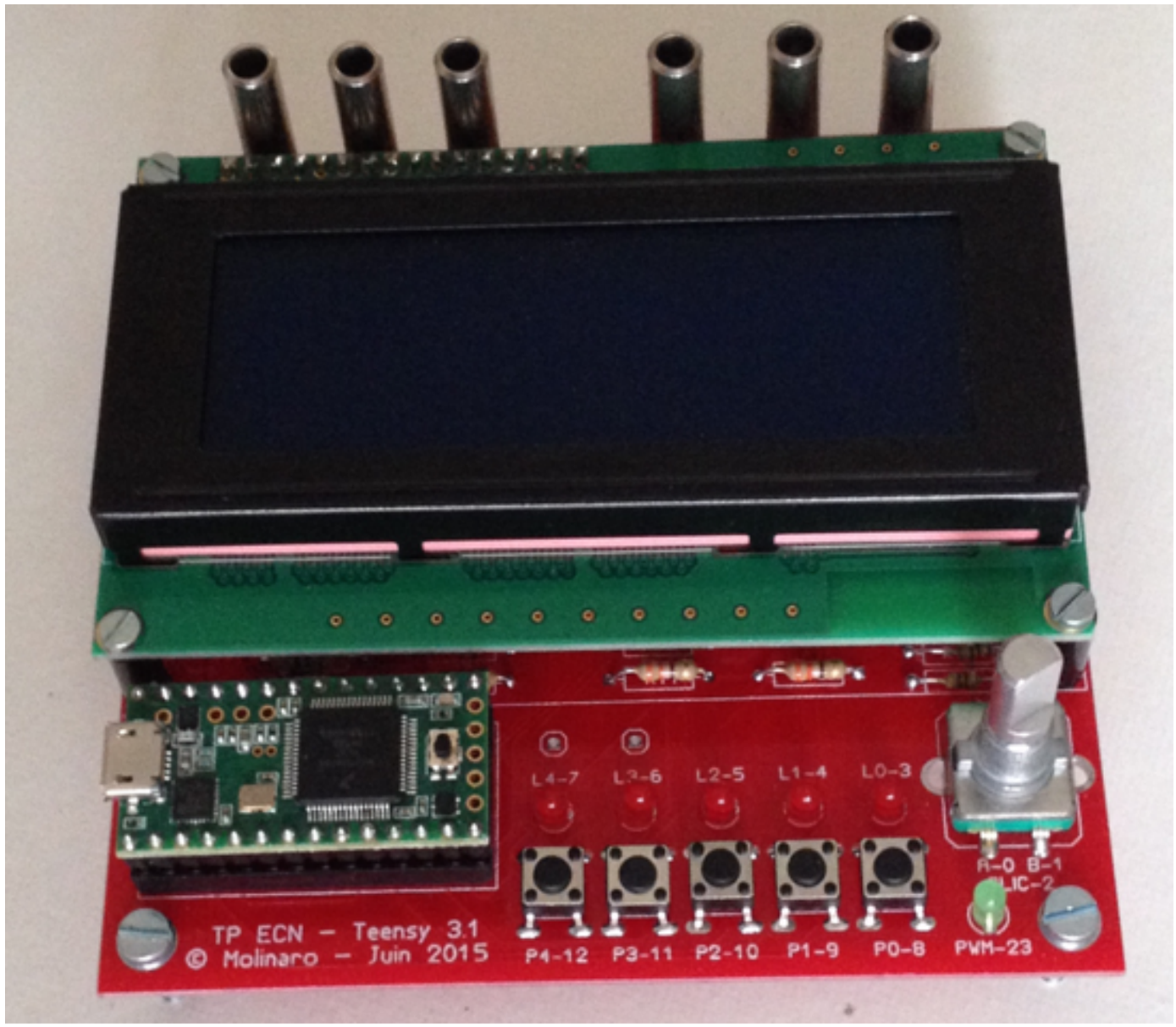
