Manuel utilisateur de la bibliothèque GASSP72, dérivée des drivers dynamiques STM32

Cette bibliothèque contient un sous-ensemble des fonctions de drivers dynamiques, en version « **ff** » ***(float-free)***, plus la fonction d'initialisation d'horloge statique ***Init_Clock_System();***

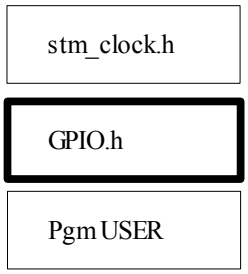
Le header **`gassp72.h`** contient un condensé des headers des éléments du projet drivers dynamiques.

Micro-contrôleur: STM32F103RB et STM32F107VCT6

Auteurs: T. ROCACHER, S. DI MERCURIO, J-L NOULLET

Version : 7 « FF »

NB : Pour que les fonctions soient compatibles avec le STM32F107VCT6, préciser la clé de compilation **STM32F107**.
Par défaut, les fonctions s'adressent au STM32F103RB8

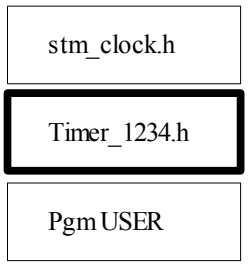


GPIO.c

Gère les périphériques GPIO
Voir *GPIO.h* pour plus d'infos.

Param entrée	**Valeur**	**Param sortie**	**Valeur**	**Description**
Fonction : char** GPIO_Configure**(GPIO_TypeDef *** Port**, char** Broche**, char** Sens**, char** Techno**);				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC (GPIOD , GPIOE pour STM107 : ajouter la clé de compilation STM32F107)**	Char	Renvoie **0** si tout est OK, et **1** s'il y a un problème (plage d'entrée non repsectée)	La fonction initialise n'importe quelle broche de port (entrée, sortie, techno....) Exemple : // *GPIO_Configure(*GPIOB, 8, OUTPUT, OUTPUT_PPULL); Place le bit 8 du port B en sortie Push-pull
Char Broche	**0 à 15**			
Char Sens	**INPUT** ou **OUTPUT** (define associé)			
Char Techno	**ANALOG** **INPUT_FLOATING** **INPUT_PULL_DOWN_UP** **OUTPUT_PPULL** **OUTPUT_OPDRAIN** **ALT_PPULL** **ALT_OPDRAIN** (define associé)			
Macro :** GPIO_Set**(GPIO,Broche) *(pour info : #define GPIO_Set(GPIO,Broche) GPIO->ODR=(GPIO->ODR)|(0x01<<Broche))*				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC**	void		La macro remplace de manière lisible une seule instruction C qui permet de fixer la broche précisée à '1'. Ex : GPIO_Set(GPIOB,8); // place la sortie 8 du GPIO B à 1
Char Broche	**0 à 15**			
Macro :** GPIO_Clear**(GPIO,Broche) *(pour info : #define GPIO_Clear(GPIO,Broche) GPIO->ODR=(GPIO->ODR)&~(0x01<<Broche))*				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC**	void		La macro remplace de manière lisible une seule instruction C qui permet de fixer la broche précisée à '0'. Ex : GPIO_Clear(GPIOB,8); // place la sortie 8 du GPIO B à 0
Char Broche	**0 à 15**			
Macro :** GPIO_Toggle**(GPIO,Broche) !! non testée *(pour info : #define GPIO_Toggle(GPIO,Broche) GPIO->ODR=(GPIO->ODR)^(0x01<<Broche))*				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC**	void		La macro remplace de manière lisible une seule instruction C qui permet de faire changer l'état de la broche considérée.
Char Broche	**0 à 15**			
Macro :** GPIO_Write**(GPIO,Broche, Val)				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC**	void		Cette macro est équivalente à GPIO_Clear et GPIO_Set, mais dans ce cas, c'est la variable *val*, suivant qu'elle est à 1 ou à 0, qui provoquera le clear ou set. Cette macro prend tout sons sens lors d'une test bianire et d'une affectation en suivant sur une
Char Broche	**0 à 15**			

