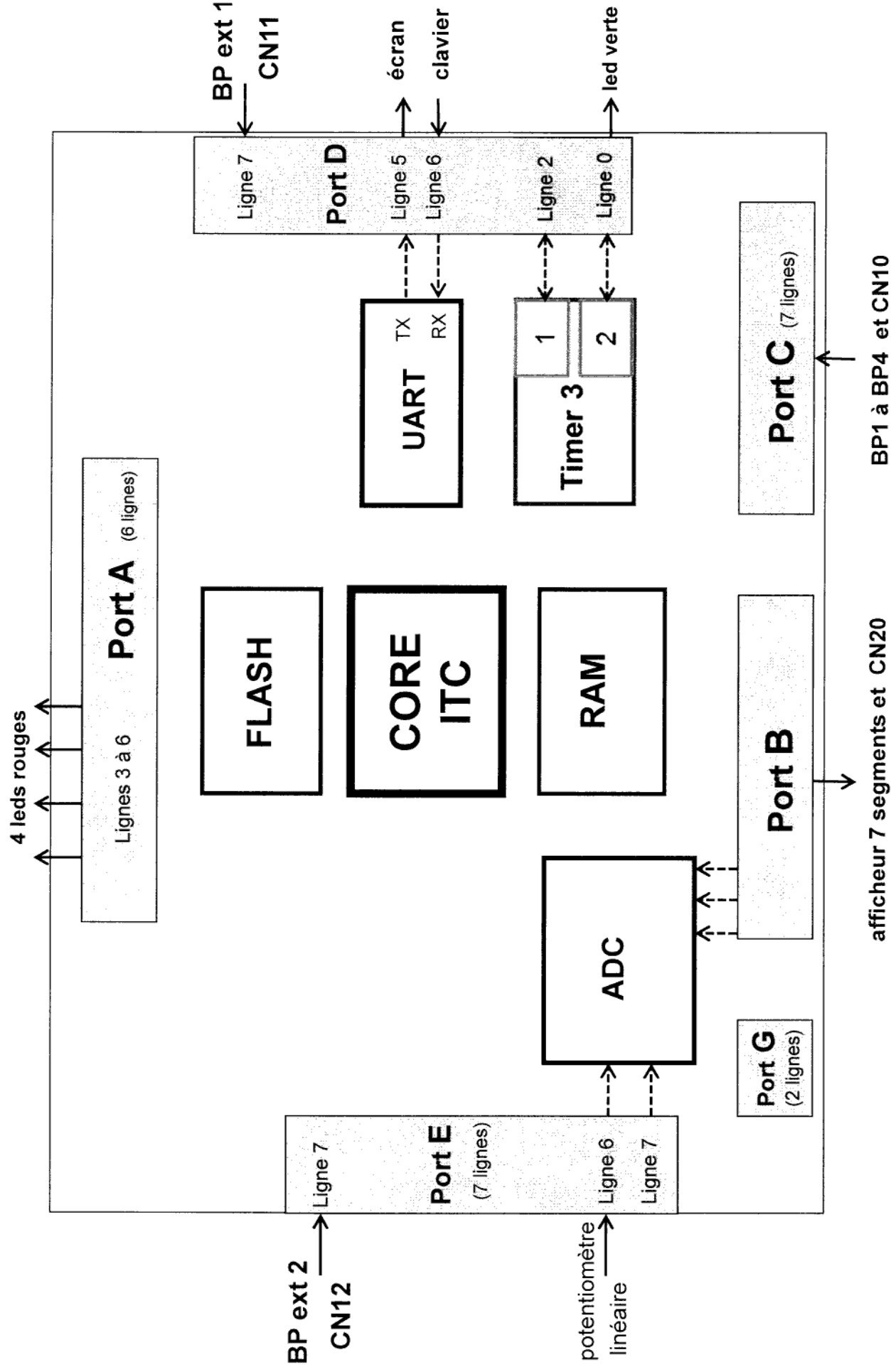
: lignes raccordées également au connecteur HE10 pour carte extension. Ce qui conduit, pour les entrées, à la nécessité d'un système de commutation par cavaliers

Connecteur HE10 pour carte extension (moteur ou puissance)

