SUR

STM32F429-DISCO

Thierry JOUBERT

Octobre 2016

Table des matières

1 Contenu des répertoires

Le contenu du répertoire « tools » est le suivant :

- eclipse-cpp-neon-1a-win32-x86_64.zip : IDE Eclipse C/C++
- gcc-arm-none-eabi-5_4-2016q3-20160926-win32.zip : Toolchain de compilation croisée
- OpenOCD-0.9.0-Win32 : installateur de l'utilitaire de programmation et débogage
- STM32 ST-LINK Utility v3.9.0.exe : installateur du pilote USB pour la sonde JTAG on-board et de l'application ST-LINK fournie par ST Microelectronics
- Putty : utilitaire pour la communication via port série (Un autre terminal peut bien sûr être choisi)

Dans le sous répertoire Alternatif, on trouve des installateurs :

- eclipse-inst-win64 : installateur de l'IDE Eclipse C/C++
- arm-2010.09-51-arm-none-eabi : installateur de la toolchain de cross compilation GCC

Le contenu du répertoire « docs » est le suivant :

- DM00071990 : datasheet du STM32F429XX qui contient entre autres l'ensemble des fonctions alternatives (USART1, SPI, I2C, etc.)
- RM0090_Reference_manual : manuel de référence du STM32F429XX qui contient entre autres les fonctions et valeurs de l'ensemble des registres du microcontrôleur.
- STM32F429IDISCO_user_manual : contient le détail des différents composants disponibles sur la carte de démonstration ainsi que la façon dont ils sont connectés au microcontrôleur.
- L3GD20 : datasheet du gyroscope
- STMP811 : datasheet de l'expandeur d'IO

Le répertoire « thirdparties » contient tout le nécessaire pour la génération d'un firmware pour la carte d'évaluation (drivers pour les périphériques standards fournis par ST, les sources de FreeRTOS, etc.). Le sous-répertoire « STM32F4xx_StdPeriph_Driver » contient une API standard qui est fournie par ST et qui permet de simplifier l'initialisation et l'utilisation des périphériques du microcontrôleur.

Le répertoire « eclipse_sources_template » contient le squelette minimal permettant de générer un firmware déployable. Il convient ensuite de le modifier et de l'adapter aux besoins du projet.

Le répertoire « Dev » contient l'environnement de développement prêt à l'emploi et destiné à être copié à la racine du disque dur local.

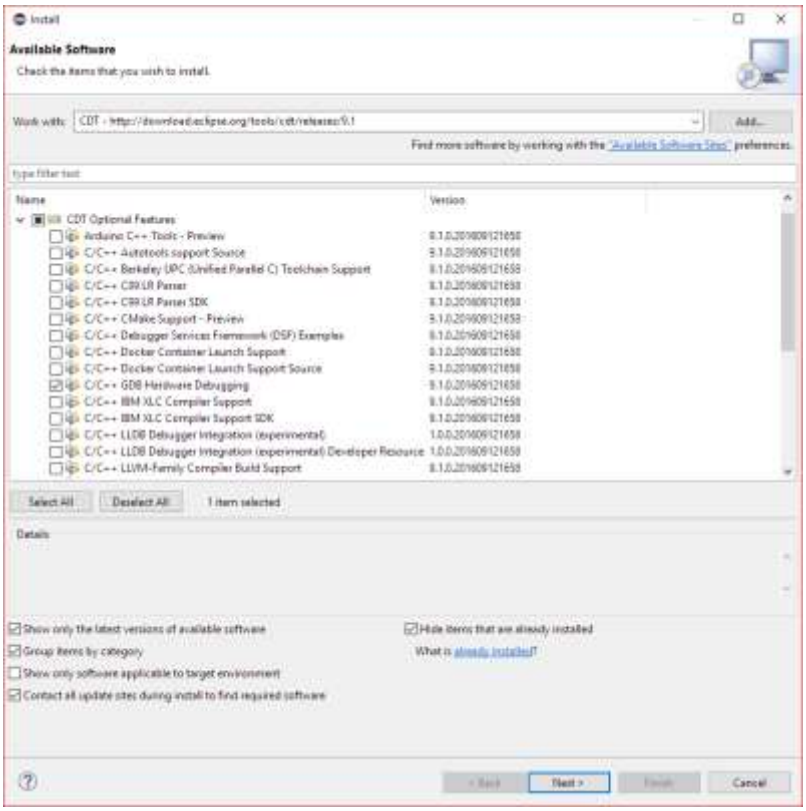
2 Installation des logiciels

Avant de pouvoir démarrer le développement d'applications, il est nécessaire d'installer les logiciels suivants :

1. La toolchain de cross compilation GCC ARM
2. L'utilitaire OpenOCD
3. Les drivers USB pour la sonde on-board. L'utilitaire installé permet également de tester la connectivité avec la carte de démonstration. (menu Target -> Connect)
4. Eclipse CDT C/C++ ainsi que le plugin « C/C++ GDB Hardware Debugging » qui nous permettra de se connecter à OpenOCD pour la programmation et le débogage du firmware

Les outils de compilation GCC, Eclipse et OpenOCD sont prêts à l'emploi dans le dossier « Dev ». Pour des raisons évidentes de portabilité d'un ordinateur à l'autre, tous les chemins d'accès dépendant de l'utilisateur local de la machine sont exclus et le dossier « Dev » peut être copié à la racine du disque local C:

Pour information, C/C++ GDB Hardware Debugging doit être ajouté après décompression de eclipse-cpp-neon-1a-win32-x86_64.zip ou installation de eclipse-inst-win64.exe, à l'aide du menu Help -> Install New Software

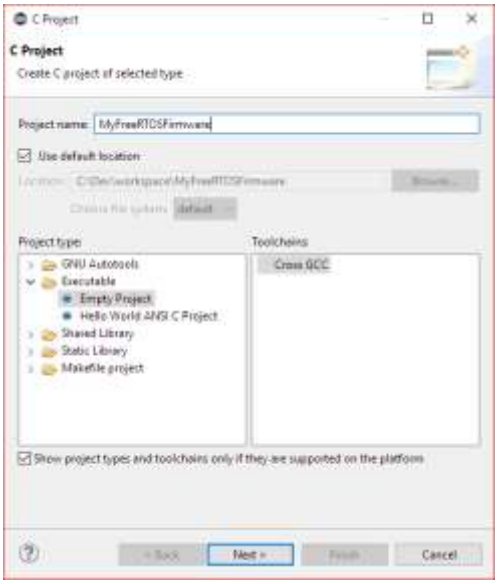


3 Création du firmware minimal

3.1 Création et configuration du projet Eclipse

La création du projet Eclipse se fait de la façon suivante :

1. Créer un nouveau projet C via la menu File -> New -> Projet C
2. Après avoir spécifié le nom du projet, sélectionner un type de projet « Executable -> Empty Project » et un toolchain de type « Cross GCC ».

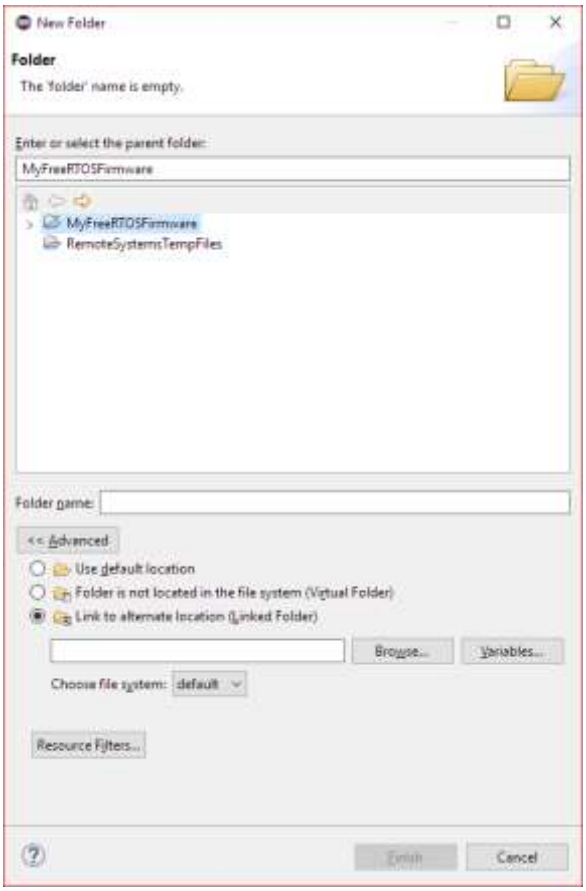


3. Après quelques étapes « next », définir la valeur du « Cross compiler prefix » à « arm-none-eabi- » et la valeur du « Cross compiler path » à « C:\Dev\gcc-arm-none-eabi ». Ce dernier est nécessaire si plusieurs GCC sont disponibles sur la machine de développement.

3.2 Importer les ressources nécessaires pour créer un firmware

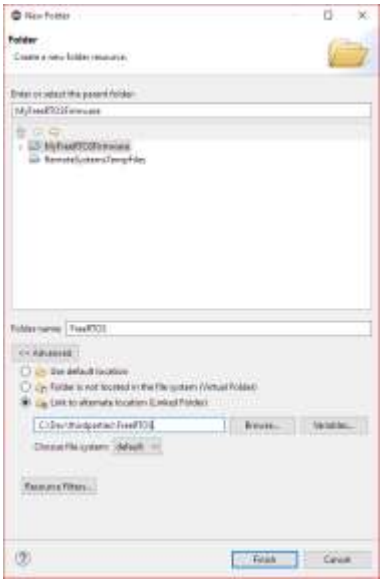
Comme ces ressources sont fournies par de tierces parties, il est conseillé de les ajouter via des « linked resources » plutôt que de déplacer les répertoires dans le workspace Eclipse. Cela permet de partager des sources entre divers projets et/ou espace de travail.