				broche de port (pas besoin d'opérer un test pour ensuite faire explicitement un set ou un clear).
Macro : *GPIO_Write_Multi_Bits(**GPIO,Masque32Bits**)* ***!! non testée*** *(pour info :#define GPIO_Write_Multi_Bits(GPIO,Masque32Bits) (GPIO->BSRR)=Masque32Bits)*				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC**	void		Préciser le masque comme suit : Masque 32 bits = | 16 bits Reset | 16 bits Set | Chacun des 16 bits est dans l'ordre des broches du GPIO : Br15 ... Br0. Exemple : on veut mettre à 1 le bit 15 et mettre le bit 6 à 0, la valeur du masque sera (binaire puis hex): Masque32Bits = 0000 0000 0010 0000 1000 0000 0000 0000 = 0x00208000
Masque32bits	**Entier non signé**			
Macro : *GPIO_Read(**GPIO,Broche**)* ***!! non testée*** *(pour info :#define GPIO_Read(GPIO,Broche) (GPIO->IDR)&(0x01<<Broche))*				
GPIO_TypeDef *Port	**GPIOA, GPIOB, GPIOC**	void		La macro remplace de manière lisible une seule instruction C qui opère un masque sur le registre d'entrée : Port_IO_Lire (GPIO,Broche) est un entier 16 bits qui vaut 0x0000 si la broche est à 0, 0x01<<Broche sinon. Exemple : Test = GPIO_Read (GPIOB,2) ; Test vaudra 0x0004 si la ligne 2 est à '1', 0 sinon.
Char Broche	**0 à 15**			



Timer_1234.c

Gère le périphérique Timer
Voir *Timer_1234.h* pour plus d'infos.

Rappel des ressources (STM32F103RB) :

3 Timer "general Purpose", TIM2, TIM3 et TIM4 + TIM1
Chacun d'entre eux dispose de 4 voies de sorties numérotées de 1 à 4
Mapping des IO:

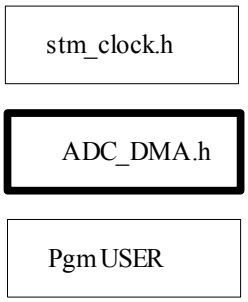
TIM1_CH1 - PA08	TM2_CH1_ETR - PA0	TM3_CH1 - PA6	TIM4_CH1 - PB6
TIM1_CH1 - PA09	TM2_CH2 - PA1	TM3_CH2 - PA7	TIM4_CH2 - PB7
TIM1_CH1 - PA10	TM2_CH3 - PA2	TM3_CH3 - PB0	TIM4_CH3 - PB8
TIM1_CH4 – PA11	TM2_CH4 - PA3	TM3_CH4 - PB1	TIM4_CH4 - PB9

Param entrée	**Valeur**	**Param sortie**	**Valeur**	**Description**
void** Timer_1234_Init_ff**(TIM_TypeDef ***Timer**, u32** Duree_ticks **)				

TIM_TypeDef *Timer	**TIM1,TIM2, TIM3 ou TIM4**	void		Le prescaler est calculé le plus petit possible pour une précision la plus fine possible.
				Le Timer est lancé en mode down. Arrivé à 0, il reprend sa valeur de départ (calculée ds la fonction).
				
				Exemple à 72MHz :
				Init_Clock_System();
				// lance l'initialisation du système d'horloge de la puce (PLL)
				Timer_1234_Init(TIM2, 720);
				// Lance le timer avec une période de 10.0 uS. La fréquence correspondante est donc 100 kHz.

u32 Duree_ticks	Duree demandée pour la période du Timer, exprimée en périodes de l'horloge CPU. Pas de test de limite.			
***void** Active_IT_Debordement_Timer**(TIM_TypeDef ***Timer**, char** Prio**, void** (*IT_function) (void))*				
TIM_TypeDef *Timer	**TIM1,TIM2, TIM3 ou TIM4**	void		La fonction initialise le périphérique et le NVIC de manière à générer une interruption à chaque débordement du timer précisé. Le handler est écrit dans la lib (non accessible). Il opére un test sur les flags (sauf pour TIM1, vectorisation multiple) pour identifier la source d'IT (déb, capure...), puis lance la fonction précisée en paramètre (pointeur de fonction). Avant de lancer la fonction, le handler rabaisse le flag d'IT pour ne pas rerentrer immediatement. Aucune gestion de ce type n'est à faire dans la fonction “IT_function”. Exemple : ***Active_IT_Debordement_Timer(TIM2, 1, IT_Timer2)***; // La fonction active une interruption lors du débordement du Timer 2. La fonction appelée est IT_Timer2. La priorité associée es 1
Char Prio	**0 à 15 (0 : IT de prio la plus élevée)**			
*Void (*IT_function) (void)*	**Nom de la fonction d'interruption** (écrite dans le module qui lance *Active_IT_Debordement_Timer)*			