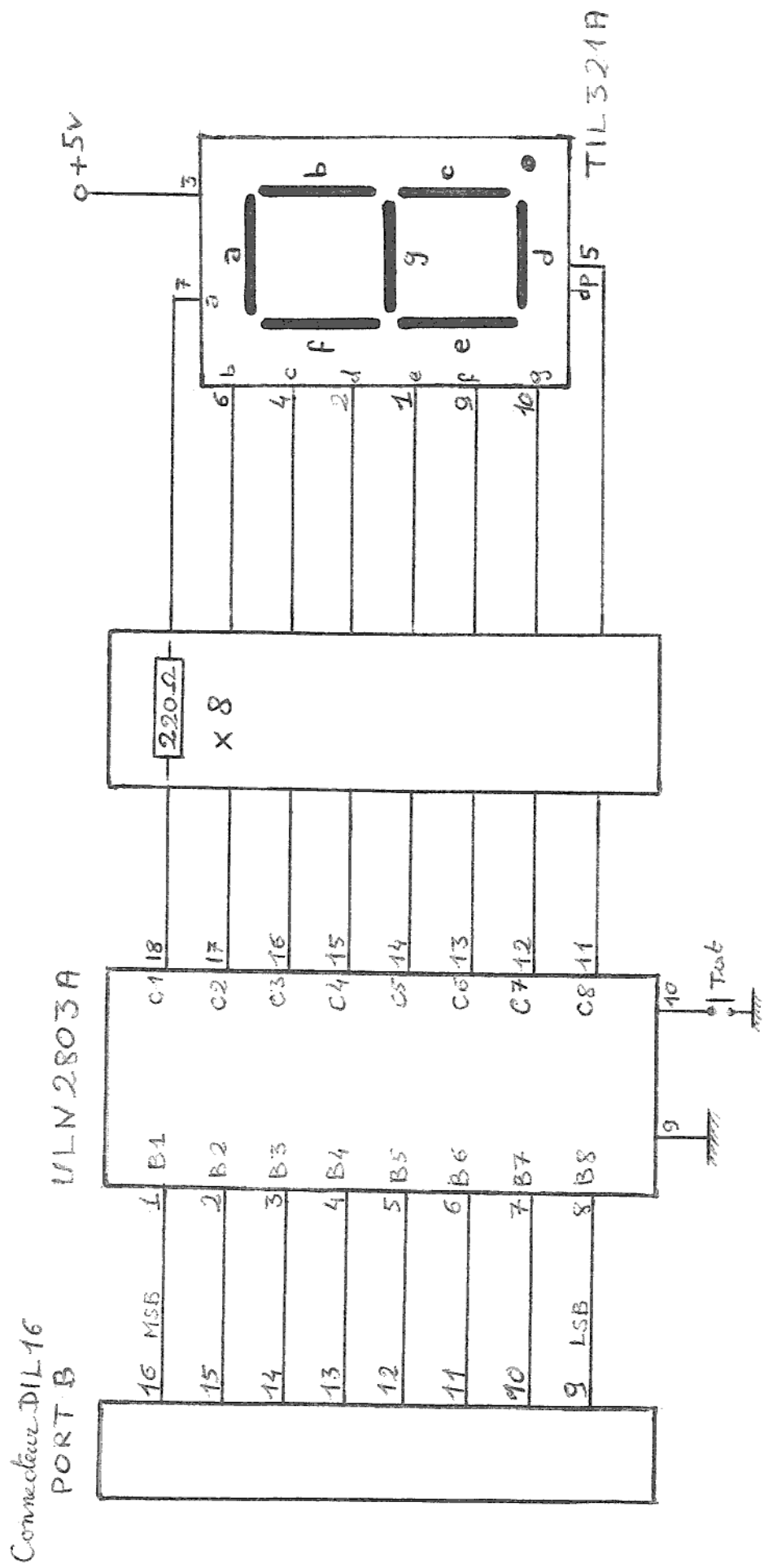
	Alimentation 5v	1	2	Masse	
Tim 3 channel 1	Port D 2 – commande moteur sens 1	3	4	Port D 3 – commande moteur sens 2	Tim 2 channel 2
	Masse	5	6	Port A 2 -	
Tim 2 channel 3	Port A 3 – Led rouge 4	7	8	Port A 4 – Led rouge 3	
Analog Input 8	Port E 6 – entrée analogique	9	10	Port E 7 – entrée analogique	Analog Input 9

MICROCONTROLEUR STM8S105C6

utilisation des ports par la carte d'accueil pour TP



Carte extension Entrées - Sorties



DEVELOPPEMENTS LOGICIELS POUR STM8
DEVELOPPEMENT EN LANGAGE C
GPIO : la structure et les définitions des bits des registres

Extrait du fichier **stm8s.h** fourni par ST.

```
typedef struct GPIO_struct
{
    vu8 ODR;    /*!< Output Data Register */
    vu8 IDR;    /*!< Input Data Register */
    vu8 DDR;    /*!< Data Direction Register */
    vu8 CR1;    /*!< Configuration Register 1 */
    vu8 CR2;    /*!< Configuration Register 2 */
}
GPIO_TypeDef;
```

Cette structure est à utiliser pour chacun des 6 ports du STM8S105C6 que vous utilisez.

Exemple : **GPIOB->DDR** pour identifier le registre DDR du port B

Extrait du fichier **stm8s_gpio.h** fourni par ST.

Définition des bits d'un port

```
typedef enum
{
    GPIO_PIN_0 = ((u8)0x01),    /*!< Pin 0 selected */
    GPIO_PIN_1 = ((u8)0x02),    /*!< Pin 1 selected */
    GPIO_PIN_2 = ((u8)0x04),    /*!< Pin 2 selected */
    GPIO_PIN_3 = ((u8)0x08),    /*!< Pin 3 selected */
    GPIO_PIN_4 = ((u8)0x10),    /*!< Pin 4 selected */
    GPIO_PIN_5 = ((u8)0x20),    /*!< Pin 5 selected */
    GPIO_PIN_6 = ((u8)0x40),    /*!< Pin 6 selected */
    GPIO_PIN_7 = ((u8)0x80),    /*!< Pin 7 selected */
    GPIO_PIN_LNIB = ((u8)0x0F), /*!< Low nibble pins selected */
    GPIO_PIN_HNIB = ((u8)0xF0), /*!< High nibble pins selected */
    GPIO_PIN_ALL = ((u8)0xFF)   /*!< All pins selected */
}
GPIO_Pin_TypeDef;
```

Il est possible de redéfinir un nom personnalisé à votre application à partir de cette définition.