L'ajout de « linked resources » se fait via le menu « File -> New -> Folder ». La boite de dialogue suivante va permettre l'ajout de l'ensemble des ressources externes nécessaires à la génération d'un firmware :

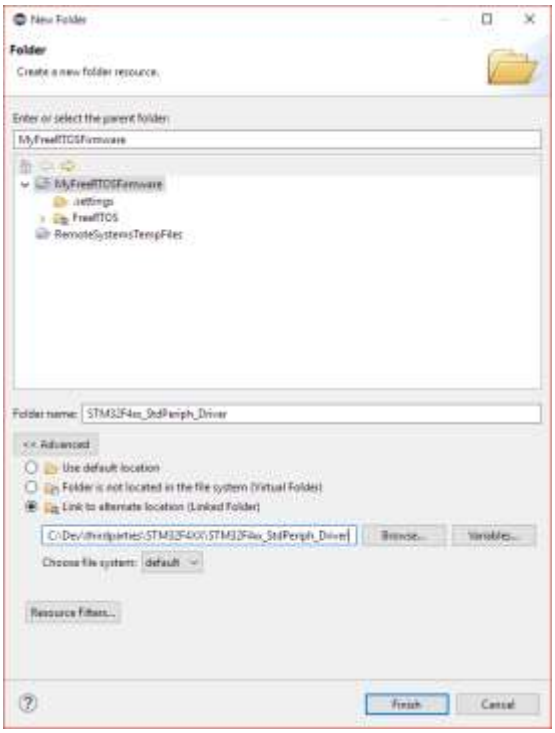


Les ressources à ajouter sont les suivantes :

- Le répertoire : « FreeRTOS »



- Le répertoire : « STM32F4xx_StdPeriph_Driver »

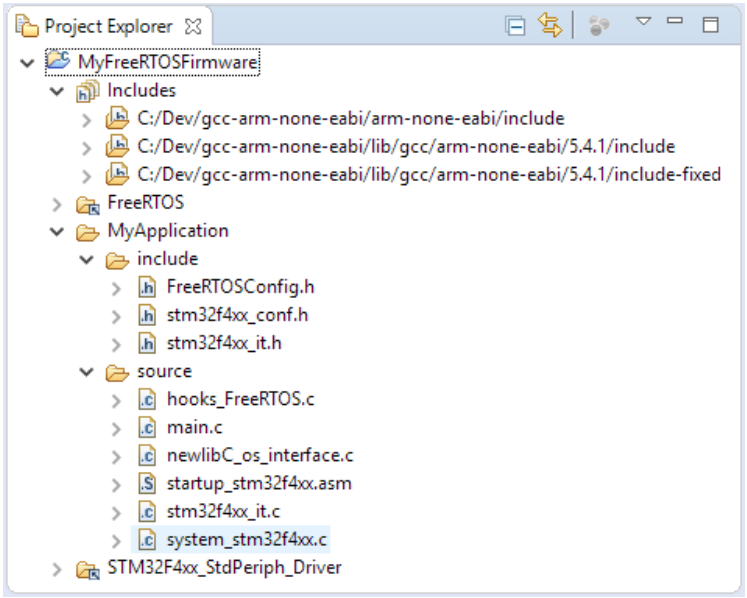


3.3 Création d'un répertoire pour les sources du projet

Il est maintenant nécessaire de créer un répertoire destiné à contenir les sources spécifiques du firmware qui va être généré. Il est ensuite possible de le compléter avec un template de base.

Le plus simple est d'importer le dossier MyApplication depuis le dossier « eclipse_sources_template »

A ce stade, le projet Eclipse créé correspond à l'illustration suivante :



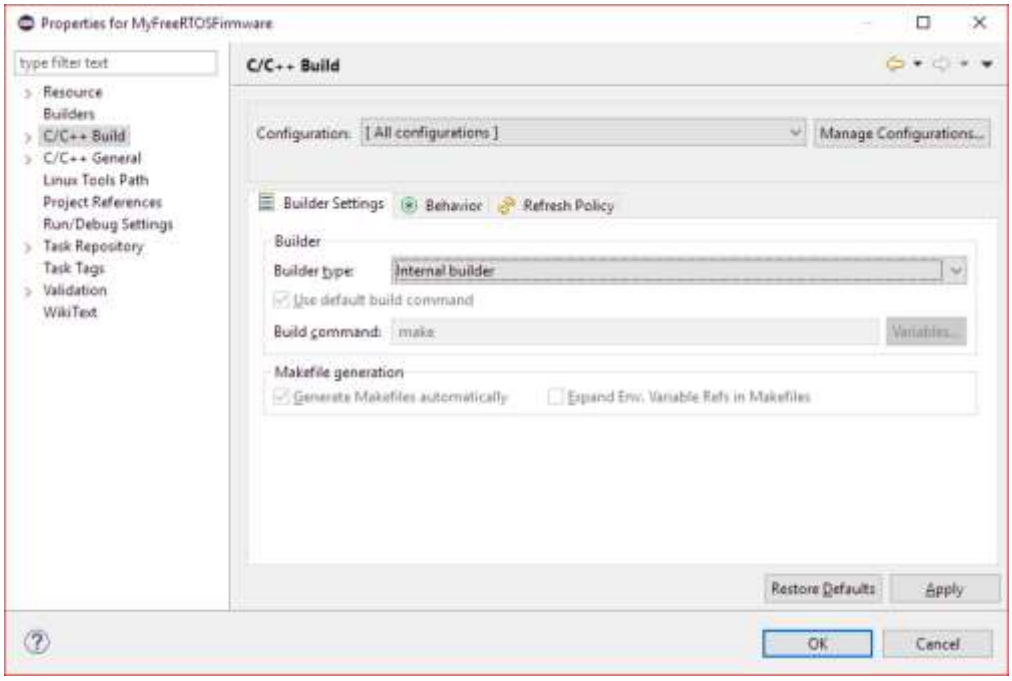
4 Configuration et compilation du firmware

Cette section explicite l'ensemble des configurations qu'il est nécessaire d'effectuer pour pouvoir générer un firmware.

ATTENTION : le paramétrage peut être effectué simultanément sur les deux configuratiuon Debug et Release.

4.1 Utilisation de la mécanique interne d'Eclipse pour la compilation

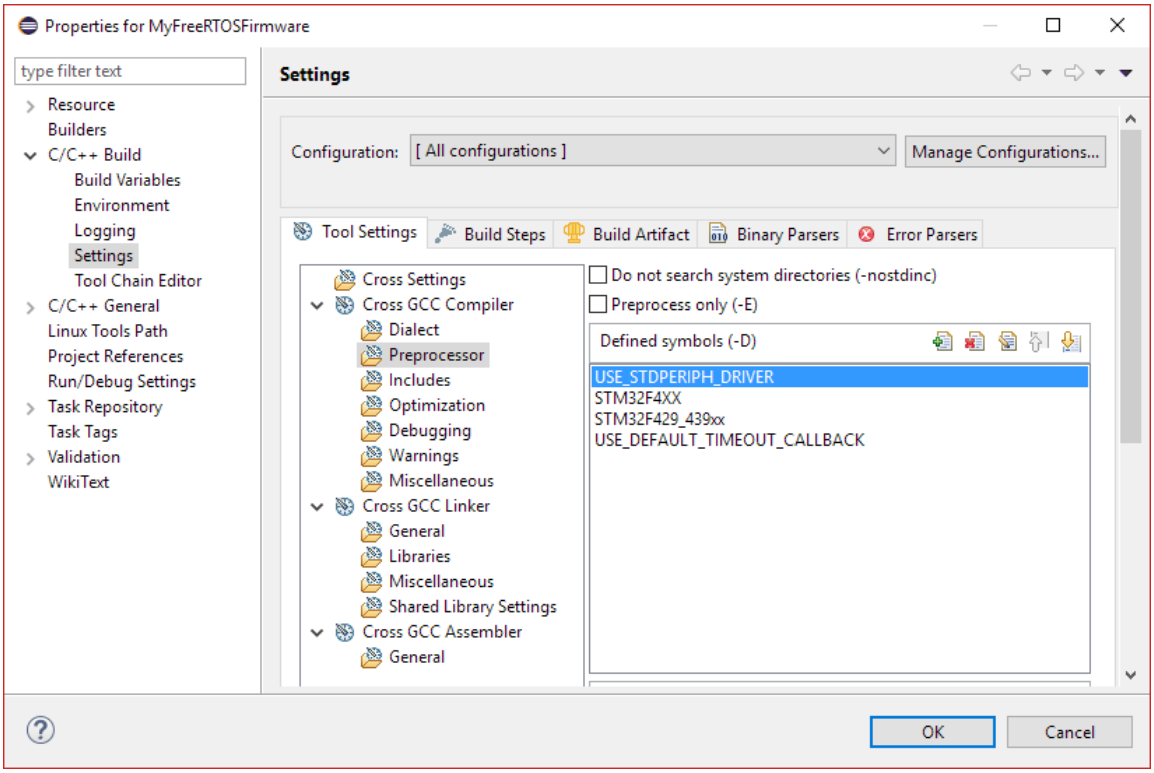
Par défaut, les projets Eclipse sont créés avec un « builder type » de type « external » permettant l'utilisation de make ou autres utilitaires de la sorte. Dans notre cas, nous utilisons la mécanique interne d'Eclipse. Il est donc nécessaire de redéfinir ce paramètre via le menu des propriétés du projet.



