1 T'es mignon mais t'es un tout p'tit bouton [1]

Nous allons commencer par quelque chose de relativement simple au premier abord et dans le même ordre d'idées que le support des leds LD3 et LD4 de la carte. Vous l'avez sans doute remarqué, le devkit dispose de deux boutons, l'un marqué RESET et l'autre, bleu, libellé USER. Le schéma de la carte (`MB1075.pdf`) nous montre que le RESET est connecté comme il se doit sur la broche NRST du microcontrôleur. Le bouton bleu, quant à lui, est lié à la ligne PA0 (port A ligne 0) et est équipé de tous les composants nécessaires pour gérer le rebond (résistant de rappel, condensateur et résistance en réseau RC, etc). Deux ponts de soudures, SB1 et SB2 permettent éventuellement de déconnecter les deux boutons au fer à souder.

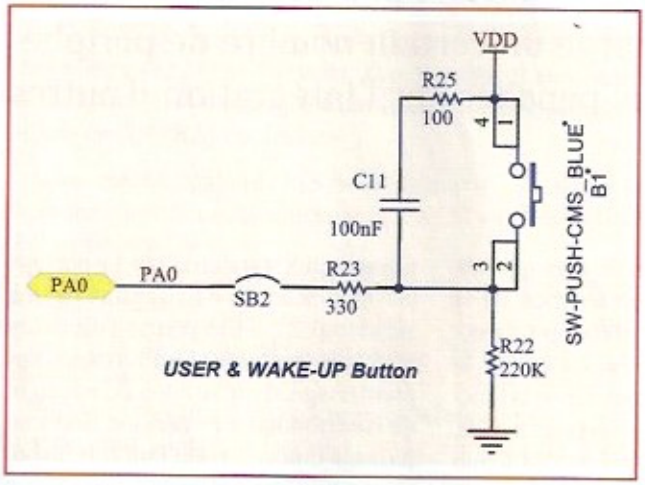

Schéma de connexion du bouton USER de la carte avec sa connexion à PA0.

L'objectif visé par notre développement consiste à générer une interruption lors d'une pression sur le bouton bleu et, par la même occasion un petit message dans les logs du noyau. Ceci pourra servir de base pour un module plus évolué permettant une interaction avec un utilisateur par exemple mais surtout nous offre l'occasion de nous familiariser avec la notion d'interruption sous Linux.

Le microcontrôleur STM32, comme l'ensemble des composants basés sur un coeur ARM Cortex-M, dispose d'un contrôleur d'interruption très évolué : le NVIC pour *Nested Vectored Interrupt Controller*. Celui-ci gère les priorités entre interruptions et la gestion des registres (sauvegarde et restauration) de manière optimisée et ce de la même manière quelque soit le coeur ARM utilisé (ceci facilite le portage). Le fondeur, tel STMicroelectronics, greffe sur le NVIC ses interruptions en fonction des périphériques ajoutés dans le SoC. Un module supplémentaire nommé EXTI (*EXTernal Interrupts*) permet de lier un événement extérieur (bouton, ligne /CS en *SPI slave*, etc) à une interruption. Via une configuration de l'EXTI et du NVIC, il est ainsi possible de définir une interruption sur n'importe quel GPIO.

Le contrôleur EXTI est connecté au NVIC et peut être configuré pour être associé à 16 des 168 GPIO disponibles avec le STM32F429ZIT6. L'EXTI utilise donc 7 lignes de base pour « remonter » les changements d'états :

- EXTI0 : connectable aux lignes 0 des ports A à K,
- EXTI1 : connectable aux lignes 1 des ports A à K,
- EXTI2 : connectable aux lignes 2 des ports A à K,
- EXTI3 : connectable aux lignes 3 des ports A à K,
- EXTI4 : connectable aux lignes 4 des ports A à K,
- EXTI5-9 : connectable aux lignes 5 à 9 des ports A à K,
- EXTI10-15 : connectable aux lignes 10 à 16 des ports A à K,

Notez que, par exemple, pour EXTI0 les broches 0 des ports A à K sont gérés. La valeur inscrite dans le registre correspondant (`SYSCFG_EXTICR1` bits `0:3`) détermine le port utilisé :

- `0000` : broche PA0
- `0001` : broche PB0
- `0010` : broche PC0
- `0011` : broche PD0
- `0100` : broche PE0
- `0101` : broche PF0
- `0110` : broche PG0
- `0111` : broche PH0
- `1000` : broche PI0
- `1001` : broche PJ0
- `1010` : broche PK0

Ainsi, en oubliant de choisir un port pour une ligne donnée et en activant une interruption sur la ligne EXTI en question, vous utiliserez systématiquement le port A, sachant que celui-ci déjà passablement utilisé dans le cas du devkit (PA4 = LCD-TFT_VSYNC par exemple). Ce mécanisme impacte également la répartition des interruptions. La valeur des bits en question est en BCD (*Binary Coded Decimal*), il est donc impossible par exemple d'avoir à la fois PA0 et PB0 générant une interruption.

Sept autres lignes particulières sont connectées à l'EXTI pour la gestion, par exemple des interruptions liés à la RTC, l'USB ou encore l'Ethernet *wakeup*. Ceci nous donne un total de 23 sources/lignes d'interruptions disposant d'une détection de front configurable. Ce qui abouti à la possibilité d'utiliser 7 vecteurs d'interruptions :

[1] Désolé :-/