Exemple : #define **BOUTON_POUSSOIR_1** GPIO_PIN_7

Pour une application dans laquelle un bouton-poussoir a été raccordé sur la ligne 7 d'un port.

DEVELOPPEMENTS LOGICIELS POUR STM8
DEVELOPPEMENT EN LANGAGE C
TIMER4 : la structure et les définitions des masques des bits des registres

Extrait du fichier stm8s.h fourni par ST.

```
typedef struct TIM4_struct
{
    vu8 CR1; /*!< control register 1 */
    vu8 IER; /*!< interrupt enable register */
    vu8 SR1; /*!< status register 1 */
    vu8 EGR; /*!< event generation register */
    vu8 CNTR; /*!< counter register */
    vu8 PSCR; /*!< prescaler register */
    vu8 ARR; /*!< auto-reload register */
}
TIM4_TypeDef;
```

TIM4_Registers_Bits_Definition

```
#define TIM4_CR1_ARPE ((u8)0x80) /*!< Auto-Reload Preload Enable mask. */
#define TIM4_CR1_OPM ((u8)0x08) /*!< One Pulse Mode mask. */
#define TIM4_CR1_URS ((u8)0x04) /*!< Update Request Source mask. */
#define TIM4_CR1_UDIS ((u8)0x02) /*!< Update DIisable mask. */
#define TIM4_CR1_CEN ((u8)0x01) /*!< Counter Enable mask. */

#define TIM4_IER_UIE ((u8)0x01) /*!< Update Interrupt Enable mask. */

#define TIM4_SR1_UIF ((u8)0x01) /*!< Update Interrupt Flag mask. */

#define TIM4_EGR_UG ((u8)0x01) /*!< Update Generation mask. */

#define TIM4_CNTR_CNT ((u8)0xFF) /*!< Counter Value (LSB) mask. */

#define TIM4_PSCR_PSC ((u8)0x07) /*!< Prescaler Value mask. */

#define TIM4_ARR_ARR ((u8)0xFF) /*!< Autoreload Value mask. */
```

DEVELOPPEMENTS LOGICIELS POUR STM8
DEVELOPPEMENT EN LANGAGE C
TIMER 2 : la structure et les définitions des bits des registres

 Extrait du fichier **stm8s.h** fourni par ST.

```
typedef struct TIM2_struct
{
    vu8 CR1; /*!< control register 1 */
    vu8 IER; /*!< interrupt enable register */
    vu8 SR1; /*!< status register 1 */
    vu8 SR2; /*!< status register 2 */
    vu8 EGR; /*!< event generation register */

    vu8 CCMR1; /*!< CC mode register 1 */
    vu8 CCMR2; /*!< CC mode register 2 */
    vu8 CCMR3; /*!< CC mode register 3 */
    vu8 CCER1; /*!< CC enable register 1 */
    vu8 CCER2; /*!< CC enable register 2 */
    vu8 CNTRH; /*!< counter high */
    vu8 CNTRL; /*!< counter low */

    vu8 PSCR; /*!< prescaler register */
    vu8 ARRH; /*!< auto-reload register high */
    vu8 ARRL; /*!< auto-reload register low */

    vu8 CCR1H; /*!< capture/compare register 1 high */
    vu8 CCR1L; /*!< capture/compare register 1 low */

    vu8 CCR2H; /*!< capture/compare register 2 high */
    vu8 CCR2L; /*!< capture/compare register 2 low */

    vu8 CCR3H; /*!< capture/compare register 3 high */
    vu8 CCR3L; /*!< capture/compare register 3 low */
}
TIM2_TypeDef;
```

