Guide utilisateur

Installation et utilisation de Riot

Auteurs : OBEISSART Morgan – ROBIC Vincent

Année scolaire : 2016 – 2017

Sommaire

Introduction

Nous allons tenter ici d'expliquer, de la manière la plus simple possible, comment utiliser le système d'exploitation Riot. Riot est un logiciel libre publié sous Licence publique générale limitée GNU (LGPL). Le but de ce guide est de faire gagner du temps à l'utilisateur qui souhaite développer avec Riot. En effet, cet OS est en cours de développement, et étant assez récent, la communauté n'est pas encore très grande et donc les supports ne sont pas nombreux.

Nous débuterons ce guide avec l'installation de Riot et de tous les prérequis, c'est-à-dire l'ensemble des outils logiciels et matériels nécessaires pour suivre entièrement ce guide. Puis, nous indiquerons comment créer son premier projet Riot et comment le compiler sous Linux. Enfin, nous nous intéresserons à un programme de test particulier : celui du pilote de la puce AT86RF231 qui permet de faire de la communication radio.

I. Installation de Riot

1. Matériel utilisé

Dans tout ce guide, la plateforme sur laquelle nous souhaitons travailler est la STM32F4discovery développée par STMicroelectronics. Notre application finale (qui ne sera pas abordé dans ce guide) consiste à mettre en place un réseau dont les nœuds communiqueraient par liaison radio. Nous allons donc en profiter pour expliquer comment compiler Riot sur la carte STM32F4discovery en incluant le pilote de l'AT86RF231 (puce RF que nous utilisons pour la communication radio) qui est déjà développé par Riot.

2. Prérequis et installation de Riot

2.1. Prérequis

Pour pouvoir compiler Riot sur une une carte STM32F4discovery, il faut tout d'abord installer la suite logicielle de compilation correspondante **gcc-arm-none-eabi – A GNU-based toolchain for arm embedded** processors ainsi que le débogueur associé. Pour cela, sous Linux, il suffit d'entrer les lignes de commandes suivantes :

```
sudo add-apt-repository ppa:terry.guo/gcc-arm-embedded
sudo apt-get update
sudo apt-get install gcc-arm-none-eabi gdb-arm-none-eabi
```

Ensuite, pour pouvoir flasher le programme et contrôler la carte en cas de débogage, Riot utilise **openocd**, il nous faut donc l'installer :

```
sudo apt-get install openocd
```

2.2. Installation de Riot

Tous les outils sont désormais installés pour pouvoir compiler le programme et le flasher sur notre carte de développement. Passons maintenant à l'installation de Riot. Il est possible d'obtenir la dernière version de Riot depuis le répertoire git suivant :

```
git clone https://github.com/RIOT-OS/RIOT.git
```

C'est la seule étape nécessaire pour installer Riot. Avant de passer à la création de notre premier projet, nous allons installer l'outil pyterm. Il s'agit d'un terminal permettant d'interagir avec les microcontrôleurs via les pins d'UART.

2.3. Installation et configuration de pyterm

Pour pouvoir utiliser ce terminal, il faut installer le paquet python 2.7. Pour cela :

```
apt-get install python2.7
```

Ensuite, il suffit de lancer les commandes suivantes en se plaçant dans le répertoire **$RIOT_PATH/dist/tools/pyterm** afin d'installer tous les autres paquets nécessaires au fonctionnement du terminal **($RIOT_PATH** désigne la racine de votre répertoire RIOT) :

```
./setup.py build
./setup.py install
```

Nous verrons dans la partie suivante comment lancer ce terminal.

II. Création et compilation de notre premier projet

Nous rappelons que nous travaillons ici avec la carte STM32F4discovery. Le premier programme qu'il serait intéressant de mettre en place est celui qui permet d'allumer une des 4 LEDs que contient notre carte.

Pour créer un projet, la manière la plus rapide et la plus simple est de partir d'un projet existant. En effet, Riot propose plusieurs exemples, que l'on trouvera dans le répertoire **$RIOT_PATH/examples/**. Copions donc l'un de ces exemples (nous prendrons l'exemple « default ») pour créer notre projet, que nous appelons « test_led » :

```
cp –r default/ test_led/
```

Notre projet se compose alors d'un Makefile, d'un fichier README, et d'un fichier main.c qui contient le code du programme. Le programme que nous avons récupéré permet de lancer un terminal RIOT, ce qui sera très utile dans notre prochain exemple, mais pas ici. Riot fournit un support pour la carte STM32F4discovery : autrement dit, des fichiers de configurations (qui permettent de connaître les pins des LEDs, des PWM, etc.) sont déjà fournis. Nous les trouvons dans le répertoire **$RIOT_PATH/boards/stm32f4discovery/include/**. Il faut donc inclure ces fichiers dans notre programme pour pouvoir utiliser la configuration fournie.

Enfin, pour pouvoir allumer une LED, il faut définir la pin correspondante en sortie, puis mettre cette pin au niveau HIGH. Riot propose des fonctions qui permettent de faire cela facilement. Ces fonctions se trouvent dans le répertoire **$RIOT_PATH/drivers/include/periph**. Dans ce répertoire se trouvent plusieurs fichiers .h qui fournissent les signatures de différentes fonctions (relatives aux timers, aux signaux PWM, etc) dont celles de gestion des GPIOs.

