Compléments de programmation en C pour le processeur STM32

Auteurs et contributeurs : Francis Dupin

V.1.6 mai 2019

Table des matières

1 Les fichiers essentiels

Dans chaque projet *Eclipse*, plusieurs fichiers sont à consulter pour connaître le nom des macros d'accès aux registres, le nom des interruptions, etc.

Les macros d'accès aux registres et le nom des vecteurs d'interruptions
Ils sont définis dans :
*<projet>/Drivers/CMSIS/Device/ST/STM32L0xx/Include/**stm32l053xx.h***

Pour chaque grande fonction, une structure a été définie, qui donne accès à ses registres. Exemple :

Pour les registres GPIO (gérant les ports d'e/s), un type *GPIO_TypeDef* est défini ligne 291, et des variables d'accès sont définies lignes 742 :

```
// Ligne 291
typedef struct
{
        __IO uint32_t MODER;      /*!< GPIO port mode register,
                                    Address offset: 0x00 */
        __IO uint32_t OTYPER;     /*!< GPIO port output type register,
                                   Address offset: 0x04 */
        __IO uint32_t OSPEEDR;    /*!< GPIO port output speed register,
                                  Address offset: 0x08 */
        __IO uint32_t PUPDR;      /*!< GPIO port pull-up/pull-down register,
                             Address offset: 0x0C */
        __IO uint32_t IDR;        /*!< GPIO port input data register,
                                    Address offset: 0x10 */
        __IO uint32_t ODR;        /*!< GPIO port output data register,
                                   Address offset: 0x14 */
        __IO uint32_t BSRR;       /*!< GPIO port bit set/reset registerBSRR,
                             Address offset: 0x18 */
        __IO uint32_t LCKR;       /*!< GPIO port configuration lock register,
                           Address offset: 0x1C */
        __IO uint32_t AFR[2];     /*!< GPIO alternate function register,
                                 Address offset: 0x20-0x24 */
        __IO uint32_t BRR;        /*!< GPIO bit reset register,
                                    Address offset: 0x28 */
}GPIO_TypeDef;

// Ligne 678
#define GPIOA_BASE          (IOPPERIPH_BASE + 0x00000000UL)

// Ligne 742
#define GPIOA               ((GPIO_TypeDef *) GPIOA_BASE)
```

Exemple d'utilisation :

```
/* Exemple d'utilisation */
GPIOA->ODR = 0x0 /* Mise à0 du registre ODR du port GPIOA */
```

Il est donc nécessaire de fouiller ce fichier pour savoir comment accéder aux registres.
La liste des vecteurs d'interruptions se trouve entre les lignes 79 et 107 :

```
/****** STM32L-0 specific Interrupt Numbers
    *****************************************************/
WWDG_IRQn                 = 0,      /*!< Window WatchDog Interrupt
                                      */
PVD_IRQn                  = 1,      /*!< PVD through EXTI Line detect Interrupt
                                    */
RTC_IRQn                  = 2,      /*!< RTC through EXTI Line Interrupt
                                   */
FLASH_IRQn                = 3,      /*!< FLASH Interrupt
                                       */
RCC_CRS_IRQn              = 4,      /*!< RCC and CRS Interrupts
                                     */
EXTI0_1_IRQn              = 5,      /*!< EXTI Line 0 and 1 Interrupts
                                    */
EXTI2_3_IRQn              = 6,      /*!< EXTI Line 2 and 3 Interrupts
                                    */
EXTI4_15_IRQn             = 7,      /*!< EXTI Line 4 to 15 Interrupts
                                     */
TSC_IRQn                  = 8,      /*!< TSC Interrupt
                                        */
DMA1_Channel1_IRQn        = 9,      /*!< DMA1 Channel 1 Interrupt
                                    */
DMA1_Channel2_3_IRQn      = 10,     /*!< DMA1 Channel 2 and Channel 3 Interrupts
                                  */
DMA1_Channel4_5_6_7_IRQn  = 11,     /*!< DMA1 Channel 4, Channel 5, Channel 6 and
    Channel 7 Interrupts */
ADC1_COMP_IRQn            = 12,     /*!< ADC1, COMP1 and COMP2 Interrupts
                                    */
```

```
LPTIM1_IRQn               = 13,     /*!< LPTIM1 Interrupt
                                       */
TIM2_IRQn                 = 15,     /*!< TIM2 Interrupt
                                        */
TIM6_DAC_IRQn             = 17,     /*!< TIM6 and DAC Interrupts
                                    */
TIM21_IRQn                = 20,     /*!< TIM21 Interrupt
                                        */
TIM22_IRQn                = 22,     /*!< TIM22 Interrupt
                                        */
I2C1_IRQn                 = 23,     /*!< I2C1 Interrupt
                                         */
I2C2_IRQn                 = 24,     /*!< I2C2 Interrupt
                                         */
SPI1_IRQn                 = 25,     /*!< SPI1 Interrupt
                                         */
SPI2_IRQn                 = 26,     /*!< SPI2 Interrupt
                                         */
USART1_IRQn               = 27,     /*!< USART1 Interrupt
                                       */
USART2_IRQn               = 28,     /*!< USART2 Interrupt
                                       */
RNG_LPUART1_IRQn          = 29,     /*!< RNG and LPUART1 Interrupts
                                     */
LCD_IRQn                  = 30,     /*!< LCD Interrupt
                                         */
USB_IRQn                  = 31,     /*!< USB global Interrupt
                                      */
} IRQn_Type;
```

Les fonctions d'interruptions

Le nom des fonctions qui seront appelées pour servir une interruption sont dans : *<projet>/startup/**startup_stm32l053xx.s*** :