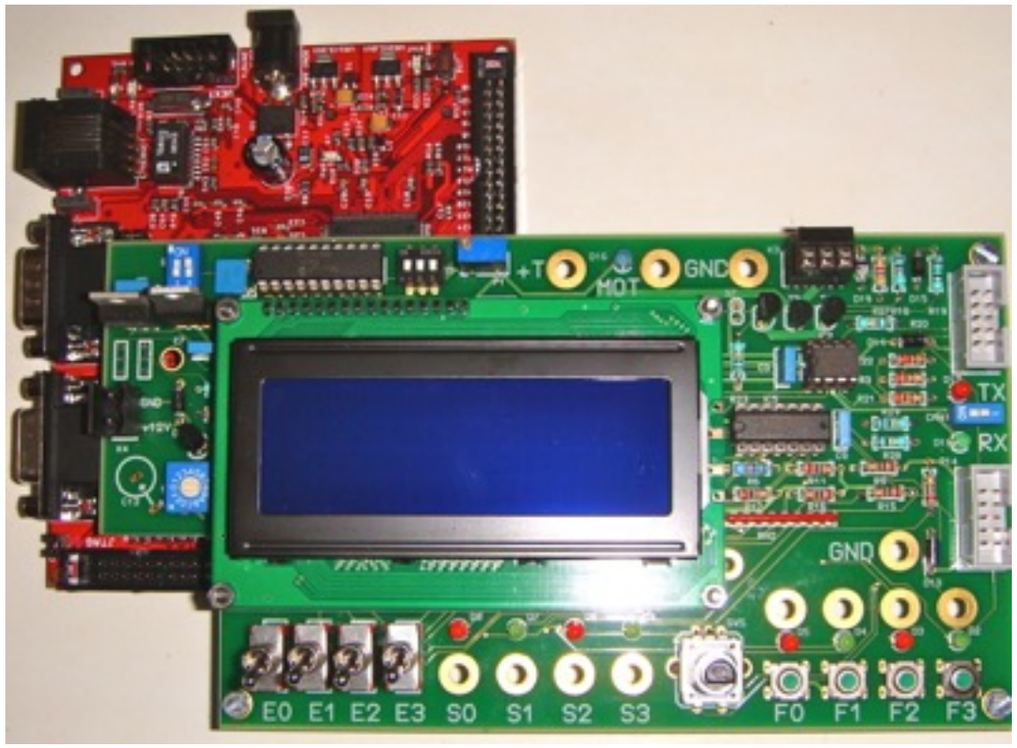
Temps Réel


