Faculté des Sciences & Techniques — Université de Limoges

Master 2nde année

TMC

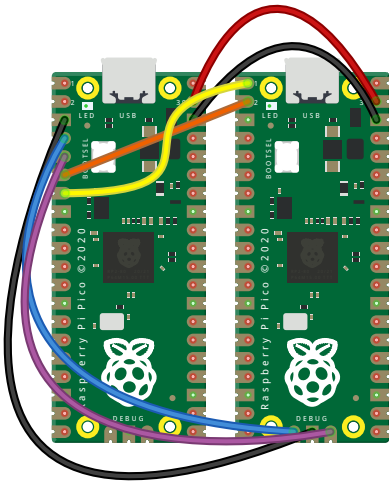
Pico Hardware

Exécution de code en SRAM – Utilisation d'une sonde hardware – Obfuscation de code

Déboguage matériel : connexion Pico probe ⟺Pico

On va interconnecter deux Raspberry Picos :

- ▷ le premier Pico servira de « *probe* », **sonde**, de déboguage matériel *celle à gauche sur le schéma* :
  - ◇ il utilisera un **firmware spécial** fourni par Raspberry disponible à :
    `https://github.com/raspberrypi/debugprobe/releases/download/debugprobe-v2.3.1/debugprobe_on_pico.elf`
  - ◇ il sera **connecté en USB** à votre ordinateur.
- ▷ le second Pico sera connecté au premier :
  - ◇ pour son alimentation ;
  - ◇ pour son déboguage ;
  - ◇ pour son port série : ses communications séries seront relayées par le premier vers l'ordinateur.



Pour flasher le firmware sur la pico-probe **avant de connecter l'autre PICO** :

xterm
```
$ openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg -c "program
debugprobe_on_pico.elf verify reset exit"
```

Exéction de code en SRAM

Vous trouverez le source du projet à :

`https://git.p-fb.net/PeFClic/tmc_sram_execution.git`

Vous exécuterez le code sur :

- ▷ renode :

xterm
```
$ docker run -it --rm --name renode -p 8080:8080 -p 3333:3333 -v "$(pwd)":/BUILD
renode-rp2040 renode /BUILD/launch_in_renode.resc
```

- ▷ sur le Pico, en le flashant dans sa mémoire :

xterm
```
$ openocd -f interface/cmsis-dap.cfg -f target/rp2040.cfg -c "program
blink_flash.elf verify reset exit"
```

Questions :

1. Décrivez ce que fait le projet.
2. À quoi sert la ligne `while ((GET32(0x4000c008) & (1 << 5)) == 0);` ?
3. À quoi sert l'option `-fpic` ?
4. Dans `boot2.s`, pourquoi utiliser 4Ko ?
5. À quelle vitesse la LED clignote ?

Obfuscation de code

Ci-dessus le code à modifier dans `boot2.s` :

```
ldr r7, =0xA5A5A5A5
_copyToRam:
ldmia r4!, {r0-r3}
eor r0, r0, r7
eor r1, r1, r7
eor r2, r2, r7
eor r3, r3, r7
stmia r5!, {r0-r3}
sub   r6, #16
bne   _copyToRam
```

Questions :

1. Que fait-il ?
2. Écrivez un programme Python réalisant le travail nécessaire au bon fonctionnement du firmware sur le Pico.
3. Est-ce une bonne protection ? Pourquoi ?

Utilisation d'interruption

Vous trouverez le source du projet à :
`https://git.p-fb.net/PeFClic/tmc_sram_isr.git`

Questions :

a. Que fait le projet ?

b. Qu'est-ce que VTOR ?

c. Comment évoluent les fichiers du linker ?

d. Où se trouve le code à exécuter ?

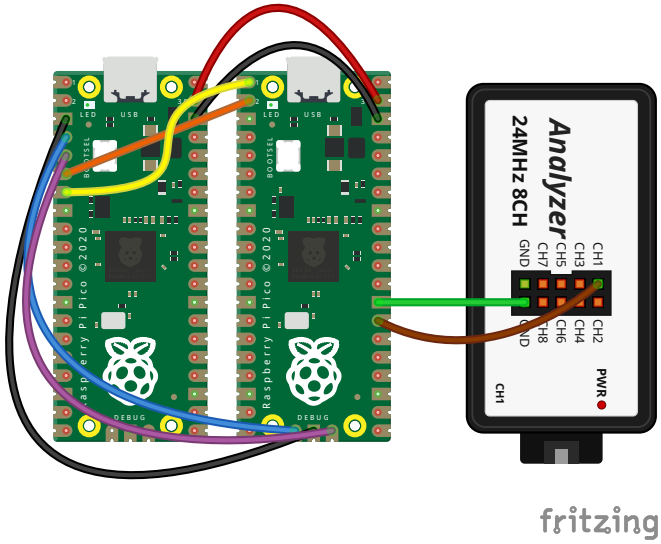
e. Comment fonctionne l'interruption ? Comment est-elle configurée ?

f. Est-ce que cela fonctionne sur le Pico ou sur Renode ?