TIM2_Registers_Bits_Definition

```
/*CR1*/
#define TIM2_CR1_ARPE ((u8)0x80) /*!< Auto-Reload Preload Enable mask. */
#define TIM2_CR1_OPM ((u8)0x08) /*!< One Pulse Mode mask. */
#define TIM2_CR1_URS ((u8)0x04) /*!< Update Request Source mask. */
#define TIM2_CR1_UDIS ((u8)0x02) /*!< Update DIisable mask. */
#define TIM2_CR1_CEN ((u8)0x01) /*!< Counter Enable mask. */

/*IER*/
#define TIM2_IER_CC3IE ((u8)0x08) /*!< Capture/Compare 3 Interrupt Enable mask. */
#define TIM2_IER_CC2IE ((u8)0x04) /*!< Capture/Compare 2 Interrupt Enable mask. */
#define TIM2_IER_CC1IE ((u8)0x02) /*!< Capture/Compare 1 Interrupt Enable mask. */
#define TIM2_IER_UIE ((u8)0x01) /*!< Update Interrupt Enable mask. */
```

```
/*SR1*/
#define TIM2_SR1_CC3IF ((u8)0x08) /*!< Capture/Compare 3 Interrupt Flag mask. */
#define TIM2_SR1_CC2IF ((u8)0x04) /*!< Capture/Compare 2 Interrupt Flag mask. */
#define TIM2_SR1_CC1IF ((u8)0x02) /*!< Capture/Compare 1 Interrupt Flag mask. */
#define TIM2_SR1_UIF ((u8)0x01) /*!< Update Interrupt Flag mask. */

/*SR2*/
#define TIM2_SR2_CC3OF ((u8)0x08) /*!< Capture/Compare 3 Overcapture Flag mask. */
#define TIM2_SR2_CC2OF ((u8)0x04) /*!< Capture/Compare 2 Overcapture Flag mask. */
#define TIM2_SR2_CC1OF ((u8)0x02) /*!< Capture/Compare 1 Overcapture Flag mask. */

/*EGR*/
#define TIM2_EGR_CC3G ((u8)0x08) /*!< Capture/Compare 3 Generation mask. */
#define TIM2_EGR_CC2G ((u8)0x04) /*!< Capture/Compare 2 Generation mask. */
#define TIM2_EGR_CC1G ((u8)0x02) /*!< Capture/Compare 1 Generation mask. */
#define TIM2_EGR_UG ((u8)0x01) /*!< Update Generation mask. */

/*CCMR*/
#define TIM2_CCMR_ICxPSC ((u8)0x0C) /*!< Input Capture x Prescaler mask. */
#define TIM2_CCMR_ICxF ((u8)0xF0) /*!< Input Capture x Filter mask. */
#define TIM2_CCMR_OCM ((u8)0x70) /*!< Output Compare x Mode mask. */
#define TIM2_CCMR_OCxPE ((u8)0x08) /*!< Output Compare x Preload Enable mask. */
#define TIM2_CCMR_CCxS ((u8)0x03) /*!< Capture/Compare x Selection mask. */

/*CCER1*/
#define TIM2_CCER1_CC2P ((u8)0x20) /*!< Capture/Compare 2 output Polarity mask. */
#define TIM2_CCER1_CC2E ((u8)0x10) /*!< Capture/Compare 2 output enable mask. */
#define TIM2_CCER1_CC1P ((u8)0x02) /*!< Capture/Compare 1 output Polarity mask. */
#define TIM2_CCER1_CC1E ((u8)0x01) /*!< Capture/Compare 1 output enable mask. */

/*CCER2*/
#define TIM2_CCER2_CC3P ((u8)0x02) /*!< Capture/Compare 3 output Polarity mask. */
#define TIM2_CCER2_CC3E ((u8)0x01) /*!< Capture/Compare 3 output enable mask. */

/*CNTR*/
#define TIM2_CNTRH_CNT ((u8)0xFF) /*!< Counter Value (MSB) mask. */
#define TIM2_CNTRL_CNT ((u8)0xFF) /*!< Counter Value (LSB) mask. */

/*PSCR*/
#define TIM2_PSCR_PSC ((u8)0xFF) /*!< Prescaler Value (MSB) mask. */

/*ARR*/
#define TIM2_ARRH_ARR ((u8)0xFF) /*!< Autoreload Value (MSB) mask. */
#define TIM2_ARRL_ARR ((u8)0xFF) /*!< Autoreload Value (LSB) mask. */
```

DEVELOPPEMENTS LOGICIELS POUR STM8
DEVELOPPEMENT EN LANGAGE C
TIMER3 : la structure et les définitions des bits des registres

Extrait du fichier **stm8s.h** fourni par ST.

```
typedef struct TIM3_struct
{
    vu8 CR1; /*!< control register 1 */
    vu8 IER; /*!< interrupt enable register */
    vu8 SR1; /*!< status register 1 */
    vu8 SR2; /*!< status register 2 */
    vu8 EGR; /*!< event generation register */
    vu8 CCMR1; /*!< CC mode register 1 */
    vu8 CCMR2; /*!< CC mode register 2 */
    vu8 CCER1; /*!< CC enable register 1 */
    vu8 CNTRH; /*!< counter high */
    vu8 CNTRL; /*!< counter low */
    vu8 PSCR; /*!< prescaler register */

    vu8 ARRH; /*!< auto-reload register high */
    vu8 ARRL; /*!< auto-reload register low */
    vu8 CCR1H; /*!< capture/compare register 1 high */
    vu8 CCR1L; /*!< capture/compare register 1 low */
    vu8 CCR2H; /*!< capture/compare register 2 high */
    vu8 CCR2L; /*!< capture/compare register 2 low */
}
TIM3_TypeDef;
```

TIM3_Registers_Bits_Definition (liste partielle hors CCMR, CCER)

```
/*CR1*/
#define TIM3_CR1_ARPE ((u8)0x80) /*!< Auto-Reload Preload Enable mask. */
#define TIM3_CR1_OPM ((u8)0x08) /*!< One Pulse Mode mask. */
#define TIM3_CR1_URS ((u8)0x04) /*!< Update Request Source mask. */
#define TIM3_CR1_UDIS ((u8)0x02) /*!< Update DIisable mask. */
#define TIM3_CR1_CEN ((u8)0x01) /*!< Counter Enable mask. */

/*IER*/
#define TIM3_IER_CC2IE ((u8)0x04) /*!< Capture/Compare 2 Interrupt Enable mask. */
#define TIM3_IER_CC1IE ((u8)0x02) /*!< Capture/Compare 1 Interrupt Enable mask. */
#define TIM3_IER_UIE ((u8)0x01) /*!< Update Interrupt Enable mask. */

/*SR1*/
#define TIM3_SR1_CC2IF ((u8)0x04) /*!< Capture/Compare 2 Interrupt Flag mask. */
#define TIM3_SR1_CC1IF ((u8)0x02) /*!< Capture/Compare 1 Interrupt Flag mask. */
#define TIM3_SR1_UIF ((u8)0x01) /*!< Update Interrupt Flag mask. */

/*CNTR*/
#define TIM3_CNTRH_CNT ((u8)0xFF) /*!< Counter Value (MSB) mask. */
#define TIM3_CNTRL_CNT ((u8)0xFF) /*!< Counter Value (LSB) mask. */

/*PSCR*/
#define TIM3_PSCR_PSC ((u8)0xFF) /*!< Prescaler Value (MSB) mask. */
```