vu16 *PWM_Init_ff(**TIM_TypeDef ***Timer**, char** Voie**, u32** Periode_ticks **)***				
TIM_TypeDef *Timer	**TIM1,TIM2, TIM3 ou TIM4**	vu16	Renvoie la résolution de la PWM (la pleine échelle de durée d'impulsion).	Cette fonction initialise la voie spécifiée du timer spécifié en PWM. La période souhaitée est passée en paramètre. La fonction renvoie un entier qui correspond à la résolution de la PWM pour pouvoir ensuite régler les rapports cycliques (ARR+1) 3 Timer "general Purpose", TIM2, TIM3 et TIM4 + TIM1 Chacun d'entre eux dispose de 4 voies de sorties numérotées de 1 à 4 Mapping des IO, voir .h. **!!C'est au user de configurer la sortie correctement, Altenate ppull !**
Char Voie	**1 à 4**			
u32 Periode_ticks	**Période souhaitée pour la PWM,** exprimée en périodes d'horloge CPU**.** Aucun test de limite n'est fait.			
***Macro :** PWM_Valeur(Timer,Voie)* *(pour info : #define PWM_Valeur(Timer,Voie) Timer->CCR##Voie)*				
Timer	**TIM1,TIM2, TIM3 ou TIM4**			Permet de fixer la durée à l'état haut de la PWM, dont indirectement son rapport cyclique. La grandeur doit être comprise entre 0 et ARR. Ex: *Reso = PWM_Init (TIM3,2,25.0);* *PWM_Valeur(TIM3,2) = Reso /4; // arrondi à gerer*
Char Voie	**1 à 4**			
***Macro :** CNT(Timer)* *(pour info :#define CNT(Timer) Timer->ARR)*				
Timer	**TIM1,TIM2, TIM3 ou TIM4**			Permet un accès direct au compteur du Timer spécifié sans avoir à connaître les registres du STM32
***Macro :** ARR(Timer)* *(pour info :#define ARR (Timer) Timer->CNT)*				
Timer	**TIM1,TIM2, TIM3 ou TIM4**			Permet un accès direct à l'autoreload du Timer spécifié sans avoir à connaître les registres du STM32
char** Timer_Inc_Init**(TIM_TypeDef ***Timer**, char** Resolution**);				
TIM_TypeDef *Timer	**TIM1,TIM2, TIM3 ou TIM4**	char	Renvoie **0** si tout est OK, et **1** s'il y a un problème (mauvaise résolution)	Fonction de configuration du timer en question en mode codeur incrémental Compte les fronts montant et descendant. Selon la configuration désirée, la résolution est de ½ période du channel 1, ou 2, ou encore d' ¼ de période en comptant sur les deux voies. Voir .h pour les broches correspondantes aux ch 1 & 2 du timer en question. **!!C'est au user de configurer des canaux en entrée !!**
Char Resolution	Reso_Demie_Per_Ch1 2 Reso_Demie_Per_Ch2 1 Reso_Quart_Per 3			
***Macro** : Reset_Timer(Timer)* *(pour info :* #define Reset_Timer(Timer) Timer->CNT=0)				
Timer	**TIM1,TIM2, TIM3 ou TIM4**			Permet de remettre le compteur à 0, utile pour le mode codeur incrémental.



ADC_DMA.c

Gère le module ADC 12 bits et la DMA associée
Voir *ADC_DMA.h* pour plus d'infos.