Pour les répérer, leur nom se termine par *Handler*...

```
g_pfnVectors:
.word _estack
.word Reset_Handler
.word NMI_Handler
.word HardFault_Handler
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word 0
.word SVC_Handler
.word 0
.word 0
.word PendSV_Handler
.word SysTick_Handler
.word    WWDG_IRQHandler                /* Window WatchDog           */
.word    PVD_IRQHandler                 /* PVD through EXTI Line detection */
.word    RTC_IRQHandler                 /* RTC through the EXTI line     */
.word    FLASH_IRQHandler               /* FLASH                     */
.word    RCC_CRS_IRQHandler             /* RCC and CRS               */
.word    EXTI0_1_IRQHandler             /* EXTI Line 0 and 1         */
.word    EXTI2_3_IRQHandler             /* EXTI Line 2 and 3         */
.word    EXTI4_15_IRQHandler            /* EXTI Line 4 to 15         */
.word    TSC_IRQHandler                  /* TSC                        */
.word    DMA1_Channel1_IRQHandler       /* DMA1 Channel 1            */
.word    DMA1_Channel2_3_IRQHandler     /* DMA1 Channel 2 and Channel 3 */
.word    DMA1_Channel4_5_6_7_IRQHandler /* DMA1 Channel 4, Channel 5, Channel 6 and
    Channel 7*/
.word    ADC1_COMP_IRQHandler           /* ADC1, COMP1 and COMP2 */
.word    LPTIM1_IRQHandler              /* LPTIM1                    */
.word    0                              /* Reserved                  */
```

```
.word    TIM2_IRQHandler              /* TIM2                    */
.word    0                            /* Reserved                */
.word    TIM6_DAC_IRQHandler          /* TIM6 and DAC            */
.word    0                                          /* Reserved
                    */
.word    0                                                  /* Reserved
                    */
.word    TIM21_IRQHandler             /* TIM21                   */
.word    0                            /* Reserved                */
.word    TIM22_IRQHandler             /* TIM22                   */
.word    I2C1_IRQHandler              /* I2C1                    */
.word    I2C2_IRQHandler              /* I2C2                    */
.word    SPI1_IRQHandler              /* SPI1                    */
.word    SPI2_IRQHandler              /* SPI2                    */
.word    USART1_IRQHandler            /* USART1                  */
.word    USART2_IRQHandler            /* USART2                  */
.word    RNG_LPUART1_IRQHandler       /* RNG and LPUART1         */
.word    LCD_IRQHandler               /* LCD                     */
.word    USB_IRQHandler               /* USB                     */
```

2 Les ports d'entrée/sortie

2.1 Quelques macros bien pratiques

Et qu'on utilisera dans ce qui suit ...

```
// Pour registre RCC_IOPENR (activation horloge pour les ports GPIO
#define GPIO_ACTIVE_HORLOGE 0x1UL
#define GPIO_INACTIVE_HORLOGE 0x0UL
// Pour registre GPIOx_MODER
#define GPIO_SORTIE   0x1UL
#define GPIO_ENTREE   0x0UL
#define GPIO_FCT_ALTERN 0x2UL
#define GPIO_ANALOG   0x3UL
// Pour registre GPIOx_OTYPER
#define GPIO_SORTIE_PUSHPULL  0x0UL
#define GPIO_SORTIE_DRAIN_OUVERT 0x1UL
// Pour registre GPIOx_PUPDR
#define GPIO_INACTIVE_PULL_UP_DOWN 0x0UL
#define GPIO_ACTIVE_PULL_UP     0x1UL
#define GPIO_ACTIVE_PULL_DOWN   0x2UL
// Pour Registre GPIOx_OSPEEDR
// Détermine la raideur des fronts de transition entre états haut et bas
#define GPIO_VITESSE_TRES_FAIBLE 0x0UL
#define GPIO_VITESSE_FAIBLE    0x1UL
#define GPIO_VITESSE_MOY       0x2UL
#define GPIO_VITESSE_ELEVEE    0x3UL
// Pour registre GPIOx_AFSR
#define GPIO_FCT_ALTERN_0      0
#define GPIO_FCT_ALTERN_1      1
#define GPIO_FCT_ALTERN_2      2
#define GPIO_FCT_ALTERN_3      3
#define GPIO_FCT_ALTERN_4      4
#define GPIO_FCT_ALTERN_5      5
#define GPIO_FCT_ALTERN_6      6
#define GPIO_FCT_ALTERN_7      7
```

2.2 Activer l'horloge

Toutes les grandes fonctions du processeur ont leur horloge inactivée par défaut (c'est à dire que ces fonctions sont inactives), ceci pour diminuer la consommation électrique. Il en va de même pour les ports d'e/s. Chaque port a son horloge, qu'il faut activer.

```
/* Activer l'horloge du port A */
RCC->IOPENR = (RCC->IOPENR & ~(RCC_IOPENR_IOPAEN)) | GPIO_ACTIVE_HORLOGE <<
    RCC_IOPENR_IOPAEN_Pos;
/* Activer l'horloge du port C */
RCC->IOPENR = (RCC->IOPENR & ~(RCC_IOPENR_IOPCEN)) | GPIO_ACTIVE_HORLOGE <<
    RCC_IOPENR_IOPCEN_Pos;
```

2.3 Configurer des broches en entrées numériques

Rappel : Les entrées non connectées ont une très haute impédance, ce qui veut dire qu'une petite perturbation de l'environnement peut changer leur état et -plus grave- les faire osciller. Pour cette raison, lorsqu'elle ne sont pas connectées, on diminue leur impédance

d'entrée par une petite résistance interne de rappel connectée soit à VCC (résistance de pull-up), soit à la masse (résistance de pull-down).
Il faut y penser lorsqu'on relie l'entrée à un simple interrupteur. Sur le STM32, les possibilités sont :

- Résistance de pull-up
- Résistance de pull-down
- Résistance de pull-up et Résistance de pull-down (rarement utile)
- Pas de résistance