DEVELOPPEMENTS LOGICIELS POUR STM8**DEVELOPPEMENT EN LANGAGE C****UART2 : la structure et les définitions des masques des bits des registres**

Extrait du fichier stm8s.h fourni par ST.

```
typedef struct UART2_struct
{
    vu8 SR; /*!< UART2 status register */
    vu8 DR; /*!< UART2 data register */
    vu8 BRR1; /*!< UART2 baud rate register */
    vu8 BRR2; /*!< UART2 DIV mantissa[11:8] SCIDIV fraction */
    vu8 CR1; /*!< UART2 control register 1 */
    vu8 CR2; /*!< UART2 control register 2 */
    vu8 CR3; /*!< UART2 control register 3 */
    vu8 CR4; /*!< UART2 control register 4 */
    vu8 CR5; /*!< UART2 control register 5 */
    vu8 CR6; /*!< UART2 control register 6 */
    vu8 GTR; /*!< UART2 guard time register */
    vu8 PSCR; /*!< UART2 prescaler register */
}
UART2_TypeDef;
```

UART2_Registers_Bits_Definition

```
#define UART2_SR_TXE ((u8)0x80) /*!< Transmit Data Register Empty mask */
#define UART2_SR_TC ((u8)0x40) /*!< Transmission Complete mask */
#define UART2_SR_RXNE ((u8)0x20) /*!< Read Data Register Not Empty mask */
#define UART2_SR_IDLE ((u8)0x10) /*!< IDLE line detected mask */
#define UART2_SR_OR ((u8)0x08) /*!< OverRun error mask */
#define UART2_SR_NF ((u8)0x04) /*!< Noise Flag mask */
#define UART2_SR_FE ((u8)0x02) /*!< Framing Error mask */
#define UART2_SR_PE ((u8)0x01) /*!< Parity Error mask */

#define UART2_BRR1_DIVM ((u8)0xFF) /*!< LSB mantissa of UART2DIV [7:0] mask */
#define UART2_BRR2_DIVM ((u8)0xF0) /*!< MSB mantissa of UART2DIV [11:8] mask */
#define UART2_BRR2_DIVF ((u8)0x0F) /*!< Fraction bits of UART2DIV [3:0] mask */

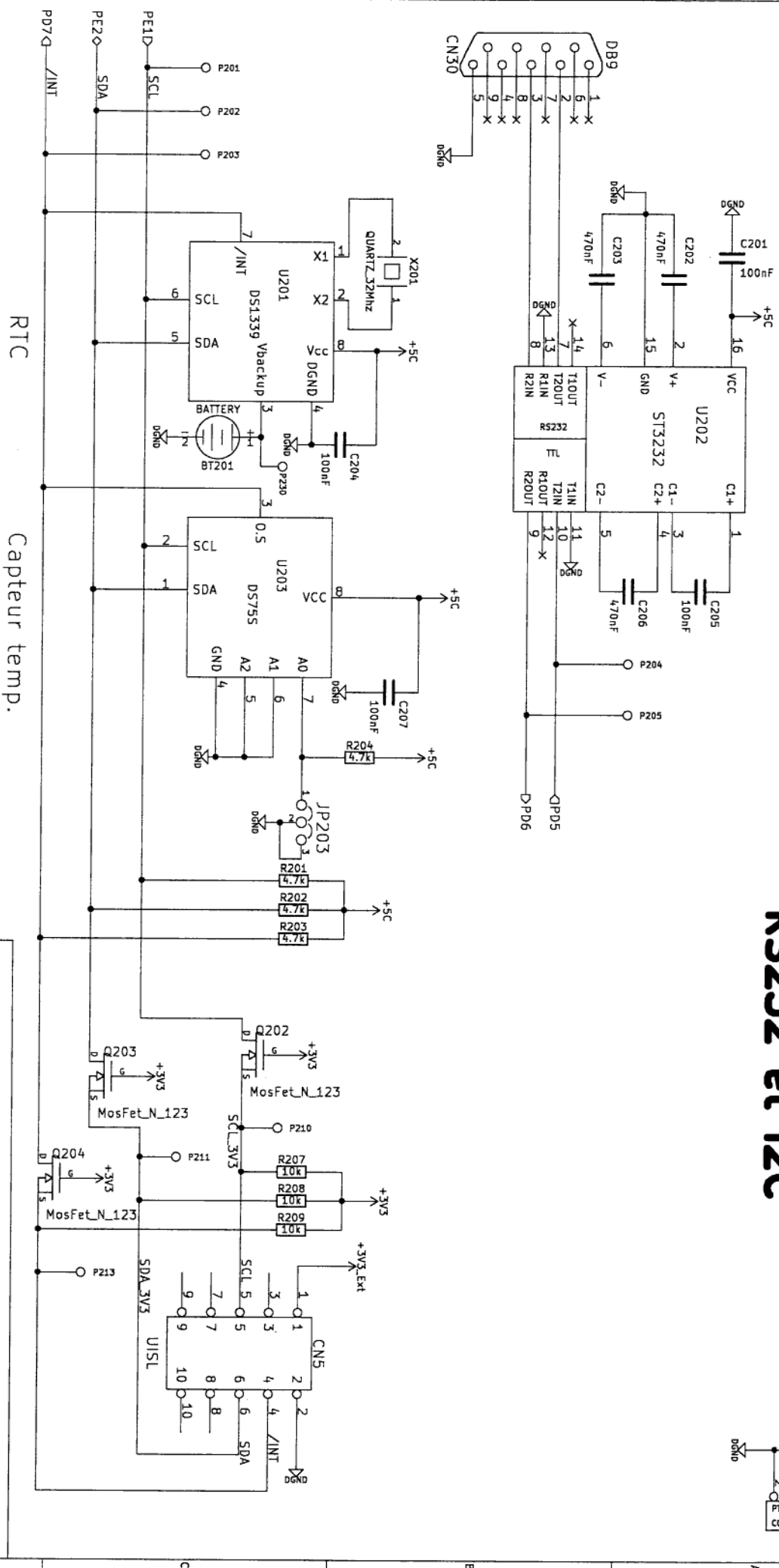
#define UART2_CR1_R8 ((u8)0x80) /*!< Receive Data bit 8 */
#define UART2_CR1_T8 ((u8)0x40) /*!< Transmit data bit 8 */
#define UART2_CR1_UARTD ((u8)0x20) /*!< UART2 Disable (for low power consumption) */
#define UART2_CR1_M ((u8)0x10) /*!< Word length mask */
#define UART2_CR1_WAKE ((u8)0x08) /*!< Wake-up method mask */
#define UART2_CR1_PCEN ((u8)0x04) /*!< Parity Control Enable mask */
#define UART2_CR1_PS ((u8)0x02) /*!< UART2 Parity Selection */
#define UART2_CR1_PIEN ((u8)0x01) /*!< UART2 Parity Interrupt Enable mask */
```

```
#define UART2_CR2_TIEN ((u8)0x80) /*!< Transmitter Interrupt Enable mask */
#define UART2_CR2_TCIEN ((u8)0x40) /*!< TransmissionComplete Interrupt Enable mask */
#define UART2_CR2_RIEN ((u8)0x20) /*!< Receiver Interrupt Enable mask */
#define UART2_CR2_ILIEN ((u8)0x10) /*!< IDLE Line Interrupt Enable mask */
#define UART2_CR2_TEN ((u8)0x08) /*!< Transmitter Enable mask */
#define UART2_CR2_REN ((u8)0x04) /*!< Receiver Enable mask */
#define UART2_CR2_RWU ((u8)0x02) /*!< Receiver Wake-Up mask */
#define UART2_CR2_SBK ((u8)0x01) /*!< Send Break mask */

#define UART2_CR3_LINEN ((u8)0x40) /*!< Alternate Function output mask */
#define UART2_CR3_STOP ((u8)0x30) /*!< STOP bits [1:0] mask */
#define UART2_CR3_CKEN ((u8)0x08) /*!< Clock Enable mask */
#define UART2_CR3_CPOL ((u8)0x04) /*!< Clock Polarity mask */
#define UART2_CR3_CPHA ((u8)0x02) /*!< Clock Phase mask */
#define UART2_CR3_LBCL ((u8)0x01) /*!< Last Bit Clock pulse mask */
```