Les fonctions sont organisées de la manière suivante :

Les fonctions de configuration

Fonctions de conf incontournable de l'ADC	***Single / multiple***	***Manuel / periodique***	***Interruption***	***DMA***
Init_TimingADC_ActiveADC	Single_Channel_ADC	Init_Conversion_On_Trig_Timer (NB: non simulable)	Init_IT_ADC_EOC	Init_ADC1_DMA1 (simulable)
	Init_MultiChan_Regular		Init_IT_End_Of_DMA1	

Les fonctions d'utilisation

Fonction de lancement	***Scrutation***	***Lecture donnée***
#define Start_ADC_Conversion(ADC)	Wait_On_EOC_ADC	#define Read_ADC(ADC
Start_DMA1	Wait_On_End_Of_DMA1	
#define Stop_DMA1		

NB : lorsque la DMA est activée, le flag EOC n'est plus disponible (pas de polling ni sans doute d'IT ADC quand la DMA est en fonction).

Détail des fonctions

Param entrée	**Valeur**	**Param sortie**	**Valeur**	**Description**
***u32** Init_TimingADC_ActiveADC_ff(**ADC_TypeDef** * ADC, **u32** Duree_Ech_ticks);*				

ADC_TypeDef *ADC	ADC1 ou ADC2	u32	Durée totale de conversion exprimée en périodes d'horloge CPU.	// Renvoie la durée de conversion totale (Tacq+Tconv)
				// ADC1 ou ADC2
				
				Met l'ADC en service. Le temps passé en paramètre est la durée d'échantillonnage. La fonction renvoie la durée exacte, immédiatement supérieure à celle spécifiée (dépend du choix des prescaler)+Tconv. L'ADC est « on », prêt à fonctionner. Tous les canaux

u32 Duree_Ech_ticks	Durée d'échantillonnage exprimée en périodes d'horloge CPU			

				sont paramétrés avec la même durée d'échantillonnage. Exemple : Duree_Reelle=Init_TimingADC_ActiveADC_ff(ADC1, 72); // ADC1 avec une durée de 1us d'échantillonnage.
***void** Single_Channel_ADC(**ADC_TypeDef** * ADC, **char** Voie_ADC);*				
ADC_TypeDef *ADC	ADC1 ou ADC2	void		Règle une seule voie pour la conversion ADC. Cette voie est spécifiée dans Voie_ADC.
Char Voie_ADC	0 à 15			
***void** Init_MultiChan_Regular(**ADC_TypeDef** * ADC, **char** Nb_Canaux, **char** Seq_Canaux[]);*				

ADC_TypeDef *ADC	ADC1 ou ADC2	vu16	Valeur convertie calée à gauche ou à droite (par défaut)	// Seq_Canaux[] est une table de longueur Nb_Canaux. Elle contient la séquence
				// d'échantillonnage voulue.
				// Exemple Seq_Canaux[4]={1,0,4,4};
				// provoquera les conversions successives des channel 1,0,4,4.

Char Nb_Canaux	1 à 16			
Char Seq_Canaux[ ]	Adresse du tableau (première case) qui contient la **séquence** d'échantillonnage des canaux (l'**ordre** d'échantillonnage).			
***void** Init_Conversion_On_Trig_Timer_ff(**ADC_TypeDef** * ADC , **char** Source, **u32** Periode_ticks);*				

ADC_TypeDef *ADC	ADC1 ou ADC2	void		// l'ADC est déclenché matériellement par un timer
				// Le timer est lancé (en PWM 50% interne au uC)
				// Les sources de déclenchement :
				#define TIM1_CC1 0
				#define TIM1_CC2 1
				#define TIM1_CC3 2
				#define TIM2_CC2 3
				#define TIM4_CC4 5

char Source	0 à 15			
u32 Periode_ticks	Période d'acquisition exprimée en périodes d'horloge CPU			
***void** Init_IT_ADC_EOC(**ADC_TypeDef** * ADC, **char** Prio, void (*IT_function) (void));*				
ADC_TypeDef *ADC	ADC1 ou ADC2	void		Configure l'ADC de manière à rentrer en interruption lors de la fin de conversion. Le nom de la fonction est passé en paramètre ainsi que la priorité d'interruption associée. Exemple : *Init_IT_ADC_EOC(ADC1, 1, IT_ADC);* // Provoque une interruption de priorité 1, dès que l'ADC a terminé sa conversion. La fonction *IT_ADC* est alors lancée.
Char	Prio			
*Void (*IT_function) (void)*	Nom de la fonction d'interruption à lancer en fin de conv ADC			
***void** Init_IT_End_Of_DMA1(**char** Prio, void (*IT_function) (void));*				

Char	Prio	void		Configure la DMA channel 1 de manière à rentrer en interruption lors de la fin de transfert DMA
				Le nom de la fonction est passé en paramètre ainsi que la priorité d'interruption associée.
				