L4-7
L3-6
L2-5
L1-4
L0-3
R-0 B-1
LIC-2
TP ECN - Teensy 3.1
© Molinaro - Juin 2015
P4-12
P3-11
P2-10
P1-9
P0-8
PWM-23

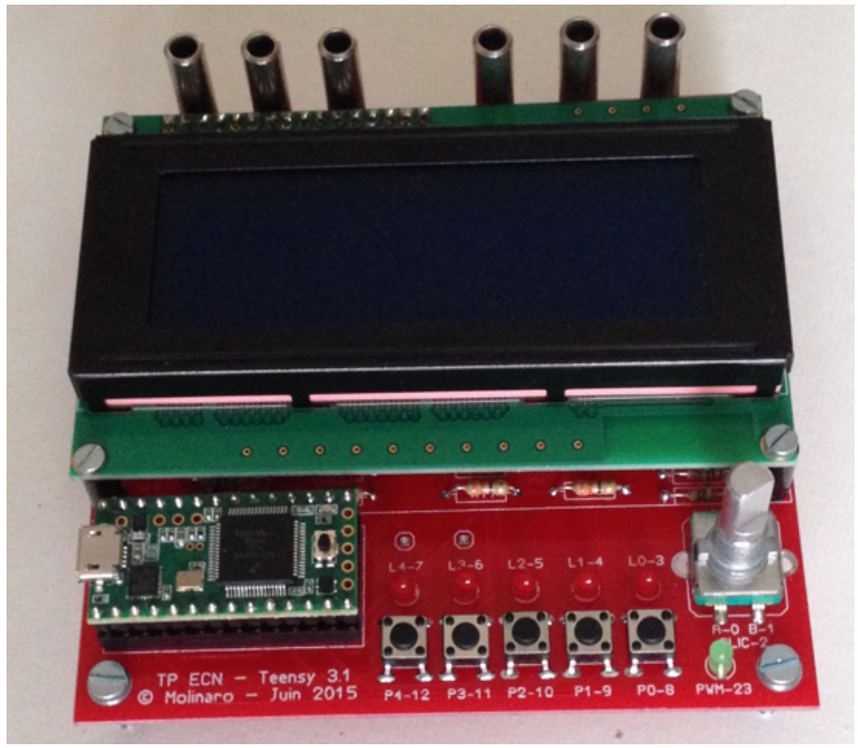
Note préalable

Le matériel de développement a été changé pour l'année 2015/2016.

Années précédentes



À partir de 2015/2016



Objectifs de l'enseignement TReel

Objectifs :

- *comprendre le fonctionnement d'un exécutif Temps Réel, en réalisant son écriture pas à pas ;*
- *comprendre le développement sur un micro-contrôleur.*

Moyens :

- mise à disposition de cartes micro-contrôleurs ;

Organisation :

- cours / TD / TP ;

Étapes de réalisation (prévisionnel)

10 séances de 3 h :

1. présentation / introduction : programme « blink led » ;
2. infrastructure pour séquentiel ;
3. SysTick ;
4. leds utilisateur ;
5. boutons poussoir ;
6. afficheur LCD ;
7. piège des interruptions ;
8. appel SVC ;
9. MSP et PSP ;
10. modes logiciels ;
11. structure exécutif ;
12. lancement tâche unique ;
13. préemption ;
14. fin des tâches ;
15. attente délai ;
16. sémaphore de Dijkstra ;
17. attente échéance ;
18. buffer affichage LCD ;
19. primitive Puntil ;

Évaluation :

- devoir surveillé.

I — Installation de la chaîne de développement

Installation

Plateforme de développement :

- Linux 32 bits ;
- Linux 64 bits ;
- Mac OS X, processeur Intel ;
- Windows : installer une machine virtuelle Linux.

Pré-requis :

- Linux : installer `gcc`, `g++`, `make`, `libftdi-dev` ;
- Mac : installer *Xcode* ;

Archive :

1. récupérer l'archive `info-treel.tbz` sur le serveur pédagogique ;
2. décompressez-la ;
3. placer le répertoire obtenu dans le répertoire de votre choix, du moment que son chemin ne comporte ni espace ni caractère accentué ;
4. lancer `install/install-all.py` ; ce script va télécharger le système de compilation propre à votre plateforme ; sur Linux, le script demande en plus le mot de passe administrateur pour installer l'accès à l'USB (`udev`).

Tout est installé dans le répertoire `~/dev-arm`. Pour désinstaller, exécuter le script `install/uninstall-all.py`.

Exemple d'installation sur Mac

```
+ cd /Volumes/dev-svn/dev-lpc2294/install
+ mkdir -p /Users/pierremolinaro/dev-arm
URL: http://crossgcc.rts-software.org/downloads/dev-arm/dev-arm-Intel-Darwin-
gccarm-5_2-2015q4-20151219-openocd-0.8.0.tar.bz2
Downloading...
 100.0% of 75MiB
+ cp dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0.tar.bz2 /Users/pierremolinaro/
dev-arm/dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0.tar.bz2
+ cd /Users/pierremolinaro/dev-arm
+ bunzip2 dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0.tar.bz2
+ tar xf dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0.tar
+ rm dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0.tar
---------- Success!
```

2 — Compilation et exécution du 1er programme

Présentation

Le premier programme « `01-blinkled` » est un programme minimum pour débuter en faisant clignoter la led Teensy.