Communications en série

RS232 et I2C



Carte accueil TP

ESISAR

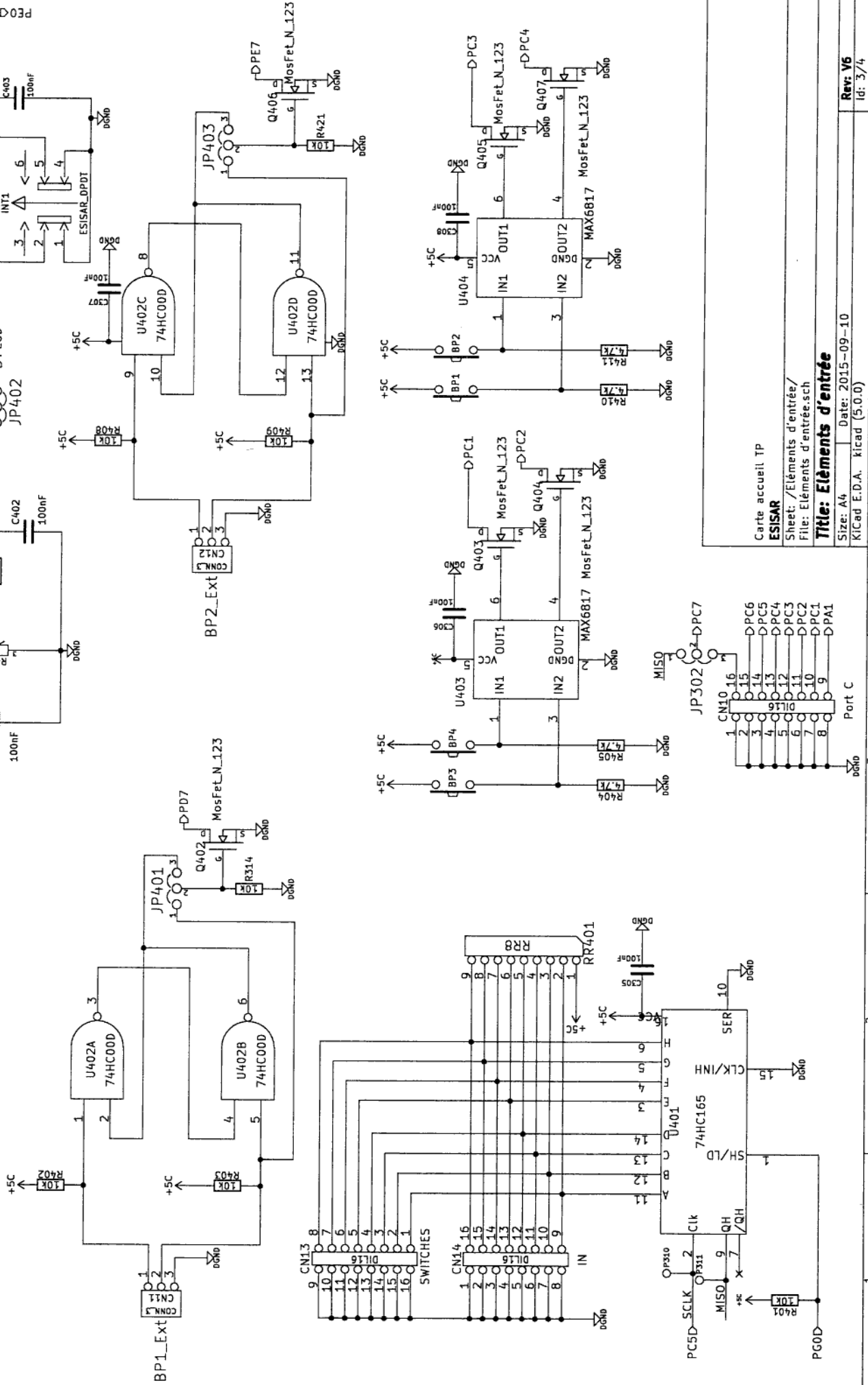
Sheet: /Communications en série/
File: Communications en série.sch