4.2 Configuration du compilateur

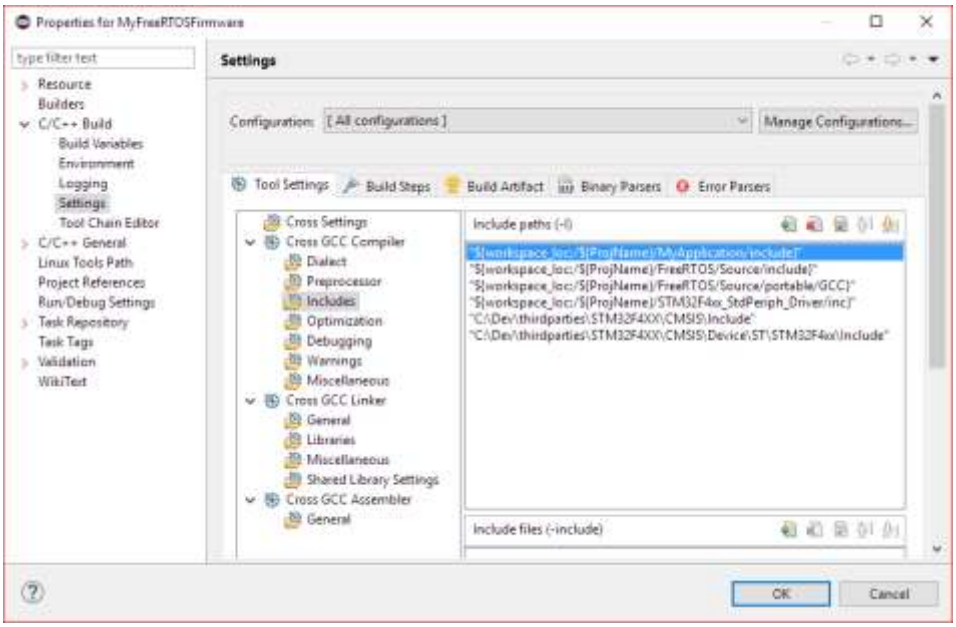
4.2.1 Configuration des symboles de compilation (option -D)

Il est nécessaire de définir un certain nombre de symboles spécifiques à la cible qui est utilisée. Cette configuration peut s'effectuer par le menu « C/C++ Build -> Settings -> Cross GCC Compiler -> Preprocessor » du menu des propriétés du projet.



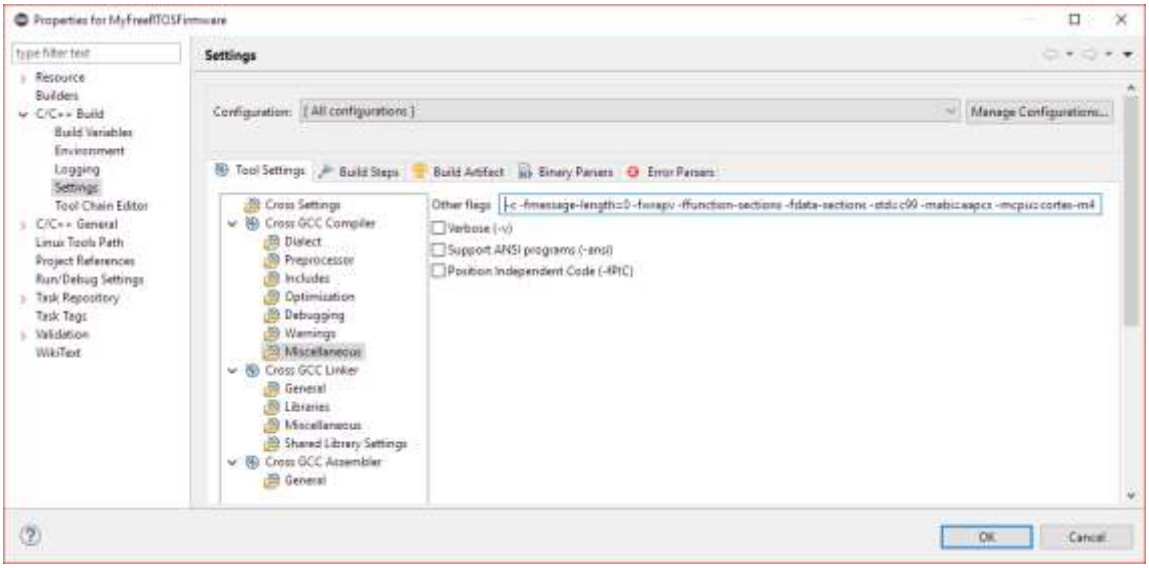
4.2.2 Configuration des « include » nécessaires (option -I)

Il est nécessaire d'ajouter des « include path » afin de satisfaire toutes les définitions. Ces derniers serviront à la fois à l'indexer Eclipse mais aussi à la compilation du firmware. Les « include path » peuvent être configurés via le menu « C/C++ Build -> Settings -> Cross GCC Compiler -> Includes » du menu des propriétés du projet.



4.2.3 Configuration des autres flags spécifiques à la cible

Les options suivantes « -c -fmessage-length=0 -fwrapv -ffunction-sections -fdata-sections -std=c99 -mabi=aapcs -mcpu=cortex-m4 -mlittle-endian -fdollars-in-identifiers -mthumb -mfpu=fpv4-sp-d16 -mfloat-abi=softfp » sont à ajouter via le menu « C/C++ Build -> Settings -> Cross GCC Compiler -> Miscellaneous » du menu des propriétés du projet.

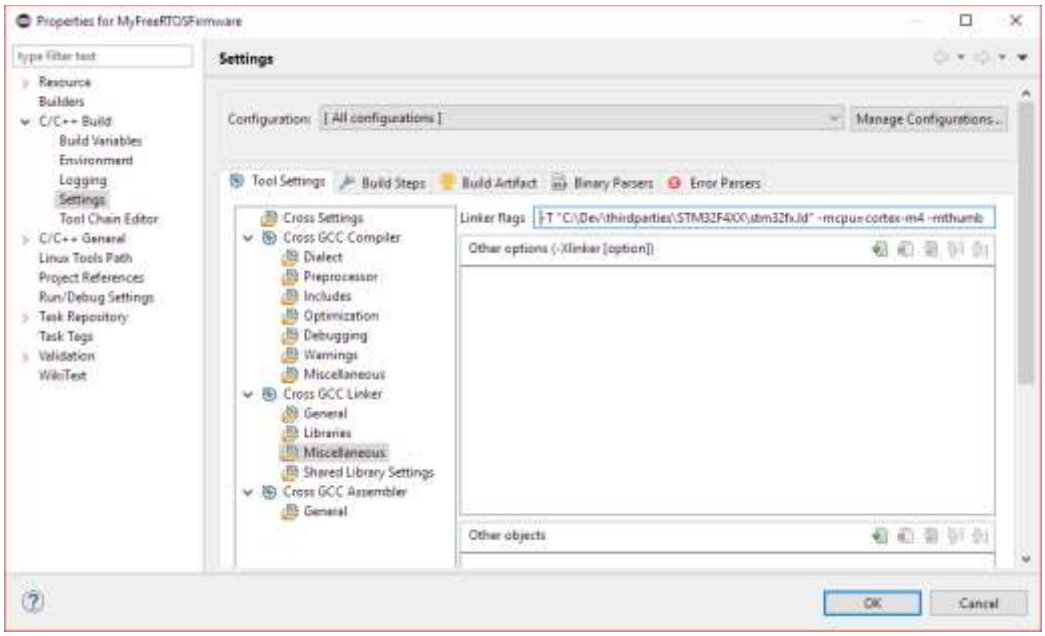


4.3 Configuration du linker

L'ajout des flags du linker se fait via le menu « C/C++ Build -> Settings -> Cross GCC Linker -> Miscellaneous » du menu des propriétés du projet.

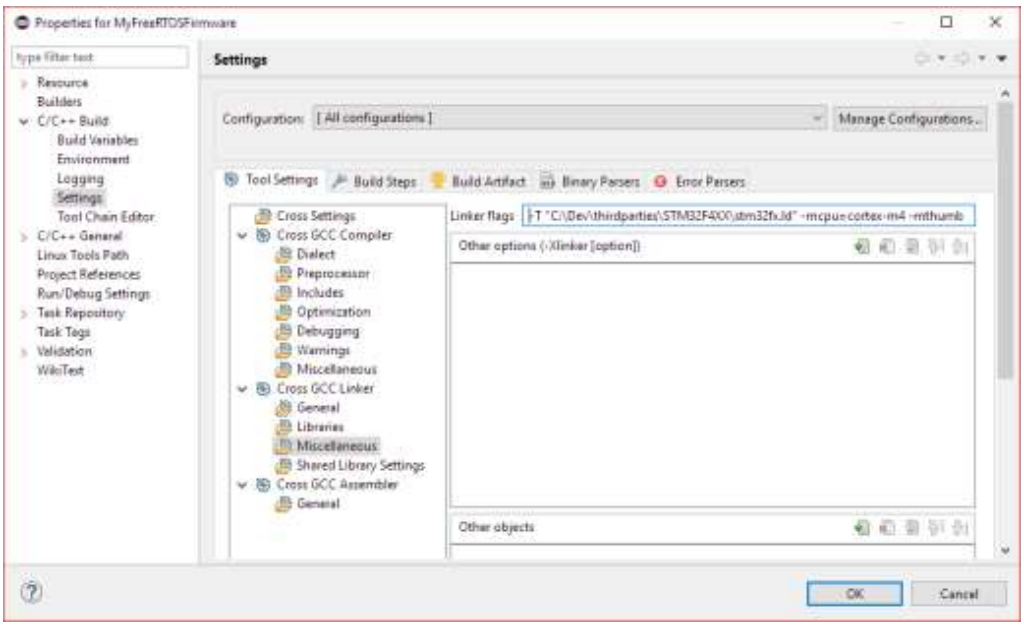
La configuration à ajouter est la suivante :

« -T "C:\Dev\thirdparties\STM32F4XX\stm32fx.ld" -mcpu=cortex-m4 -mthumb »



4.4 Configuration de l'Assembler

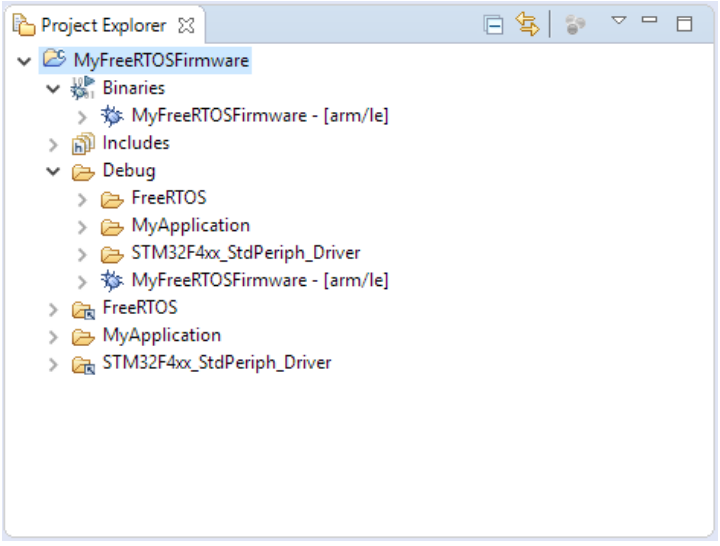
Les options suivantes « -mcpu=cortex-m4 -g » sont à ajouter via le menu « C/C++ Build -> Settings -> Cross GCC Assembler -> General » du menu des propriétés du projet.