Registre *MODER* : Gestion du mode entrée/sortie
Registre *PUPDR* : Gestion des résistances de rappel

```
/* Configurer la broche 1 du port C en entrée */
GPIOC->MODER = (GPIOC->MODER & ~(GPIO_MODER_MODE1)) | GPIO_ENTREE <<
    GPIO_MODER_MODE1_Pos;
/* Mode pull-up */
GPIOC->PUPDR = (GPIOC->PUPDR & ~(GPIO_PUPDR_PUPD1)) | GPIO_ACTIVE_PULL_UP <<
    GPIO_PUPDR_PUPD1_Pos;
```

2.4 Configurer des broches en sorties numériques

Sauf pour des branchements particuliers, les sorties sont à configurer en *push-pull* au lieu de *drain ouvert*. C'est le défaut à l'initialisation, il n'y a donc pas à s'en préoccuper.
Registre *MODER* : Gestion du mode entrée/sortie

```
/* Configurer la broche 12 du port A en sortie */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODE12)) | GPIO_SORTIE <<
    GPIO_MODER_MODE12_Pos;
```

2.5 Configurer des broches en entrée analogique

Registre *MODER* : Gestion du mode entrée/sortie

```
/* Configure la broche 7 du port A en entrée analogique */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODE7)) | GPIO_ANALOG <<
    GPIO_MODER_MODE7_Pos;
```

2.6 Les fonctions alternatives et les fonctions additionnelles

On peut associer à chaque broche d'autres fonctions que les fonctions d'entrées/sorties classiques.
Les fonctions alternatives doivent être sélectionnées par les registres *AFRL et AFRH*, alors que les fonctions additionnelles ne nécessitent pas de configuration supplémentaire. La liste des fonctions alternatives et additionnelles par broche est dans *stm32l053r8.pdf*.
Registres *AFRL, AFRH* : Pour le choix de la fonction alternative.
Registre *MODER* : Pour sélectionnner le mode fonction alternative.

```
    /* Sélection de la fonction alternative 2 pour la broche 0 du port A */
    /*AFR[0] pour AFRL, AFR[1] pour AFRH*/
GPIOA->AFR[0] |= GPIO_FCT_ALTERN_2 << GPIO_AFRL_AFSEL0_Pos;

/* Choisir le mode fonctions alternatives */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODE0)) | GPIO_FCT_ALTERN <<
    GPIO_MODER_MODE0_Pos;
```

2.7 Lire une broche d'entrée numérique

Registre *IDR*

```
uint16_t pA;
char br6:
/* Lecture de tout le port A. */
/* !! Seuls les bits des broches qui sont en entrée ont une signification */
pA = GPIOA->IDR;

/* La valeur de la broche 6 */
br6 = (pA >> 6) & 1;
```

2.8 Ecrire sur une broche de sortie numérique

Le registre BSRR de 32 bits permet, en écrivant un 1 pour la broche correspondante, de :

— Dans sa partie haute, mettre la broche à 0
— Dans sa partie basse, mettre la broche à 1

Ce registre est bien pratique car il permet de modifier la sortie d'une broche isolément sans avoir à écrire un masque. On y accède en C par les pseudo registres BRR (mise à 0) et BSRR (mise à 1)

```
/* Mise à 0 de la broche 6 du port A */
    GPIOA->BRR = 1 << 6

    /* Mise à 1 de la broche 6 du port A */
    GPIOA->BSRR = 1 << 6
```

3 Les interruptions

Beaucoup d'événements peuvent déclencher une interruption : Fin de comptage d'un timer, fin de conversion analogique-numérique, changement d'état d'une broche en entrée d'un port GPIO, ...

Rappel du principe d'interruption : Un événement déclenche l'arrêt du déroulement du programme, qui est dérouté vers une fonction particulière. À la fin de celle-ci, le programme reprend son cours à partir de l'endroit où il a été interrompu.

3.1 Particulatités du STM32

Le contrôleur d'interruptions NVIC
Les interruptions sont regroupées et dirigées vers 35 vecteurs d'interruptions. En pratique, en C, cela veut dire que plusieurs événements différents déroutent le programme vers une même fonction d'interruption. Il faudra donc à l'intérieur de celle-ci mener quelques tests pour savoir quel événement s'est produit.

Pour la table d'interruptions, voir le fichier STM32L053xx_Reference.pdf, table 50 p.283.

Exemple : Tous les événements sur les broches 4 à 15 des ports A, B, ... déclenchent la fonction d'interruption EXTI4_15_IRQHandler.

Pour le nom des fonctions d'interruptions, voir *§ 1*.

C'est le contrôleur d'interruption *NVIC* (Nested vectored interrupt controller) - qui fait partie du coeur ARM - qui gère tout le processus.

Plusieurs fonctions permettent d'agir sur le comportement du contôleur, sur les interruptions pour les activer, choisir leur priorité.
Exemple : NVIC_EnableIRQ(vecteur_IT) est nécessaire pour activer une interruption.
La liste des fonctions est décrite dans :
<projet>/Drivers/CMSIS/Include/core_sc000.h vers la ligne 748 :

```
void     NVIC_EnableIRQ(IRQn_Type IRQn)
void     NVIC_DisableIRQ(IRQn_Type IRQn)
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)
void     NVIC_SetPendingIRQ(IRQn_Type IRQn)
void     NVIC_ClearPendingIRQ(IRQn_Type IRQn)
void     NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
uint32_t NVIC_GetPriority(IRQn_Type IRQn)
void     NVIC_SystemReset(void)
```

Le contrôleur d'interruptions EXTI
Certaines interruptions - celles déclenchées par les modules externes au coeur ARM (par exemple les GPIO) - sont gérées en amont par le contrôleur *EXTI* (Extended interrupt and event contoller) qui les transmet ensuite au contrôleur NVIC.
Le contrôleur EXTI organise les interruptions en 30 lignes *EXTI lines*.

Les plus importantes pour nous sont les lignes 0 à 15, qui concernent les GPIO. (Voir ci-dessous).

3.2 Interruption sur le changement d'état d'une entrée numérique

Le contrôleur d'interruptions *EXTI* regroupe les interruptions GPIO dans les lignes 0 à 15 de la manière suivante :

— Toutes les broches *n* des ports A, B, ..., H sont regroupées dans la ligne *n*,
— Une ligne ne peut servir qu'une seule source d'interruption.