Title: Communications en série RS232 I2C

Size: A4 Date: 2015-09-10
Kicad E.D.A. kicad (5.0.0)

Rev: V6
Id: 2/4

Eléments d'entrée



Carte accueil TP

ESISAR

Sheet: /Eléments d'entrée/

File: Eléments d'entrée.sch

Title: Eléments d'entrée

Size: A4

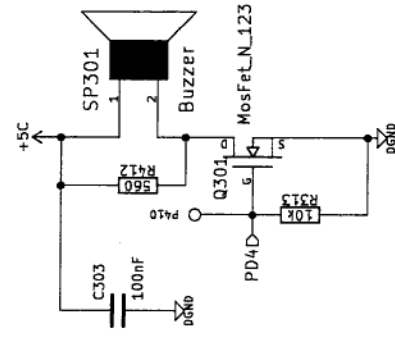
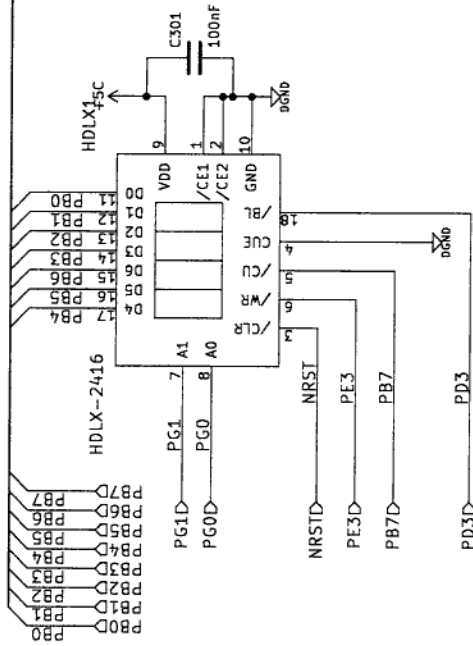
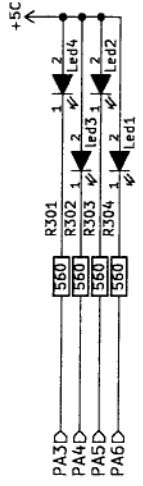
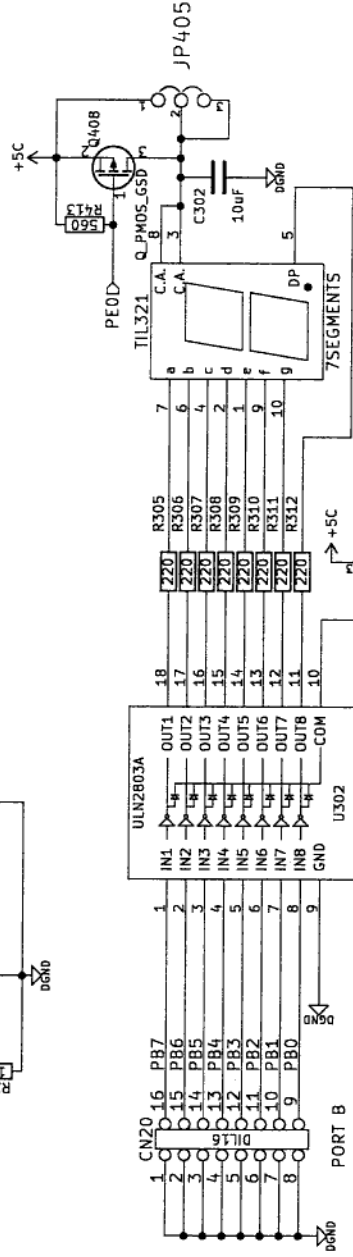
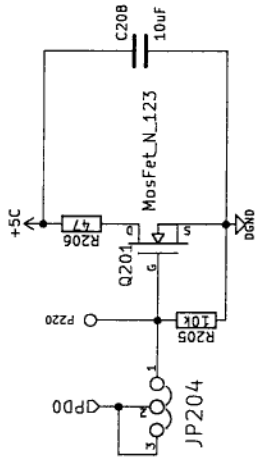
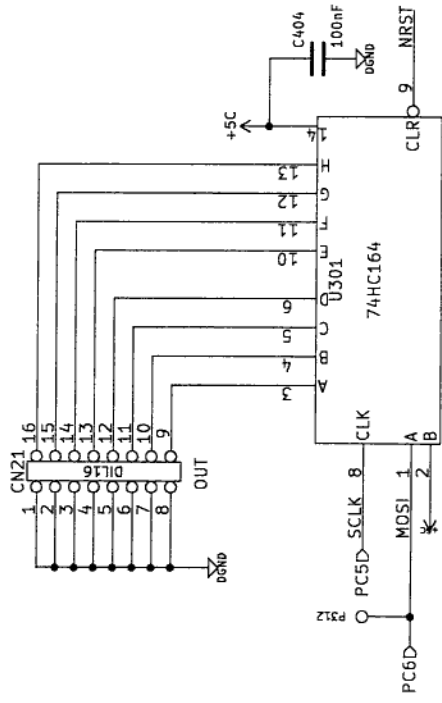
Date: 2015-09-10