4.5 Compilation du firmware

Il est maintenant possible de lancer la compilation du firmware via le menu contextuel du projet. (Build Project)

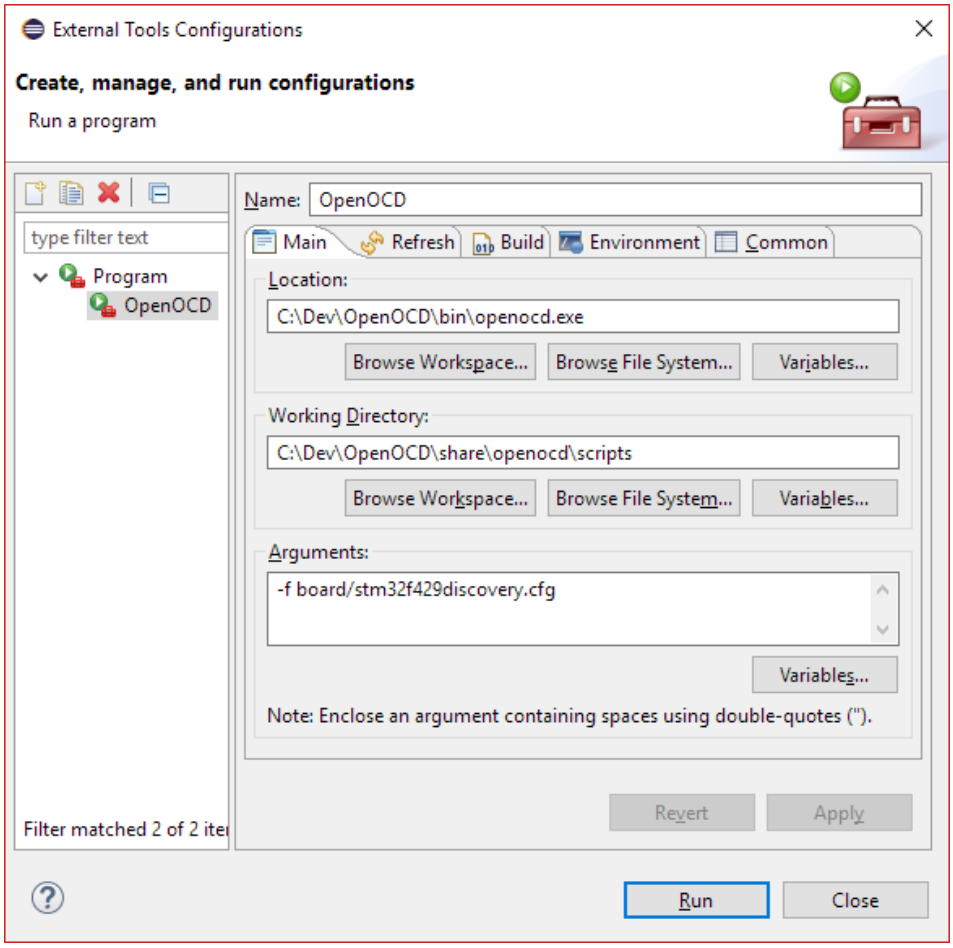
Une fois la compilation terminée avec succès, le firmware nouvellement généré apparait dans l'explorateur de projet. C'est ce dernier que nous allons pouvoir déployer sur la cible.



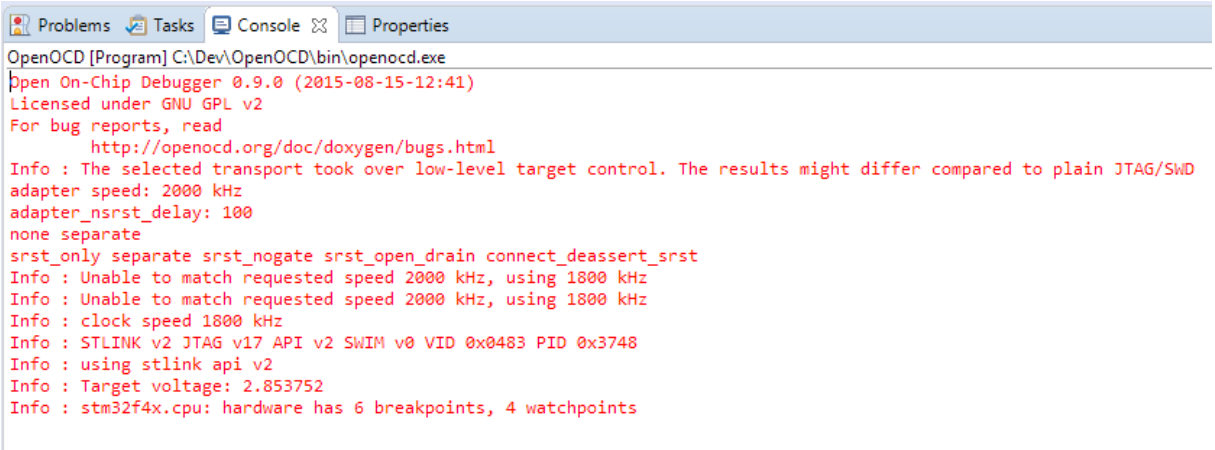
5 Déploiement et débogage du firmware

5.1 Création de la « External Tools configuration » pour OpenOCD

Afin de démarrer OpenOCD via Eclipse, il est nécessaire de passer par la fonctionnalité qui permet de lancer une application externe. Cette fonctionnalité est accessible via la menu « Run -> External Tools -> External Tools Configurations… »



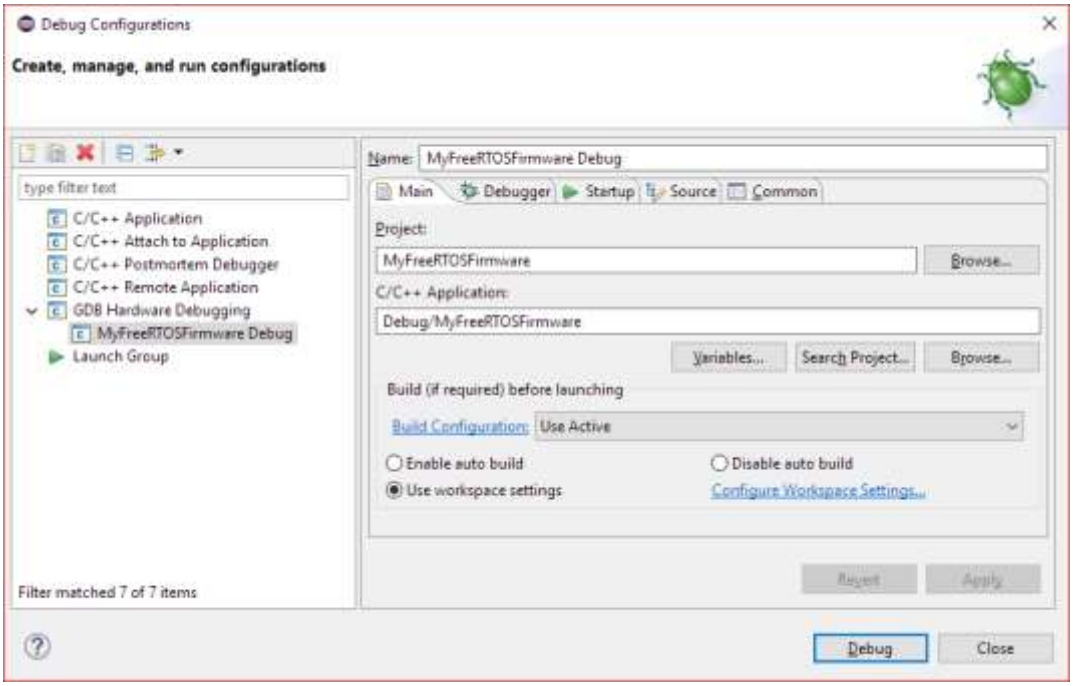
Pour vérifier la configuration, il suffit de cliquer sur le bouton ‘Run’. Le résultat dans la console Eclipse est le suivant :