Exemple :
Vouloir récupérer par interruptions des événements sur PA8 et PC8 (broche 8 du port A

et broche 8 du port C) n'est pas possible. Il faut choisir des numéros de broche différents, comme PA8 et PC9.

Exemple de déclenchement d'une interruption sur le front montant de PA7 :

Registres à configurer :

— *MODER* et *PUPDR* du module GPIO : Configurer la broche en sortie
— *IMR* et *EMR* du module EXTI : Démasquer l'interruption pour la ligne concernée
— *RTSR* et *FTSR* du module EXTI : Sélectionner le type d'événement déclencheur (front montant et/ou descendant)
— *EXTICRn* du module SYSCFG : Choisir le port et le numéro de bit

```
/* Dans la fonction main() */
/* ----------------------- */

/* Configuration de PA7 en entrée, sans résistance de rappel */
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODE7)) | GPIO_ENTREE <<
    GPIO_MODER_MODE7_Pos;
GPIOA->PUPDR = ((GPIOA->PUPDR) & ~(GPIO_PUPDR_PUPD7)) |
    GPIO_INACTIVE_PULL_UP_DOWN << GPIO_PUPDR_PUPD7_Pos;

/* Démasquer l'IT pour la ligne 7 */
EXTI->IMR = EXTI->IMR | EXTI_IMR_IM7; // écriture 1 << 7 (car ligne 7) dans registre
     IMR pour démasquer l'IT de cette ligne
EXTI->EMR = EXTI->EMR | EXTI_EMR_EM7; // 1 << 7 dans registre EMR : pour dé
     masquer l'événement de cette ligne

/* Configuration IT sur front montant (1 dans RTSR) mais pas sur front descendant (0
     dans FTSR), ceci toujours pour la ligne 7 */
EXTI->RTSR = EXTI->RTSR | EXTI_RTSR_RT7;
EXTI->FTSR = EXTI->FTSR & ~EXTI_FTSR_FT7;

/* Démasquer le bit 7 du port A. Pour le bit 7, configurer le registre EXTICR2 */
/* Les registres EXTICRn permettent de choisir, pour1 bit, quel port doit déclencher
     l'interruption */
/* Pour les bits 0 à 3,   configurer EXTICR1 (SYSCFG->EXTICR[0]) */
/* Pour les bits 4 à 7,   configurer EXTICR2 (SYSCFG->EXTICR[1]) */
/* Pour les bits 8 à 11,  configurer EXTICR3 (SYSCFG->EXTICR[2]) */
/* Pour les bits 12 à 15, configurer EXTICR4 (SYSCFG->EXTICR[3]) */
SYSCFG->EXTICR[1] = (SYSCFG->EXTICR[1] & ~SYSCFG_EXTICR2_EXTI7_Msk) |
    SYSCFG_EXTICR2_EXTI7_PA;

/* Démasquer l'interruption au niveau du coeur ARM */
/* Pour l'argument : La liste des vecteurs d'interruptions se trouve */
/* dans le fichier stm32l053xx.h vers la ligne 79 */
NVIC_EnableIRQ(EXTI4_15_IRQn); // Concerne les lignes 4 à15
```

```
/* La fonction d'interruption, en dehors de main() */
/* ---------------------------------------------- */
/* Rappel, cette fonction est commune aux lignes 4 à 15, comme son nom l'indique */
/* La liste des noms de fonction se trouve dans le fichier startup_stm32l053xx.s */
void EXTI4_15_IRQHandler(void) {
        // Test si l'IT a été déclenchée par PA7 (soit donc la ligne EXTI7), sinon
            retour.
        if ((EXTI->PR & EXTI_PR_PIF7) == 0 )
        return;
        // Arrivé ici, c'est bien PA7 qui a déclenché une IT
        // RAZ du bit d'IT par écriture d'un 1
        EXTI->PR = EXTI_PR_PIF7;
        /* (...) */
}
```

4 Conversion analogique numérique

Une broche GPIO est connectée en entrée à une source de tension analogique (exemple : un potentiomètre). Il s'agit de convertir cette tension en une valeur numérique.
Il est possible de déclencher une interruption à chaque fois que le processus de conversion se termine.

16 broches (pas n'importe lesquelles) sont connectables à un canal de conversion :
(Voir stm32l053r8.pdf table 15 p.41)
Les broches considérées devront être programmées comme des *entrées analogiques*.

Broche	**Canal x** (ADC_INx)
PA0	0
PA1	1
PA2	2
PA3	3
PA4	4
PA5	5
PA6	6
PA7	7
PB0	8
PB1	9
PC0	10
PC1	11
PC2	12
PC3	13
PC4	14
PC5	15