Kicad E.D.A. kicad (5.0.0)

Rev: V6

Id: 3/4

Eléments de sortie



Carte accueil TP

ESISAR

Sheet: /Elements de sortie/

File: Elements de sortie.sch

Title: Elements de sortie

Size: A4 Date: 2015-09-10

KiCad E.D.A. kicad (5.0.0)

Rev: V6

Id: 4/4

TP INTERRUPTIONS STM8

MISE EN ŒUVRE DES INTERRUPTIONS

Ce qu'il faut savoir sur les interruptions du STM8

La mise en œuvre des interruptions nécessite d'agir à 4 niveaux différents :

- la fonction interruption
- la table des vecteurs
- les bits de masques d'interruptions
- l'autorisation globale des interruptions au niveau du processeur

La fonction interruption représente l'ensemble des traitements devant être effectués par le processeur lors de l'arrivée de l'évènement ou interruption.

Cette fonction particulière, développée par le programmeur, est caractérisée par plusieurs choses :

- elle ne reçoit pas de paramètre et ne renvoie pas de valeur de retour
- elle n'est pas appelée par une instruction
- sa durée d'exécution doit être la plus courte possible afin de ne pas ralentir le programme principal et d'empêcher la prise en compte d'autres demandes d'interruptions de même niveau de priorité. Les actions effectuées se limiteront aux manipulations concernant la donnée échangée entre le processeur et le périphérique et à des actions sur certains indicateurs ou variables.
- Son adresse d'implantation en mémoire (déterminée par l'éditeur de lien) doit être placée dans la table des vecteurs à l'emplacement qui convient. Cette table se trouve dans le fichier **stm8_interrupt_vector.c** du projet sur lequel vous travaillez.
- Son nom est précédé du mot **@ interrupt**. Exemple : **@ interrupt saisie_donnee(void)**

La table des vecteurs est à remplir au moment du développement

Vous devrez commencer par identifier le numéro de l'interruption qui vous conduira à la connaissance de l'emplacement du vecteur à remplacer. Pour cela vous consulterez la liste des vecteurs correspondant au STM8S105 qui se trouve dans la Data Sheet du composant.

Dans la table des vecteurs, avant toute modification, tous les vecteurs (sauf celui du RESET) sont définis par : `{0x82, NonHandledInterrupt}, /* irq x */` irq x est le numéro de l'interruption.

Vous aurez à remplacer le contenu actuel `NonHandledInterrupt` par le nom de la fonction interruption.

Exemple : l'interruption associée aux lignes d'entrées du port E a le **numéro 7**. Le nom que vous avez donné à la fonction interruption est par exemple : **saisie_donnee(void)**.

Dans la table des vecteurs le nouveau contenu correspondant à la ligne de **irq 7** devra être :

{0x82, (interrupt_handler_t) saisie_donnee}, /* irq7 */

Au début de ce fichier vous devez aussi préciser que le corps de la fonction interruption ne se trouve pas dans ce même fichier mais dans un autre fichier => **extern saisie_donnee(void) ;**

Les bits de masques d'interruptions font partie des registres de contrôle des ports ou des périphériques.

Vous devez rechercher, pour chaque périphérique, le bit correspondant à la source d'interruption que vous utilisez. Retirer le bit de masque correspond à mettre le bit correspondant à 1.

Exemples : le bit **RIEN** du registre UART2_CR2 pour la réception de caractères de l'UART2.

le bit **MSB** du registre GPIOE_CR2 pour l'entrée correspondant à BP_exterieur_2 (raccordé à la ligne 7)

Cette action est à faire dans la séquence d'initialisation.

L'autorisation globale des interruptions consiste à rendre le processeur interruptible.

Par défaut le processeur est non interruptible

Pour rendre le processeur interruptible il faut lui faire exécuter l'instruction assembleur : RIM.

Afin de pouvoir insérer une instruction assembleur dans un fichier pour le langage C vous devrez utiliser la syntaxe suivante : **_asm(« rim\n ») ;** Cette action est à faire en fin de la séquence d'initialisation.