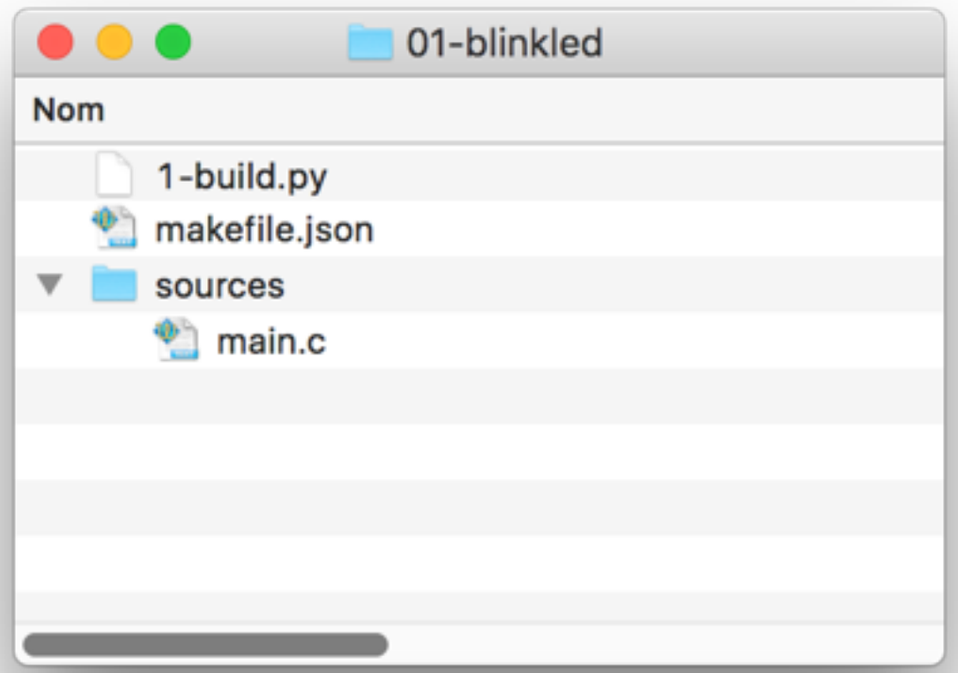
Ce n'est pas une base solide pour développer un programme sous interruption ; le prochain programme « `02-infrastructure-pour-sequentiel` » établira cette base.

Dans cette section, on présente comment compiler et exécuter le premier programme « `01-blinkled` » : ces explications seront valides pour tous les programmes suivants que vous écrirez.

L'explication du fonctionnement du premier programme fait l'objet des sections suivantes de ce document.

Ce que contient le répertoire d'un programme

Ouvrez le répertoire `etapes-realisation-executif-cours-treel/01-blink-led`.



`1-build.py`	c'est le fichier qui permet de lancer la compilation du projet (son utilisation est décrite dans les pages qui suivent).
`makefile.json`	Organisation de la compilation. Voir page suivante.
`sources`	Répertoire qui, comme son nom l'indique contient les sources du projet : dans le cas présent, le fichier main.c.

Fichier `makefile.json`

La compilation d'un programme est décrite dans un fichier `makefile.json` au format JSON.

Référence :

https://fr.wikipedia.org/wiki/JavaScript_Object_Notation

Pour le programme « `01-blinkled` », ce fichier est :

```
{
  "DEVELOPMENT_BOARD" : "teensy-3-1",
  "SOURCE_DIR" : ["sources"],
  "SOURCES" : ["main.c"]
}
```

`DEVELOPMENT_BOARD`	Caractérise la plateforme de développement. Ne pas changer.
`SOURCE_DIR`	Liste des répertoires sources. Ne pas changer, sauf si vous répartissez vos sources dans plusieurs répertoires.
`SOURCES`	Liste des fichiers sources. Dans les programmes suivants, vous ajouterez ici les noms des autres fichiers source.