Exemple : Conversion de l'entrée PA7.

```
volatile int tension = 0      // Variable globale

// La fonction d'interruption
// La fonction de traitement des IT pour le convertisseur analogique->numérique (cf
    startup_stm32l053xx.s)
void ADC1_COMP_IRQHandler(void)
{
        //printf("IT\n");
        // Lecture de la tension convertie en numérique (sur 8 bits calés à droite)
            entre 0 et 3,3V
        tension = ADC1->DR;
        // Acquittement de l'interruption. Sinon on n'en sort pas
        ADC1->ISR = ADC_ISR_EOC;
}
```

```
// D'après stm32l053r8.pdf table 15 p.41, PA7 a pour fonction additionnelle ADC_IN7.
    Pour l'utiliser,
// Il suffit de placer PA7 en entrée analogique et de configurer les registres du CAN
// Configuration PA7 en entrée analogique
// ======================================
GPIOA->MODER = (GPIOA->MODER & ~(GPIO_MODER_MODE7)) | GPIO_ANALOG <<
    GPIO_MODER_MODE7_Pos;
// Activation horloge du port A
RCC->IOPENR = (RCC->IOPENR & ~(RCC_IOPENR_IOPAEN)) | GPIO_ACTIVE_HORLOGE <<
    RCC_IOPENR_IOPAEN_Pos;
```

```
// Configuration convertisseur analogique->numérique
//=====================================================
  // Configuration de l'horloge
// -------------------------
// Activation de l'horloge du CAN
RCC->APB2ENR |= RCC_APB2ENR_ADCEN;
// Registre ADC_CCR
// PRESC = 1011 Horloge divisée par 256
ADC1_COMMON->CCR = 0xB << ADC_CCR_PRESC_Pos;

// Registre ADC_CFGR2
// CKMODE <- 2 (synchronous clock mode / 4)
ADC1->CFGR2 = 2 << ADC_CFGR2_CKMODE_Pos;

// Calibration (optionnel)
// ----------------------
// Registre ADC_CR
//ADC1->CR = ADC1->CR | ADC_CR_ADDIS; // Désélectionne le Convertisseur ! A ne
    pas faire
if((ADC1->CR & ADC_CR_ADEN) != 0) {
    ADC1->CR &= ~(uint32_t)ADC_CR_ADEN;
}
ADC1->CR |= (uint32_t)ADC_CR_ADCAL; // Calibration
while ((ADC1->ISR & (uint32_t)ADC_ISR_EOCAL) == 0){} // Attente EOCAL = 1 (fin
    calibration)
ADC1->ISR |= ADC_ISR_EOCAL; // RAZ de EOCAL (fin de calibration)

// Divers
// ------
// Registre ADC_CFGR1
// CONT <- 1 (Conversions en continu)
// RES <- 2 (Résolution 8 bits) : ADC_CFGR1_RES_1
ADC1->CFGR1 = ADC_CFGR1_CONT | ADC_CFGR1_RES_1;

// Registre ADC_SMPR
// SMP <- 111 (Durée de charge du condensateur d'entrée : 239,5 cycles d'horloge à 16
    MHZ). SMP <- 0 irait aussi, on laisse cette valeur par défaut

// Registre ADC_CHSELR (Sélection des canaux de conversion àactiver)
// CHSEL7 <- 1 Conversion uniquement sur PA7
ADC1->CHSELR = ADC_CHSELR_CHSEL7;

// Autoriser les interruptions uniquement àchaque fin de conversion d'un canal
ADC1->IER = ADC_IER_EOCIE;
```

```
// Démasquer les interruptions du CAN
// Les valeurs sont dans stm32l053xx.h lignes 90 et suivantes
NVIC_EnableIRQ(ADC1_COMP_IRQn);

// Active le CAN
ADC1->CR |= ADC_CR_ADEN;

// Démarre les conversions
ADC1->CR = ADC_CR_ADSTART;
// Fin configuration du convertisseur Analogique -> Numérique
```

5 Programmer des attentes

Attente en ms

En utilisant la bibliothèque LL :

```
LL_mDelay(200); // Attente bloquante 200 ms
```

Attente en μs

Écrire dans le code la fonction suivante, qui utilise le timer 6 :

```
/*
* Fonction qui se suffit à elle même pour une attente en us à partir de 5 us environ
    jusqu'à 32000 us
* Régler la variable mult suivant la fréquence de l'horloge.
*     attente_us : une valeur entre 5 et 32000
*     bloquer = 0 : attente non bloquante. bloquante sinon. Tester alors en fin de
    comptage (TIM6->CR1 & TIM_CR1_CEN) == 0
*     lancer : 1 pour démarrer une attente, 0 sinon.
*/
void attente_timer6_us(uint16_t attente_us, uint8_t bloquer, uint8_t lancer)
{
        uint8_t mult = 3; // 3 pour une horloge de 8 MHz, 7 pour 16 Mhz
        uint16_t offset = 10; // Offset pour compenser les instructions de confi
            guration. 10 pour horloge de 8 Mhz
        uint16_t cnt_us = (attente_us << 1) -offset;
        if (cnt_us <= 0)
        cnt_us = 1;
        if (lancer != 1) {
                return;
        }
        // Activation horloge du timer 6
        RCC->APB1ENR = RCC->APB1ENR | RCC_APB1ENR_TIM6EN;
        TIM6->EGR = TIM_EGR_UG; // Reset de tous les registres du timer
        //TIM6->DIER = 0; // Ni DMA ni IT
        TIM6->SR = 0; // RAZ du bit d'overflow'
        TIM6->PSC = mult; // Division de l'horloge (à priori 8MHz) par valeur + 1,
            sur 16 bits
        TIM6->ARR = cnt_us;
        TIM6->CR1 = TIM_CR1_OPM | TIM_CR1_CEN;
        if (bloquer == 1) {
                while ((TIM6->CR1 & TIM_CR1_CEN) == 1){}
        }
}
```

Utilisation :

```
attente_timer6_us(100, 1, 1); // Attente bloquante 100 us

/* Attente non bloquante */
attente_timer6_us(100, 0, 1);
/* (...) */
if (TIM6->CR1 & TIM_CR1_CEN) == 0) {
        /* Le timer a fini de compter */
}
```

6 Aide au déboguage

6.1 Écrire dans une console du PC

Via l'USART2, il est possible d'écrire dans une console du PC. Les caractères remontent par la connexion USB.
Pour cela, il faut définir dans le code la fonction _write :

```
// Pour que les printf remontent au PC par USB
#include <stdio.h>
int _write(int file,char * ptr, int len)
{
        for (int i=0; i<len;i++)
        {
                while(LL_USART_IsActiveFlag_TXE(USART2)==0);
                LL_USART_TransmitData8(USART2,*ptr);
                ptr++;
        }
        return len;
}
```

Utilisation :

```
/* Terminer la chaîne par \n pour un affichage immédiat (càd. non bufferisé) */
printf("Coucou\n");
```

Configuration sous Windows
Pour trouver le port, regarder dans Panneau de configuration / Système / Gestionnaire des périphériques
Utiliser putty, le port doit être configuré en 115200 bauds 8N1