Finalement, nous proposons ici le code qui permet simplement d'allumer la LED 1 :

```
#include <stdio.h>
#include <string.h>
#include "periph/gpio.h"
#include "../../boards/stm32f4discovery/include/board.h"
#include "../../boards/stm32f4discovery/include/periph_conf.h"

int main(void) {
      gpio_init(LED1_PIN,GPIO_OUT);
      gpio_set(LED1_PIN);
      return 0;
}
```

Il nous reste à compiler ce programme et à le flasher sur notre carte. Pour cela, il faut d'abord légèrement modifier le Makefile et remplacer « APPLICATION = default » par « APPLICATION = test_led ». Puis, nous pouvons compiler et flasher en une ligne de commande :

```
BOARD=stm32f4discovery make all flash
```

Nous avons vu ici un exemple très simple qui nous a permis de prendre en main le système d'exploitation avec notre carte STM32F4discovery. Nous allons maintenant nous intéresser au programme de test de l'AT86RF231.

III. Test de l'AT86RF231

Pendant notre projet, nous avons réalisé une communication radio entre plusieurs cartes STM32F4discovery. Il a donc fallu concevoir une module radio. Nous avons développé ce module à partir de la puce AT86RF231, dont le pilote est fourni par Riot. Avant d'utiliser le module, nous avons voulu le tester et pour cela, Riot propose un programme de test qui permet, notamment, de pouvoir émettre et recevoir des messages.

La communication entre la STM32 et le module radio se fait par SPI. Il faut donc d'abord définir les pins permettant de réaliser cette communication. Pour cela, il faut modifier le fichier **$RIOT_PATH/drivers/at86rf2xx/include/at86rf2xx_params.h** :

```
#ifndef AT86RF2XX_PARAM_SPI
#define AT86RF2XX_PARAM_SPI  (SPI_0)  // correspond au SP1 de la STM32F4 avec :
                                         SCK<->PA5 / MISO<->PA6 / MOSI<->PA7
#endif
#ifndef AT86RF2XX_PARAM_SPI_SPEED
#define AT86RF2XX_PARAM_SPI_SPEED   (SPI_SPEED_5MHZ) // comme indiqué dans la
                                                        datasheet ne pas dépasser
                                                        8MHz
#endif
// Choix arbitraire des pins correspondant à des GPIOs
#ifndef AT86RF2XX_PARAM_CS
#define AT86RF2XX_PARAM_CS          (GPIO_PIN(PORT_D, 6))
#endif
#ifndef AT86RF2XX_PARAM_INT
#define AT86RF2XX_PARAM_INT         (GPIO_PIN(PORT_E, 2))
#endif
#ifndef AT86RF2XX_PARAM_SLEEP
#define AT86RF2XX_PARAM_SLEEP       (GPIO_PIN(PORT_D, 7))
#endif
#ifndef AT86RF2XX_PARAM_RESET
#define AT86RF2XX_PARAM_RESET       (GPIO_PIN(PORT_D, 10))
#endif
```

Déplaçons-nous dans le répertoire **$RIOT_PATH/tests/driver_at86rf2xx/** qui contient le programme de test de la puce et intéressons-nous d'abord au Makefile.

Riot est un système d'exploitation modulaire, c'est-à-dire qu'il est possible d'ajouter et/ou d'enlever des modules dans une application.

Dans ce Makefile, nous pouvons notamment voir :

```
USEMODULE += shell

USEMODULE += shell_commands

USEMODULE += at86rf231
```

Ces lignes permettent d'inclure les modules nécessaires au lancement du terminal et de choisir l'at86rf231 comme driver. Pour compiler, flasher et lancer le terminal pyterm, il faut lancer la commande suivante :

```
BOARD=stm32f4discovery make all flash term
```

Par défaut, pyterm lance le terminal sur le port série /dev/ttyUSB0. Si votre FTDI est branché sur un autre port, il est possible de modifier ce paramètre. Si on souhaite lancer le terminal sans reflasher la carte, on peut utiliser la commande :

```
$RIOT_PATH/dist/tools/pyterm/pyterm –p /dev/ttyUSB0 –b 115200
```

Une fois le terminal lancé, il est possible de voir les commandes disponibles et la façon de les utiliser en tapant la commande « help ». Nous pouvons par exemple utiliser la commande « txtsnd <interface> <addresse> <data> » pour envoyer des données à une certaine adresse.

Conclusion

Dans ce guide, nous avons vu comment installer Riot et tous les outils nécessaires à son fonctionnement (chaîne de compilation, terminal pyterm, etc). Nous avons également pu prendre en main cet OS à travers un exemple simple (allumer une LED) et un test déjà proposé par Riot.

Riot propose beaucoup de fonctionnalités, et ce guide permet de savoir où chercher les informations afin de gagner du temps dans la compréhension du fonctionnement de ce système d'exploitation. Cela pourra s'avérer utile pour toute personne qui souhaite réaliser un projet avec Riot, sachant que la communauté autour de celui-ci n'est pas encore très grande.