				Exemple :
				//Init_IT_End_Of_DMA1(1, IT_DMA);
				// Provoque une interruption de priorité 1, dès que la DMA est terminée.
				//La fonction IT_DMA est alors lancée.

*Void (*IT_function) (void)*	*Nom de la fonction d'interruption à lancer en fin de DMA*			
***void** Init_ADC1_DMA1(**char** Circ, vu16 *Ptr_Table_DMA);*				

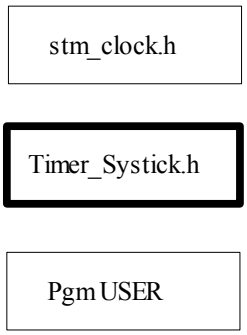
Char Circ	0 ou 1	void		// Prépare l'ADC pour rentrer en DMA. La canal est DMA1. Il est associé à ADC1.
				// La DMA transferre des mots 16 bits
				//Circ = 1 reconfigure la DMA. Une nouvelle bufferisation reprend.

*vu16 *Ptr_Table_DMA* char Source char Voie_ADC				
***void** Start_DMA1(**u16** NbEchDMA);*				
u16 NbEchDMA	C'est le nombre d'échantillons à			*// Lance une DMA sur le nombre de points spécifié.*

	enregistrer dans la table.			*La zone de RAM écrite est indiquée lors de l'appel de la fonction* *Init_ADC1_DMA1*
void Wait_On_EOC_ADC (ADC_TypeDef * ADC);				
ADC_TypeDef * ADC)	ADC1 ou ADC2			// Fonction bloquante qui scrute l'indicateur EOC de l'ADC
void Wait_On_End_Of_DMA1(void);				
				// Fonction bloquante qui scrute l'indicateur TCIF1 de la DMA

Timer_SysTick.c

Permet de régler le timer *Systick* (modulo et interruption)



Param entrée	Valeur	Param sortie	Valeur	Description
***void** Systick_Prio_IT(**char** Prio,void (*Systick_function) (void));*				
char Prio	0 à 15, 0 est la plus haute priorité.	void		Fixer la priorité de l'IT coeur Systick, plus elle est basse, plus la prio est importante Le second paramètre est le nom de la fonction à lancer lors de l'interruption
*void (*IT_function) (void)*	Nom de la fonction d'interruption (écrite dans le module qui lance la fonction)			
***void** Systick_Period_ff(**u32** Periode_ticks);*				
u32 Periode_ticks	Période de l'interruption du timer, exprimée en périodes de l'horloge CPU	void		Fixer la periode du Systick La fonction fixe le prescaler Systick (ds le clock tree) à 1/8 L'init des horloges doit donc être lancée AVANT cette fonction. NB: pas de prescaler autre que 1/8. Exemple de code : configuration en IT, horloge 72 MHz, période de 0,5s Systick_Period_ff(500000*72); Systick_Prio_IT(2,cligno); SysTick_On; SysTick_Enable_IT;
***Définition :** SysTick_On* *(pour info :#define SysTick_On ((SysTick->CTRL)=(SysTick->CTRL)|1<<0))*				
***Définition :** SysTick_Off* *(pour info :#define SysTick_Off ((SysTick->CTRL)=(SysTick->CTRL)& ~(1<<0))*				
***Définition :** SysTick_Enable_IT* *(pour info :#define SysTick_Enable_IT ((SysTick->CTRL)=(SysTick->CTRL)|1<<1))*				
***Définition :** SysTick_Disable_IT* *(pour info :#define SysTick_Disable_IT ((SysTick->CTRL)=(SysTick->CTRL)& ~(1<<1))*				