Configuration sous Linux
il faut rechercher quel est le port série à lire.

```
dmsg | tty
    cdc_acm 3-1:1.2: ttyACM3: USB ACM device
```

Installer minicom

```
sudo apt-get install minicom
```

Configurer le port avec minicom en root

```
sudo minicom
# puis CTR A-o pour configurer le port
+-----------------------------------------------------------------------+
| A -                          Port série : /dev/ttyACM3                |
| B - Emplacement du fichier de verrouillage : /var/lock                |
| C -           Programme d'appel intérieur :                           |
| D -           Programme d'appel extérieur :                           |
| E -                    Débit/Parité/Bits : 115200 8N1                 |
| F -             Contrôle de flux matériel : Oui                       |
| G -             Contrôle de flux logiciel : Non                       |
|                                                                       |
|    Changer quel réglage ?

# CTR A-x pour quitter minicom, puis

cat /dev/ttyACM3 # pour afficher
```

6.2 Accéder aux registres du microcontrôleur

Il faut exécuter le code en mode déboguage.

7 La carte Nucleo L053R8

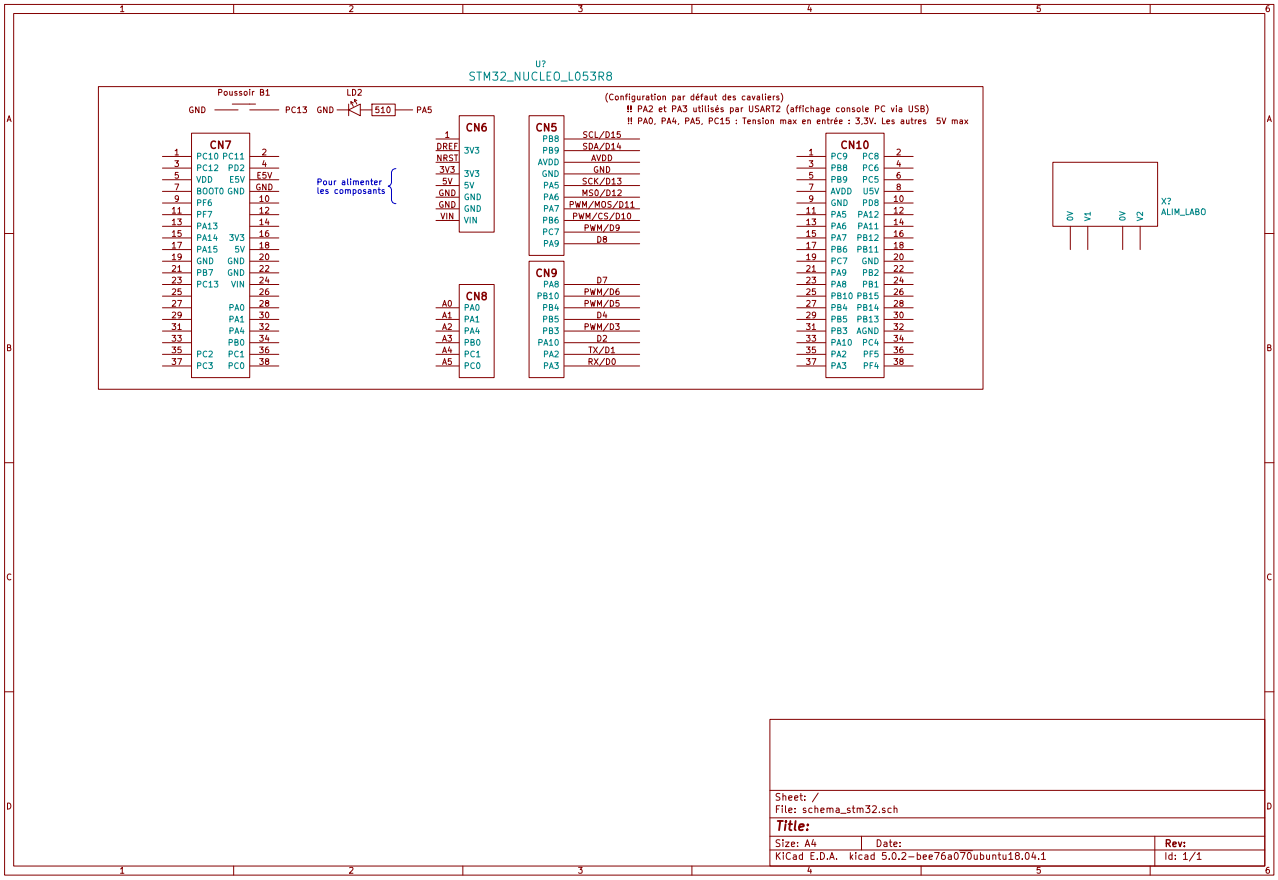

FIGURE 1 – Les connecteurs d'e/s de la carte Nucleo L053R8, dans sa configuration par défaut des cavaliers. En vert dans les connecteurs, P... est le nom des ports d'e/s (GPIO)

8 L'environnement de développement Eclipse

8.1 Dupliquer un projet existant

Un dossier projet intégré à Eclipse ne peut pas être dupliqué à partir du système d'exploitation, mais doit l'être à partir d'Eclipse :
Dans le navigateur de projets, cliquer droit sur le projet à dupliquer. Choisir de Copier puis cliquer droit Coller. Un menu propose alors de choisir un emplacement de copie.

8.2 Importer un projet dans Eclipse

File/Import/General/Existing project into workspace
Select root directory.

Si le projet qu'on veut importer a été obtenu par copie d'un projet déjà intégré à Eclipse, l'importation va échouer avec le message d'erreur : ≪Some projects cannot be imported because they already exist in the workspace≫. (Voir *§ 8.1*).