5.2 Création de la « debug configuration » pour déployer et déboguer l'application

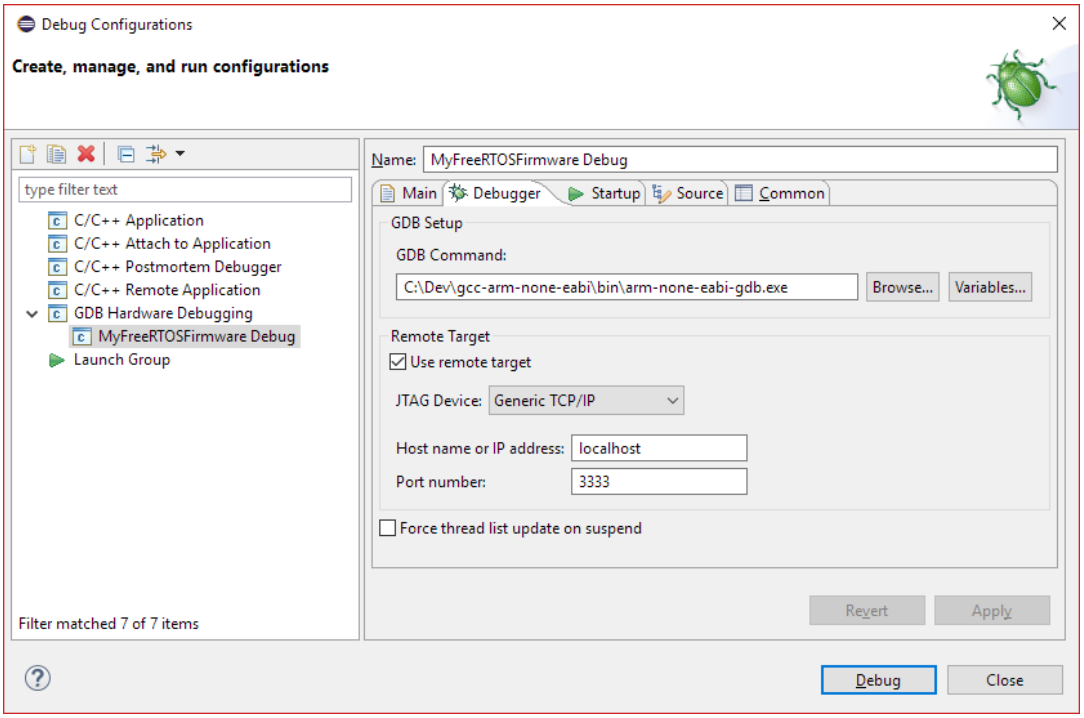
Avant de pouvoir lancer le déploiement et le débogage du firmware qui a été généré, il est nécessaire d'ajouter une « Debug configuration » de type « GDB Hardware Debugging ». Cette fonctionnalité est accessible via le menu « Run -> Debug Configurations … ».

5.2.1 Onglet « Main »



5.2.2 Onglet « Debugger »

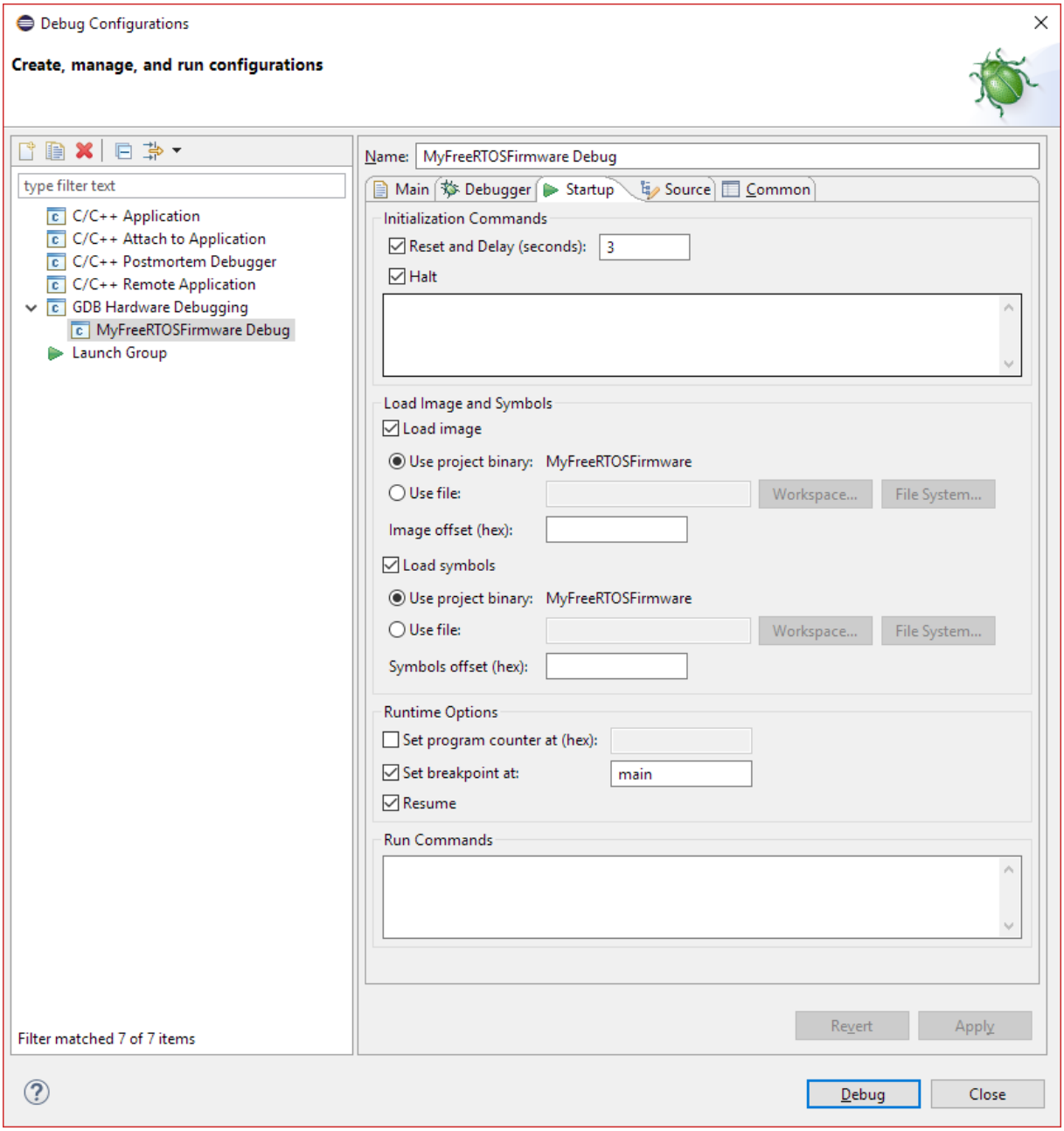
Dans cet environnement, GDB se connecte à OpenOCD via une connexion TCP. Il est donc nécessaire de connaitre la configuration utilisée lors du lancement d'OpenOCD. Nous utilisons ici la configuration par défaut.



5.2.3 Onglet « Startup »

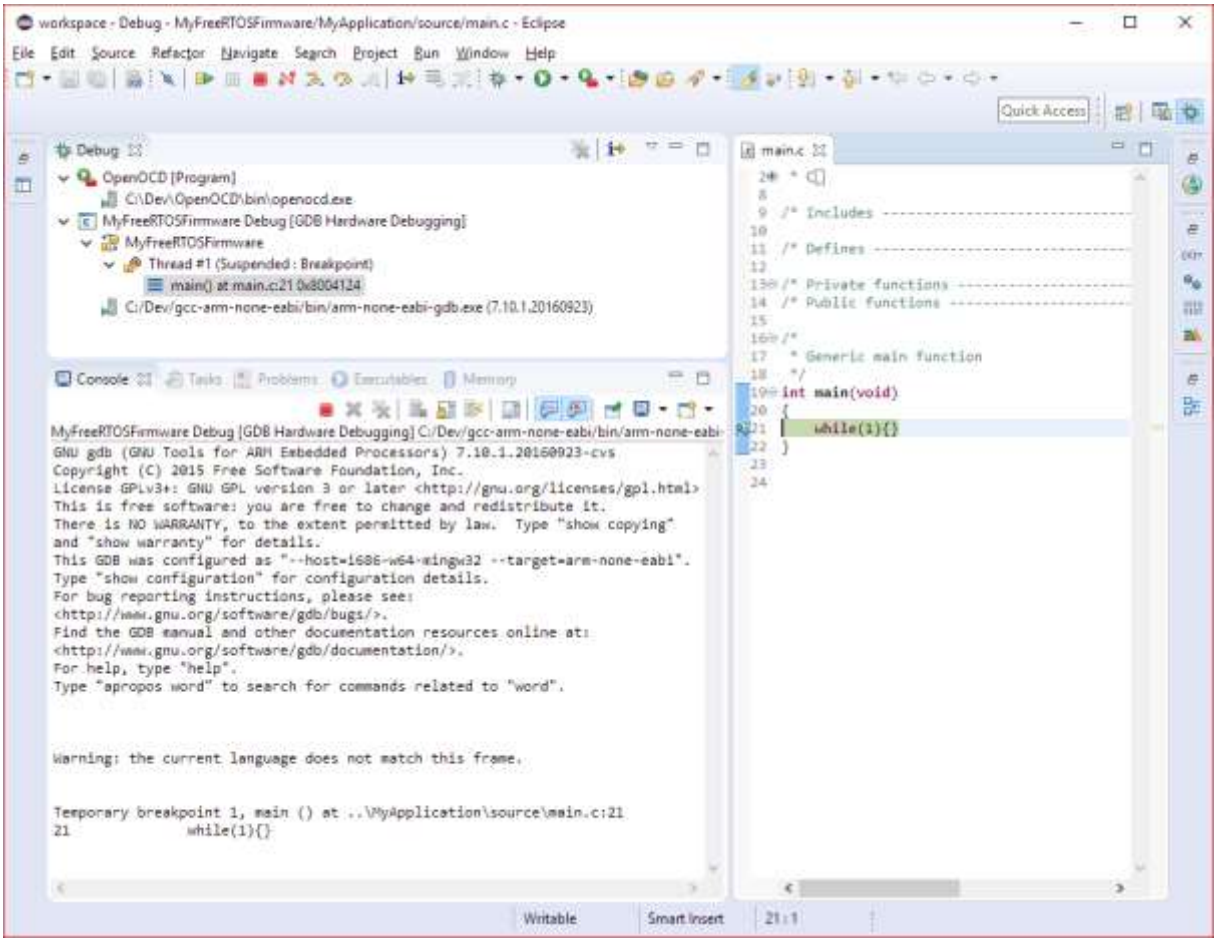
La configuration que nous utilisons est la suivante :

- Load image : indique que le firmware va être déployé sur la cible avant le lancement de chaque session de débogage. Si le firmware a déjà été déployé précédemment, il est possible de sauter cette étape en décochant cette case.
- Load symbols : indique le fichier à utiliser par GDB pour le chargement des symboles pour la session de débogage
- Set breakpoint at main : indique que l'on souhaite automatiquement ajouter un breakpoint dans la fonction « main » du firmware. On peut ici n'importe quelle fonction du firmware
- Resume : indique que GDB lance automatiquement l'exécution



5.3 Déploiement et exécution du firmware

Il est maintenant possible de déployer et déboguer le firmware via le bouton « Debug » présent sur l'illustration précédente ou via la toolbar d'Eclipse. Si le lancement du débugger se déroule avec succès, Eclipse proposera de basculer la perspective en mode debug comme illustré dans la figure suivante.