Étude du signal d'horloge : le ROSC *vs* le XOSC

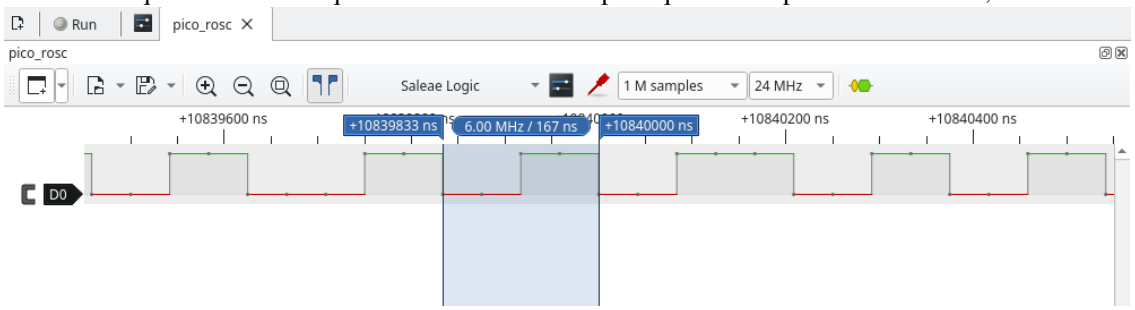
Le source du projet est à :
`https://git.p-fb.net/PeFClic/tmc_pico_clock_out.git`

Vous récupérerez l'outil Pulseview à :
`https://sigrok.org/download/binary/pulseview/PulseView-0.4.2-x86_64.AppImage`

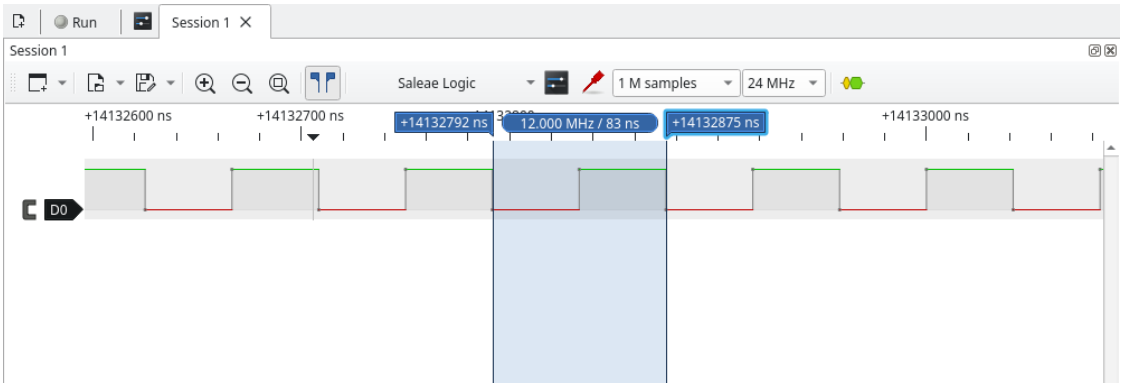


Le Raspberry Pico a la capacité de sortir le signal d'horloge sur quelques broches, dont la GPIO21.
En particulier, on va sortir le signal d'horloge fourni par :

- le ROSC, *Ring Oscillator* : circuit composé d'une succession d'inverseurs dont la sortie finale et reliée à l'entrée.
  Il a une fréquence de 6MHz qui est suffisament stable pour que le Pico puisse fonctionner ;



  C'est l'horloge qui est utilisée au démarrage du Pico.
- le XOSC, *Crystal Oscillator* : le composant d'horloge intégré sur le PCB du Pico :



  C'est l'horloge stable qui peut être utilisée pour cadencer le Pico, mais aussi en entrée des PLL :
  - *PLL_USB : l'USB exige une horloge de 48MHz ;*
  - *PLL_SYS : permet de cadencer le Pico jusqu'à 125MHz ou 133MHz.*

Questions :

1. Vous testerez chaque horloge en copiant alternativement les fichiers `clock_out_12MHZ.c` et `rosc_out.c` en `clock_out.c`.
2. Que fait l'option « flash » du Makefile ?

Remarques :
On remarquera que la consommation est différente en fonction de la fréquence :

```
----------+----------------------
  Clock   |   active    idle  (mA)
----------+----------------------
  125MHz  |   27.119   22.525
   60MHz  |   15.236   12.919
   48MHz  |   12.532   10.980
   24MHz  |    8.005    7.055
----------+----------------------
```

Comment sont générées les fréquences :
$Fout = (Fref/REFDIV) \times FBDIV/(POSTDIV1 \times POSTDIV2)$

- ▷ Fref :reference input (12 MHz crystal by default)
- ▷ REFDIV : input divider (1–63); typically set to 1
- ▷ FBDIV : integer feedback multiplier (16–320)
- ▷ POSTDIV1, POSTDIV2 : output dividers (1–7 each), with the constraint POSTDIV1 ≥ POSTDIV2

Par exemple :

PLL	REF	REFDIV	FBDIV	VCO	POSTDIV1	POSTDIV2	Output
pll_sys	12 MHz	1	125	1500 MHz	6	2	125 MHz
pll_usb	12 MHz	1	100	1200 MHz	5	5	48 MHz

Chaque PLL est mappée vers un ensemble de registres :
(base addresses `0x40028000` pour pll_sys, `0x4002C000` pour pll_usb):

Register	Offset	Key Fields
PLL_CS	0x00	LOCK (bit 31; read-only); BYPASS (bit 8); REFDIV (bits 5)
PLL_PWR	0x04	VCOPD; POSTDIVPD; DSMPD; PD — individual power-down bits
PLL_FBDIV_INT	0x08	FBDIV_INT (bits 11) — the integer feedback divisor
PLL_PRIM	0x0C	POSTDIV1 (bits 18); POSTDIV2 (bits 14)

Questions :

1. Utiliser une IA pour créer un code permettant de modifier la PLL tout en sortant le signal sur la GPIO21.
2. Modifier la fréquence du PLL et observer cette fréquence sur l'analyseur logique.