8.3 Ajouter des fichiers .c ou .h à un projet

Copier les fichiers dans les dossiers (*Inc* pour les .h, *Src* pour les .c).
Mettre à jour le projet Eclipse : Cliquer droit sur le projet/Refresh.
Les fichiers apparaissent alors dans l'arborescence du projet.

8.4 Compiler un projet

Pour la première compilation,
Sélectionner le projet dans le navigateur de gauche.
Clic droit sur le projet dans le navigateur de gauche,
build configuration / build selected

Les fois suivantes, menu horizontal du haut : Project/Build project

8.5 Charger le projet sur la carte

Pour le premier chargement, cliquer droit sur le projet (navigateur de gauche),
Run as / Run configurations...
puis Run
(Voir *figure 2*)

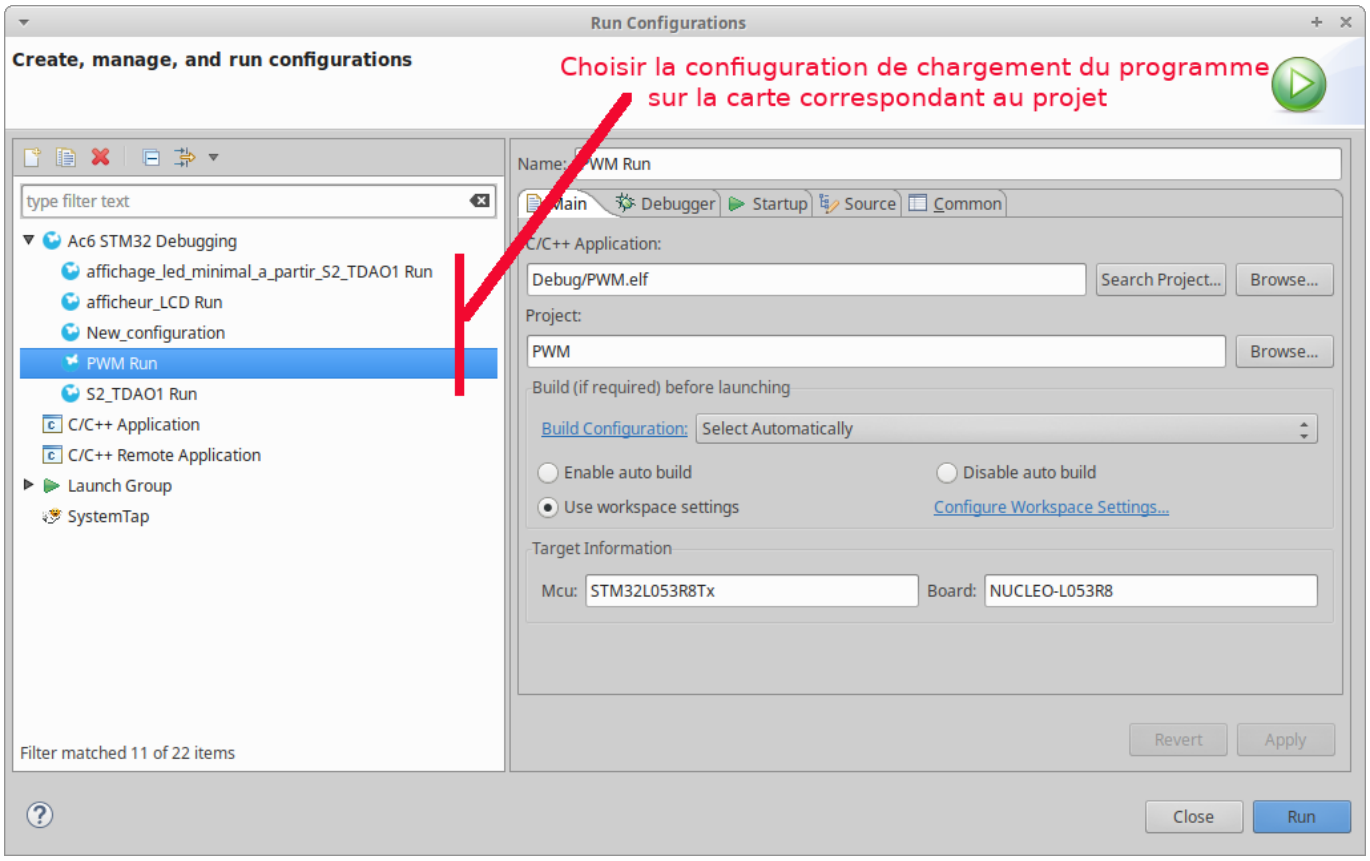

FIGURE 2 – Configuration du chargement du programme sur la carte

Les fois suivantes, choisir l'icone du menu du haut représentant une flèche vers la droite.

Lors du chargement de la carte, la led LD1 doit clignoter vert/rouge,

Message dans l'onglet console :

```
Open On-Chip Debugger 0.10.0-dev-00015-gaaf1808 (2018-11-13-12:51)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
Warn : Could not determine executable path, using configured BINDIR.
srst_only separate srst_nogate srst_open_drain connect_assert_srst
Info : The selected transport took over low-level target control. The results
     might differ compared to plain JTAG/SWD
adapter speed: 240 kHz
adapter_nsrst_delay: 100
Info : clock speed 240 kHz
Info : STLINK v2.1 JTAG v29 API v2 M18 VID 0x0483 PID 0x374B
Info : using stlink api v2
Info : Target voltage: 3.262673
Info : Stlink adapter speed set to 240 kHz
Info : STM32L053R8Tx.cpu: hardware has 4 breakpoints, 2 watchpoints
Info : Stlink adapter speed set to 240 kHz
adapter speed: 240 kHz
target halted due to debug-request, current mode: Thread
xPSR: 0xf1000000 pc: 0x08000ae0 msp: 0x20002000
STM32L0: Enabling HSI16
Info : Stlink adapter speed set to 4000 kHz
adapter speed: 4000 kHz
** Programming Started **
auto erase enabled
Info : Device: STM32L0xx (Cat. 3)
Info : STM32L flash size is 64kb, base address is 0x8000000
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000000e msp: 0x20002000
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000000e msp: 0x20002000
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000000e msp: 0x20002000
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000000e msp: 0x20002000
wrote 4096 bytes from file Debug/S2_TDA01.elf in 0.437296s (9.147 KiB/s)
** Programming Finished **
** Verify Started **
target halted due to breakpoint, current mode: Thread
xPSR: 0x61000000 pc: 0x2000002e msp: 0x20002000
verified 3028 bytes in 0.033063s (89.436 KiB/s)
** Verified OK **
** Resetting Target **
Info : Stlink adapter speed set to 240 kHz
adapter speed: 240 kHz
shutdown command invoked
```