6 Tâches FreeRTOS et utilisation des ressources du microcontrôleur

6.1 Convention de nommage des routines d'interruption

Pour que la routine d'interruption soit fonctionnelle, elle doit être définie de la même façon que lors de l'initialisation du vecteur d'interruption. Cette initialisation se trouve dans le fichier « startup_stm32f4xx.asm ».

Voici un extrait de ce dernier :

```
  .word     TIM2_IRQHandler               /* TIM2                        */
  .word     TIM3_IRQHandler               /* TIM3                        */
  .word     TIM4_IRQHandler               /* TIM4                        */
  .word     I2C1_EV_IRQHandler            /* I2C1 Event                  */
  .word     I2C1_ER_IRQHandler            /* I2C1 Error                  */
  .word     I2C2_EV_IRQHandler            /* I2C2 Event                  */
  .word     I2C2_ER_IRQHandler            /* I2C2 Error                  */
  .word     SPI1_IRQHandler               /* SPI1                        */
  .word     SPI2_IRQHandler               /* SPI2                        */
  .word     USART1_IRQHandler             /* USART1                      */
```

6.2 Interagir avec les sorties numériques du microcontrôleur

D'après les inscriptions présentent sur la carte (information également consultable dans le document « STM32F429IDISCO_user_manual » en section 6.3), on constate que les LED mises à disposition de l'utilisateur sont connectées au port GPIOG et plus particulièrement aux pins 13 et 14. Le fait de modifier l'état de sortie de ces deux pates nous permettra donc d'interagir avec les LEDs connectées.

Nous allons donc créer une tâche FreeRTOS qui va prendre en charger l'initialisation du port GPIOG, la configuration des pins et la mise à jour du registre de sortie.

```
/* Includes ------------------------------------------------------------------*/
#include "FreeRTOS.h"
#include "task.h"
#include "stm32f4xx.h"

/* Defines -------------------------------------------------------------------*/

/* Private functions ---------------------------------------------------------*/

void vTaskBlink( void * pvParameters )
{
    GPIO_InitTypeDef  GPIO_InitStructure;

    /* Enable the GPIOG Clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOG, ENABLE);

    /* Configure the GPIOG pin 13/14 which are connected to user LED using ST API */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Init(GPIOG, &GPIO_InitStructure);

    for( ;; )
    {
        vTaskDelay(1000);
        // Make XOR on output data register to make LEDs blink
        GPIOG->ODR ^= GPIO_Pin_13;
        GPIOG->ODR ^= GPIO_Pin_14;
    }
}

/* Public functions ----------------------------------------------------------*/

/*
 * Generic main function
 */
```

```
int main(void)
{
    // Enable fault handlers
    SCB->SHCSR |= SCB_SHCSR_MEMFAULTENA_Msk;
    SCB->SHCSR |= SCB_SHCSR_BUSFAULTENA_Msk;
    SCB->SHCSR |= SCB_SHCSR_USGFAULTENA_Msk;

    // Create the task dedicated to LED blink
    xTaskCreate( vTaskBlink, NULL, 128, NULL, tskIDLE_PRIORITY + 1, NULL );

    // Start the FreeRTOS task scheduler
    vTaskStartScheduler();
}
```

6.3 Communiquer via une liaison série

Afin de pouvoir établir une communication série entre la carte de démonstration et le PC de développement, il faut dans un premier temps satisfaire les points suivants :

- Un convertisseur USB – Serial 3v3
- Un USART disponible sur le microcontrôleur
- Termite ou Putty pour la lecture/écriture série sur le PC de développement

Le microcontrôleur STM32F4 dispose de 3 USARTs. Chacun d'entre eux peut être connecté à plusieurs port GPIO et plusieurs pins spécifiques en fonction des besoins. Il est donc nécessaire, en fonction des périphériques et des pins déjà utilisés par l'application, de déterminer quelle USART est disponible et utilisable.

Pour déterminer quel USART utiliser, il est possible de se référer au document « DM00071990 » qui décrit l'ensemble des fonctions alternatives pour chaque GPIO. Dans ce document, on trouve toutes les combinaisons de GPIO pour les 3 USARTs :

- USART1 : TX en PA9 et RX en PA10, ou TX en PB6 et RX en PB7
- USART2 : TX en PA2 et RX en PA3, ou TX en PD5 et RX en PD6
- USART3 : TX en PB10 et RX en PB11, ou TX en PC10 et RX en PC11, ou TX en PD8 et RX en PD9

Le chapitre 6.12 du document STM32F429IDISCO_user_manual indique que le seul USART disponible est celui en PA9 et PA10. D'autres choix peuvent être pris au détriment de périphériques utilisables sur la carte STM32F429IDISCO.

C'est donc PA9 et PA10qui vont être utilisées pour établir un lien série entre le PC de développement et la carte de démonstration.

Voici le code simple d'une tâche qui permet l'initialisation du port GPIOA, la connexion des pins 9/10 à l'USART1, la configuration de l'USART1 puis la lecture et écriture sur la liaison série. Dans le cas présent, la tache va simplement renvoyer le caractère reçu à l'émetteur.

La configuration de la liaison série est la suivante :

- 115200 baud
- 8 data bits
- 1 stop bit
- No parity
- No hardware flow control

Initialisation de l'USART

```
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_rcc.h"

/* Defines -------------------------------------------------------------------*/
typedef enum E_USART_Msg_type {
        USART_MSG_BUTTON_PUSHED,
        USART_MSG_ADC_CONVERTION,
        USART_MSG_CHAR_RECEVIED,
        USART_MSG_I2C_VALUE,
        USART_MSG_LAST
} USART_Msg_type;

typedef struct S_USART_Msg {
        USART_Msg_type msg_type;
        uint16_t       adc;
        uint16_t       temp;
    char           usart;
} USART_Msg;

/* Globals -------------------------------------------------------------------*/
static xQueueHandle USARTQueue;


void initUSART(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    USART_InitTypeDef USART_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    /* init working queue */
    USARTQueue = xQueueCreate(50,sizeof(USART_Msg));

    if(USARTQueue == NULL) {
        // Error while creating queue
        return;
    }

    /* Enable GPIO and USART clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

    /* Connecting GPIOA 9/10 to alternate function USARTx_Tx/Rx */
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);

    /* Configure GPIOA pin 9/10*/
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure USART */
    USART_InitStructure.USART_BaudRate = 115200;
    USART_InitStructure.USART_WordLength = USART_WordLength_8b;
    USART_InitStructure.USART_StopBits = USART_StopBits_1;
    USART_InitStructure.USART_Parity = USART_Parity_No;
    USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
    USART_InitStructure.USART_Mode = USART_Mode_Tx | USART_Mode_Rx;
    USART_Init(USART1, &USART_InitStructure);

    /* Enable USART */
    USART_Cmd(USART1, ENABLE);

    /* Enable RXNE interrupt, IRQHandler will be called when data available */
    USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);

    /* Add USART1 IRQ to NVIC */
    NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
```

```
    NVIC_Init(&NVIC_InitStructure);

    USART_Cmd(USART1, ENABLE);

    while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
}
```

6.3.1 Implémentation de l'ISR USART

```
/*
 * This IRQHandler is called when USART1 receive buffer is not empty.
 */
void USART1_IRQHandler(void)
{
        char rx;

    /* Check if interrupt was because data is received */
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        if (USART_GetFlagStatus(USART1, USART_SR_RXNE) != RESET)
        {
            rx = USART_ReceiveData(USART1);
              /* code to handle the received data */
        }
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);

        USART_SendData(USART1, (uint8_t) rx);

        while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
    }
}
```

6.3.2 Utilisation de « printf »

Une fois l'USART initialisé et prêt pour la transmission, il est possible de l'utiliser eu guise de « sortie standard ». Ainsi, tous les appels « printf » peuvent se traduire par l'envoi de la chaine de caractères résultante sur la liaison série.

Afin de rendre cela possible, il est nécessaire de compléter le squelette présent dans le fichier « newlibC_os_interface.c » qui était utilisé uniquement pour rendre la génération du firmware possible. (ie fournir les interfaces manquantes à la libc)

Une des adaptations possibles est la suivante :

```
/* called by newlibC for printf  */
int fputc(int ch, FILE *f)
{
    USART_SendData(USART1, (uint8_t) ch);
    while (USART_GetFlagStatus(USART1, USART_FLAG_TC) == RESET);
    return ch;
}

int _write(int file, char *ptr, int len)
{
    int remaining = len;
    while(--remaining >= 0){
        fputc(*ptr, (FILE*)0);
        ++ptr;
    }
    return len;
}
```

6.4 Utiliser les timers

6.4.1 Utilisation des timers pour générer des interruptions

Dans cet exemple, le timer 2 va être configuré pour générer une interruption à une fréquence de 1Hz. Attention, les valeurs (prescaler, etc.) sont susceptibles d'être adaptées en fonction de l'environnement utilisé.

```
void initTimer(void)
{
    NVIC_InitTypeDef NVIC_InitStructure;
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;

    /* Enable TIM2 Clock */
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);

    /* TIMPRE not set in RCC register */
    /* AP1 prescaler set to 4 so timer clock is 2 * APB clock(45MHz) = 90MHz */
    /* Configure timer for 1Hz frequency */
    TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);
    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInitStructure.TIM_Prescaler = (90000000 / 1000000) - 1;
    TIM_TimeBaseInitStructure.TIM_Period = 1000000 - 1;   /* TIM2 is 32bit counter */
    TIM_TimeBaseInit(TIM2, &TIM_TimeBaseInitStructure);

    /* Enable TIMER interrupt on counter reload */
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);

    TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);

    /* Add TIM2 IRQ to NVIC */
    NVIC_InitStructure.NVIC_IRQChannel = TIM2_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init( &NVIC_InitStructure );

    TIM_Cmd(TIM2, ENABLE);
}
```

