Faculté des Sciences & Techniques — Université de Limoges

Master 2nde année

TMC

Pico MPU

Utilisation de la MPU, « Memory Protection Unit *»*

Vous installerez une nouvelle version des outils gdb disponible à
`https://git.p-fb.net/PeFClic/outils_gdb_arm.git`

Déploiement du projet

Vous trouverez le contenu du projet MPU sur : `https://git.p-fb.net/PeFClic/TMC_pico_mpu.git`
Dans le projet, vous aurez un script pour préparer automatiquement `renode` :
`launch_in_renode.resc`

```
include @boards/initialize_raspberry_pico.resc
sysbus LoadELF @/BUILD/build/mpu_demo.elf
cpu0 PC 0x100001E9
cpu1 PC 0x100001E9
machine StartGdbServer 3333
showAnalyzer sysbus.uart0
```

Pour fabriquer le firmware :

xterm
```
$ export PICO_SDK_PATH=~/pico-sdk
$ cmake -B build .
$ cd build
$ make
$ cd ..
```

Vous lancerez la simulation par :

xterm
```
$ docker run -it --rm --name renode -p 8080:8080 -p 3333:3333 -v $(pwd):/BUILD renode-rp2040
21:12:12.0514 [WARNING] Couldn't start UI - falling back to console mode
21:12:12.3183 [INFO] Loaded monitor commands from: /opt/renode/scripts/monitor.py
Renode, version 1.16.1 (d66b0c2a-202602160923)

(monitor) include @/BUILD/launch_in_renode.resc
```

Le `include` lance le script à exécuter par renode : il charge le firmware et configure son exécution.

Ensuite vous pourrez vous connecter avec `gdb` :

xterm
```
$ gdb-multiarch -ex "target extended-remote :3333" build/mpu_demo.elf
```

Dans gdb, en se servant des outils proposés par Claude :

xterm
```
>>> bits-arch v6m
>>> bits-svd load ~/RP2040.svd
>>> bits-svd list
>>> bits-svd list PPB
>>> bits-svd show PPB.MPU_CTRL
>>> bits -r PPB.MPU_CTRL
>>> bits -r PPB.MPU_TYPE
>>> bits -m 0xE000ED90
```

Analyse du fonctionnement de la protection fournie par la MPU

1 – Vous décrirez le fonctionnement du firmware et vous vous arrêterez sur l'activation de la MPU :

xterm
```
>>> b mpu_setup
```

Suivez l'exécution jusqu'aux appels à la fonction `mpu_region`.

xterm
```
>>> source mpu.gdb
>>> mpu_ctrl
>>> mpu_regions
```

Vous vérifierez pour chaque appel de le fonction qu'une nouvelle région est définie et ajoutée.

Vous réaliserez manuellement l'analyse de la région 3 :

xterm
```
>>> bits -r PPB.MPU_RNR
>>> bits -m 0xE000ED9C
>>> bits -r PPB.MPU_RASR
>>> set *0xE000ED98=3
>>> bits-svd show PPB.MPU_RASR
```

Une fois la MPU activée, vérifiez avec les commandes `mpu_ctrl` et `mpu_regions` les régions et leurs droits d'accès.

2 – Vous analyserez **avant** la faute :

xterm
```
>>> b main.c:246
>>> continue
>>> p ro_table[1]
>>> bits $xpsr
>>> bits $control
>>> disassemble /s $pc,+12
>>> x/2i xn_code
```

Puis **après** la faute :

xterm
```
>>> break isr_hardfault
>>> continue
>>> bits $xpsr
>>> bits $control
>>> p ro_table[1]
>>> disassemble /s $pc,+12
```

Et dans la partie C du handler :

xterm
```
>>> break hardfault_c
>>> c
>>> exc_frame
```

À quoi correspond la valeur de `lr` ?

3 – Mode Privilégié :

xterm
```
>>> b main.c:263
```

Malheureusement, la valeur du registre Control est incorrect dans gdb. Il faut utiliser `cpu0 Control` dans renode.

Analysez ce qui se passe et pourquoi une faute se déclenche.