9 Problèmes divers

9.1 Erreur édition de lien : (...) undefined reference to '__errno'

Cette erreur semble se produire lorsque la bibliothèque mathématique "m" ne fait pas partie de l'édition de lien.

Démarrer un ≪Build Project≫ et vérifier dans la console que la ligne d'édition de lien contient *-lm*.

L'ajouter éventuellement : Projet / Propriétés / C/C++ Build / Settings
Onglet ≪Tool Settings≫
MCU GCC Linker / Libraries
Cocher en bas de page ≪Use C math library (-lm)≫ (Voir *figure 3* (1)).

Si l'erreur persiste, Sur la même page, ajouter dans la fenêtre ≪Libraries (-l)≫ *m*.(Voir *figure 3* (2)).
Cela a pour effet de doubler l'option -lm, ce qui curieusement semble régler le problème.

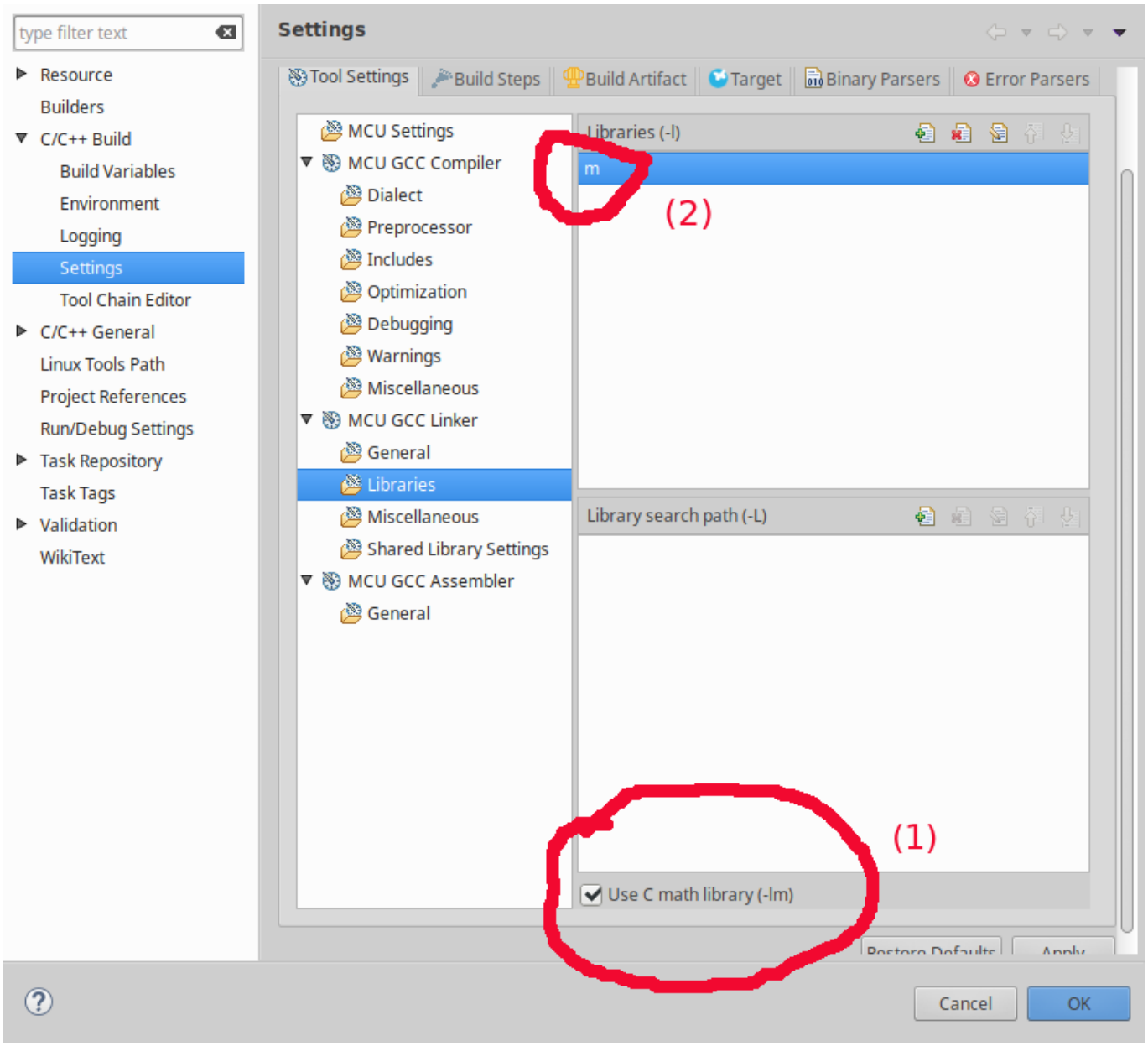

FIGURE 3 – Ajout de la bibliothèque mathématique pour l'édition de lien

Un exemple de ligne d'édition de lien :

```
arm-none-eabi-gcc -mcpu=cortex-m0plus -mthumb -mfloat-abi=soft -specs=nosys.specs -
    specs=nano.specs -T"../STM32L053R8Tx_FLASH.ld" -Wl,-Map=output.map -Wl,--gc-
    sections -o "projet_LCD_a_partir_S2_TDA01.elf" @"objects.list" -lm -lm
```