6.4.2 Implémentation de l'ISR Timer

Dans cette ISR, on va simplement agir sur les IOs connectées aux LED utilisateur. C'est une méthode alternative qui nous permet de faire clignoter les LED à intervalle régulier sans passer par une tâche FreeRTOS.

```
/*
 * This IRQHandler is called when TIM2 counter is reloaded.
 */
void TIM2_IRQHandler(void)
{
    if(TIM_GetITStatus(TIM2, TIM_IT_Update))
    {
        GPIOG->ODR ^= GPIO_Pin_13;
        GPIOG->ODR ^= GPIO_Pin_14;
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
    }
}
```

6.4.3 Générer un signal PWM dans une tâche

Dans cette section, l'exemple démontre comment utiliser le timer TIM1 pour générer un signal PWM sur le canal de sortie 1 afin de piloter la luminosité d'une LED.

```
void initTimerForPWM(void)
{
    TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStructure;
    TIM_OCInitTypeDef TIM_OCStruct;
    GPIO_InitTypeDef  GPIO_InitStructure;

    /* Enable Timer1 */
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
```

```
    /* Enable the GPIOE Clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);

    /* Configure the GPIOE 9 to TIM1 alternate function (output channel 1) */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOE, &GPIO_InitStructure);

    GPIO_PinAFConfig(GPIOE, GPIO_PinSource9, GPIO_AF_TIM1);

    /* TIMPRE not set in RCC register */
    /* AP1 prescaler set to 4 so timer clock is 2 * APB clock(90MHz) = 180MHz */
    /* Configure timer for 100Hz frequency */
    TIM_TimeBaseStructInit(&TIM_TimeBaseInitStructure);
    TIM_TimeBaseInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInitStructure.TIM_Prescaler = (180000000 / 1000000) - 1;
    TIM_TimeBaseInitStructure.TIM_Period = 10000 - 1;
    TIM_TimeBaseInit(TIM1, &TIM_TimeBaseInitStructure);

    /* Configure TIM1 Output Channel for 15% duty */
    TIM_OCStructInit(&TIM_OCStruct);
    TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM1;
    TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
    TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_High;
    TIM_OCStruct.TIM_Pulse = 1500 - 1;
    TIM_OC1Init(TIM1, &TIM_OCStruct);
    TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable);

    /* Enable TIM1 */
    TIM_Cmd(TIM1, ENABLE);

    /* Enable TIM1 outputs */
    TIM_CtrlPWMOutputs(TIM1, ENABLE);
}
```

Il est ensuite possible de faire évoluer la luminosité de la LED en ajustant la valeur TIM_Pulse.

```
void vTaskLEDBrightness(void * pvParameters)
{
        uint32_t cc1value = 1;

        for(;;)
        {
                vTaskDelay(50);
                TIM_SetCompare1(TIM1, cc1value);
                cc1value += 100;
                if(cc1value > 10001)
                        cc1value = 1;
        }
}
```

6.4.4 Modification du PWM via des entrées USART

Pour permettre d'incrémenter et décrémenter la valeur du PWM à l'aide d'entrée USART, il faut créer une tache gérant une queue de message. Cette tâche sera complétée dans les chapitres suivants pour traiter d'autres messages.

```
void vTaskhandleEvents(void * pvParameters)
{
    uint32_t pwm_duty = TIM_GetCapture1(TIM1);
    static const uint32_t pwm_duty_inc = 100;

    for ( ;; )
    {
        USART_Msg message;

        if(xQueueReceive( USARTQueue, (void*)&message, portMAX_DELAY))
        {
            switch(message.msg_type)
            {
            case USART_MSG_CHAR_RECEVIED:
                if(message.usart == 'p')
                {
                    printf("Increasing brightness\n");
                    if(pwm_duty < 10000)
                    {
                        pwm_duty += pwm_duty_inc;
                        TIM_SetCompare1(TIM1, pwm_duty);
                    }
                }
                else if(message.usart == 'm')
                {
                    printf("Decreasing brightness\n");
                    if(pwm_duty < pwm_duty_inc)
                    {
                        pwm_duty = 0;
                    }
                    else
                    {
                        pwm_duty -= pwm_duty_inc;
                    }
                    TIM_SetCompare1(TIM1, pwm_duty);
                }
                else
                    printf("Received from USART :%c\n", message.usart);
                break;
            default:
                printf("MSG not supported\n");
            }
        }
    }
}
```

Le handler d'interruption de USART1 doit aussi utiliser la queue de message

```
void USART1_IRQHandler(void)
{
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
    USART_Msg message;

    /* Check if interrupt was because data is received */
    if(USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
    {
        if (USART_GetFlagStatus(USART1, USART_SR_RXNE) != RESET)
        {
            message.msg_type = USART_MSG_CHAR_RECEVIED;
            message.usart = USART_ReceiveData(USART1);

            if(USARTQueue)
            {
                xQueueSendFromISR(USARTQueue, (void*)&message, &xHigherPriorityTaskWoken);
            }
```

```
        }
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
    }

    portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
}
```

6.5 Communiquer via SPI

Le code suivant réalise un échange simple avec le gyroscope accessible via l'interface SPI5 du microcontrôleur afin de récupérer son device ID.

6.5.1 Initialisation SPI

```
void initSPI(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    SPI_InitTypeDef SPI_InitStruct;
    uint16_t dummy;

    /* Enable GPIOC, GPIOF and SPI5 clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI5, ENABLE);

    /* Configure GPIOF 7/8/9 */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_8 | GPIO_Pin_9;
    GPIO_Init(GPIOF, &GPIO_InitStructure);

    /* Enable SPI5 alternate function */
    GPIO_PinAFConfig(GPIOF, GPIO_PinSource7, GPIO_AF_SPI5);
    GPIO_PinAFConfig(GPIOF, GPIO_PinSource8, GPIO_AF_SPI5);
    GPIO_PinAFConfig(GPIOF, GPIO_PinSource9, GPIO_AF_SPI5);

    /* Configure GPIOC1. Used for L3GD20 CS */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOC, &GPIO_InitStructure);
    GPIO_SetBits(GPIOC, GPIO_Pin_1);

    /* Configure SPI5 */
    SPI_InitStruct.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    SPI_InitStruct.SPI_Mode = SPI_Mode_Master;
    SPI_InitStruct.SPI_DataSize = SPI_DataSize_8b;
    SPI_InitStruct.SPI_CPOL = SPI_CPOL_Low;
    SPI_InitStruct.SPI_CPHA = SPI_CPHA_1Edge;
    SPI_InitStruct.SPI_NSS = SPI_NSS_Soft | SPI_NSSInternalSoft_Set;
    SPI_InitStruct.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_8;
    SPI_InitStruct.SPI_FirstBit = SPI_FirstBit_MSB;
    SPI_Init(SPI5, &SPI_InitStruct);

    /* Enable SPI5 */
    SPI_Cmd(SPI5, ENABLE);

    // Read 'WHO AM I' register (address : 0x0F)
    uint8_t device_id = read_spi_register(0x0F);

    // check for L3GD20 device (b11010100 from datasheet)
    if(dummy == 0b11010100)
        printf("L3GD20 found on SPI (dev id : 0b11010100)\n");
    else
        printf("No L3GD20 found on SPI !!!\n");
}
```

6.5.2 Fonctions simples de lecture et d'écriture via SPI

```
void write_spi_register(uint8_t register_addr, uint8_t reg_value)
{
    // Device CS goes low to start transmission
    GPIO_ResetBits(GPIOC, GPIO_Pin_1);

    // Send register address
    SPI_I2S_SendData(SPI5, register_addr & 0x3F);
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_TXE) == RESET);
```

```
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_RXNE) == RESET);
    SPI_I2S_ReceiveData(SPI5); // Dummy read

    // Send register value
    SPI_I2S_SendData(SPI5, reg_value); // Write register value
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_TXE) == RESET);
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_RXNE) == RESET);
    SPI_I2S_ReceiveData(SPI5); // Dummy read

    // Device CS goes to high to stop transmission
    GPIO_SetBits(GPIOC, GPIO_Pin_1);
}

uint8_t read_spi_register(uint8_t register_addr)
{
    uint8_t reg_value = 0;

    // Device CS goes low to start transmission
    GPIO_ResetBits(GPIOC, GPIO_Pin_1);

    // Send register address
    SPI_I2S_SendData(SPI5, register_addr | 0x80);
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_TXE) == RESET);
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_RXNE) == RESET);
    SPI_I2S_ReceiveData(SPI5); // Dummy read

    // Read register value
    SPI_I2S_SendData(SPI5, 0xFF); // Dummy write
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_TXE) == RESET);
    while (SPI_I2S_GetFlagStatus(SPI5, SPI_I2S_FLAG_RXNE) == RESET);
    reg_value =  SPI_I2S_ReceiveData(SPI5); // register value

    // Device CS goes to high to stop transmission
    GPIO_SetBits(GPIOC, GPIO_Pin_1);

    return reg_value;
}
```

6.6 Communiquer via I2C

La carte de démonstration STM32F4-disco embarque un composant multifonction STMP811 connecté sur l'une des interfaces I2C du microcontrôleur via les pates PA8 et PC9. Ces dernières peuvent être connectées à l'interface I2C3 du microcontrôleur.

6.6.1 Initialisation I2C

```
void initI2C3(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    I2C_InitTypeDef I2C_InitStructure;

    /* Enable GPIOA, GPIOC and I2C3 clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C3, ENABLE);

    /* Configure GPIOA8, must be configured has opendrain for I2C */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure GPIOC9, same configuration */
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
    GPIO_Init(GPIOC, &GPIO_InitStructure);

    /* Set GPIO alternate function to I2C */
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource8, GPIO_AF_I2C3);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_I2C3);

    /* Set basic I2C configuration */
    I2C_StructInit(&I2C_InitStructure);
    I2C_InitStructure.I2C_Mode = I2C_Mode_I2C;
    I2C_InitStructure.I2C_ClockSpeed = 100000; // 100KHz
    I2C_InitStructure.I2C_OwnAddress1 = 0;     // no matter as we are I2C master
    I2C_InitStructure.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
    I2C_InitStructure.I2C_Ack = I2C_Ack_Disable;
    I2C_Init(I2C3, &I2C_InitStructure);

    I2C_Cmd(I2C3, ENABLE);
}
```

6.6.2 Les échanges I2C

Le code suivant consiste à initialiser le capteur de température interne au STMP811 et à récupérer la valeur des registres contenant la température du composant.

6.6.2.1 Fonctions génériques

Afin de ne pas dupliquer du code inutilement, il convient de faire des fonctions d'écriture/lecture des registres.

```
/*
 * Write I2C device register
 * dev_addr  (IN) : device address
 * reg_addr  (IN) : register address
 * reg_value (IN) : register value
 */
void write_reg(uint8_t dev_addr, uint8_t reg_addr, uint8_t reg_value)
{
    while(I2C_GetFlagStatus(I2C3, I2C_FLAG_BUSY));

    I2C_GenerateSTART(I2C3, ENABLE);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(I2C3, dev_addr, I2C_Direction_Transmitter);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
```

```
    I2C_SendData(I2C3, reg_addr);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    I2C_SendData(I2C3, reg_value);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    I2C_GenerateSTOP(I2C3, ENABLE);
}

/*
 * Read I2C device register
 * dev_addr    (IN)  : device address
 * reg_addr    (IN)  : register address
 * reg_value   (OUT) : output buffer for register value
 * buffer_size (IN)  : output buffer size
 */
void read_reg(uint8_t dev_addr, uint8_t reg_addr, uint8_t * buffer, uint8_t buffer_size)
{
    uint8_t rec_idx = 0;
    I2C_AcknowledgeConfig(I2C3, ENABLE);

    while(I2C_GetFlagStatus(I2C3, I2C_FLAG_BUSY));

    I2C_GenerateSTART(I2C3, ENABLE);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(I2C3, dev_addr, I2C_Direction_Transmitter);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

    I2C_SendData(I2C3, reg_addr);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_BYTE_TRANSMITTED));

    I2C_GenerateSTART(I2C3, ENABLE);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_MODE_SELECT));

    I2C_Send7bitAddress(I2C3, dev_addr, I2C_Direction_Receiver);
    while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

    for(rec_idx = 0; rec_idx < buffer_size; rec_idx++)
    {
        /* If last byte, do not ACK, no more data expected */
        if(rec_idx + 1 == buffer_size)
            I2C_AcknowledgeConfig(I2C3, DISABLE);
        while(!I2C_CheckEvent(I2C3, I2C_EVENT_MASTER_BYTE_RECEIVED));
        *(buffer + rec_idx) = I2C_ReceiveData(I2C3);
    }

    I2C_GenerateSTOP(I2C3, ENABLE);
}
```

6.6.2.2 Initialisation de la sonde de température et lecture de la température

Les adresses des registres et leurs significations sont disponibles dans le document « STMP811.pdf ».

```
void vTaskReadTemp(void * pvParameters)
{
    USART_Msg message;

    // Enable temperature and ADC
    write_reg(0x82, 0x04, 0x0C);

    // Enable temperature sensor and start continuous acquisition
    write_reg(0x82, 0x60, 0x07);

    for(;;)
    {
        uint8_t buffer[2];

        // Read MSB/LSB temperature register
        read_reg(0x82, 0x61, &buffer[0], 1);
        read_reg(0x82, 0x62, &buffer[1], 1);

        if(USARTQueue)
        {
            message.msg_type = USART_MSG_I2C_VALUE;
```

```
            // default for ADC conversion is 12bit
            message.temp = (uint16_t)((buffer[0] & 0x03) << 8 | buffer[1]);
            xQueueSend(USARTQueue, (void*)&message, portMAX_DELAY);
        }
        vTaskDelay(2000);
    }
}
```

6.7 Utiliser le convertisseur analogique/numérique avec trigger externe

Cette section contient un exemple d'utilisation d'un convertisseur analogique/numérique (ADC). La conversion sera déclenchée par le « timer 2 » et la lecture de la valeur convertie se fera dans l'interruption de fin de conversion générée par le ADC.

6.7.1 Initialisation de l'ADC

```
/*
 * This function configure ADC1
 */
void initADC(void)
{
    ADC_InitTypeDef ADC_InitStructure;
    ADC_CommonInitTypeDef ADC_CommonInitStructure;
    GPIO_InitTypeDef GPIO_InitStructure;
    NVIC_InitTypeDef NVIC_InitStructure;

    /* Enable GPIOA and ADC1 clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

    /* Configure GPIOA 1 to use analog mode */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Common configuration for all ADC */
    ADC_CommonStructInit(&ADC_CommonInitStructure);
    ADC_CommonInit(&ADC_CommonInitStructure);

    /* Configure ADC1 and select TIM2 has external trigger for conversion */
    ADC_StructInit(&ADC_InitStructure);
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T2_TRGO;
    ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Rising;
    ADC_Init(ADC1, &ADC_InitStructure);

    /* Add channel 1 to regular group */
    ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_480Cycles);

    /* Enable End Of Conversion interrupt on ADC1 */
    ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE);

    ADC_Cmd(ADC1, ENABLE);

    /* Add ADC1 IRQ to NVIC */
    NVIC_InitStructure.NVIC_IRQChannel = ADC_IRQn;
    NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStructure);

    // Start conversion
    ADC_SoftwareStartConv(ADC1);
}
```

6.7.2 Initialisation du timer

Afin que le « timer 2 » fasse office de trigger externe pour le début de conversion de l'ADC, il est nécessaire d'adapter sa configuration pour préciser son comportement lors de son réarmement.

```
TIM_SelectOutputTrigger(TIM2, TIM_TRGOSource_Update);
```

6.7.3 Lecture des valeurs du convertisseur

```
/*
 * This IRQHandler is called each time the ADC has finished his conversion.
 */
void ADC_IRQHandler(void)
{
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
    USART_Msg message;

    if(ADC_GetITStatus(ADC1, ADC_IT_EOC))
    {
        if(USARTQueue)
        {
            message.msg_type = USART_MSG_ADC_CONVERTION;
            message.adc = ADC_GetConversionValue(ADC1);

            xQueueSendFromISR(USARTQueue, (void*)&message, &xHigherPriorityTaskWoken);
        }
        ADC_ClearITPendingBit(ADC1, ADC_IT_EOC);
    }

    portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
}
```

6.8 Utiliser le convertisseur numérique/analogique

Le microcontrôleur embarque un DAC avec 2 canaux indépendants permettant la génération de tensions analogiques. L'exemple suivant n'utilise que le premier canal dont la sortie est disponible sur PA1.

```
void initDAC(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    DAC_InitTypeDef DAC_InitStructure;

    /* Enable GPIOA and DAC clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

    /* Configure GPIOA pin 1 in analgic mode */
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
    GPIO_Init(GPIOA, &GPIO_InitStructure);

    /* Configure DAC */
    DAC_InitStructure.DAC_Trigger = DAC_Trigger_None;
    DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
    DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    DAC_InitStructure.DAC_LFSRUnmask_TriangleAmplitude = DAC_LFSRUnmask_Bit0;
    DAC_Init(DAC_Channel_1, &DAC_InitStructure);

    DAC_Cmd(DAC_Channel_1, ENABLE);

    /* Set DAC output value */
    DAC_SetChannel1Data(DAC_Align_12b_R, 200);
}
```

6.9 Utiliser des interruptions externes

L'ensemble des entrées numériques du microcontrôleur sont susceptibles de devenir une source pour la génération d'une interruption. Pour illustrer cette fonctionnalité, il est possible d'utiliser le

bouton utilisateur de la carte de démonstration. Ce dernier est connecté au port GPIO A, pin 0. Le code suivant va donc configurer l'EXTI pour que le passage de cette entrée à l'état haut génère une interruption.

```
void initEXTIButton( void )
{
    GPIO_InitTypeDef GPIO_InitStruct;
    EXTI_InitTypeDef EXTI_InitStruct;
    NVIC_InitTypeDef NVIC_InitStruct;

    /* Enable GPIOA and SYSCFG Clock */
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

    /* Configure GPIOA 0 as input because will be used as external interrupt */
    GPIO_InitStruct.GPIO_Mode = GPIO_Mode_IN;
    GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0;
    GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_DOWN; /* when button is pressed, GPIO0 is connected to Vcc so
pulldown */
    GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOA, &GPIO_InitStruct);

    /* Select GPIOA 0 for EXTILine 0 */
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

    /* Configure EXTI_Line0 */
    EXTI_InitStruct.EXTI_Line = EXTI_Line0;
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;
    EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
    EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_Init(&EXTI_InitStruct);

    /* Add EXTI0 IRQ vector to NVIC */
    NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
    NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0;
    NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
    NVIC_Init(&NVIC_InitStruct);
}
```

6.9.1 Implémentation de l'ISR EXTI0

```
/*
 * This IRQHandler is called when user button is pressed.
 */
void EXTI0_IRQHandler(void){

    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;
    USART_Msg aMessage;

    /* Make sure that interrupt flag is set */
    if (EXTI_GetITStatus(EXTI_Line0) != RESET)
    {
        if(USARTQueue)
        {
            aMessage.msg_type = USART_MSG_BUTTON_PUSHED;
            xQueueSendFromISR(USARTQueue, (void*)&aMessage, &xHigherPriorityTaskWoken);
        }
        /* Clear interrupt flag */
        EXTI_ClearITPendingBit(EXTI_Line0);
    }

    portEND_SWITCHING_ISR(xHigherPriorityTaskWoken);
}
```