Compiler le programme d'exemple

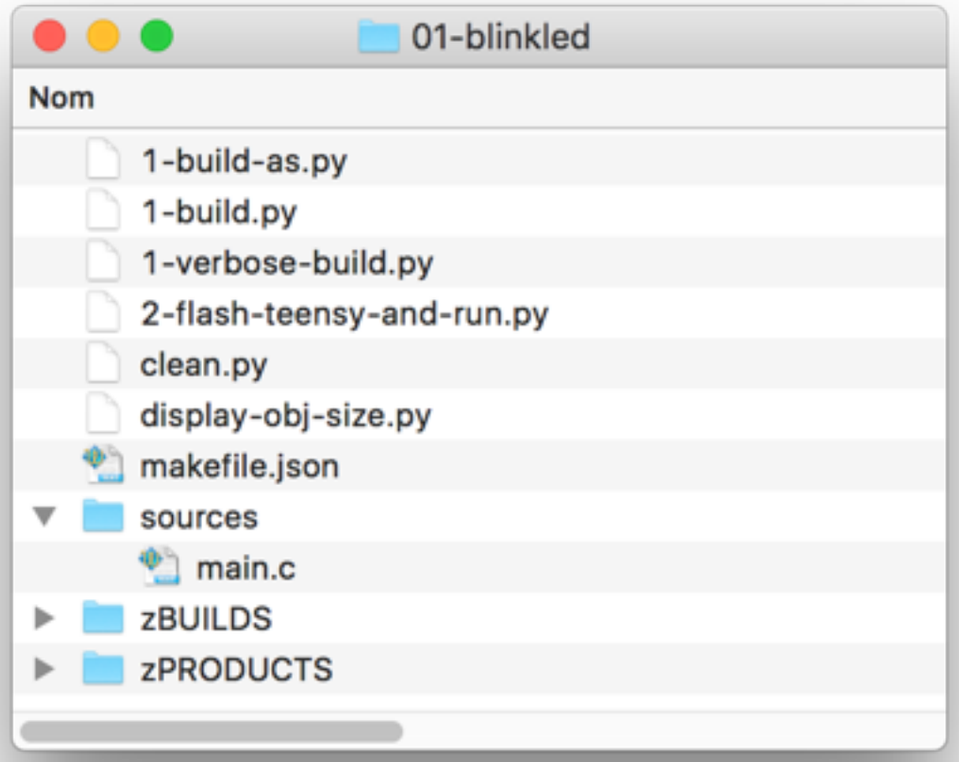
Pour compiler le programme, rien de plus simple : il suffit de double-cliquer sur **`1-build.py`**. Si les installations ont été correctement faites, le script s'exécute avec succès.

Il produit un ensemble de fichiers et de répertoires qui sont décrits dans les pages suivantes.

```
--- Making /Volumes/dev-svn/dev-lpc2294/etapes-realisation-executif-teensy-3-1/01-blinkled
Building clean.py
Building 2-flash-teensy-and-run.py
Making "zBUILDS" directory
Compiling (thumb) main.c
Building zBUILDS/internal-flash.ld
Building 1-verbose-build.py
Building 1-build-as.py
Building display-obj-size.py
Checking (thumb) main.c
Make clean.py executable
Make 1-verbose-build.py executable
Make 1-build-as.py executable
Make display-obj-size.py executable
Making "zPRODUCTS" directory
Linking zPRODUCTS/product-internal-flash.elf
Hexing zPRODUCTS/product-internal-flash.ihex
*** Internal FLASH:
  ROM code:    1164 bytes
  ROM data:    0 bytes
  RAM + STACK: 1024 bytes
```

Le répertoire d'un programme après compilation

`1-build-as.py`	Construit les fichiers assembleur issus de la compilation C dans un répertoire `zASBUILDS`.
`1-verbose-build.py`	Effectue la construction séquentiellement ; plus long, mais permet de mieux repérer les erreurs de compilation.
`2-flash-teensy-and-run.py`	Lancer ce script pour flasher et exécuter le programme.
`clean.py`	Efface tous les fichiers construits.
`display-obj-size.py`	Affiche les informations relatives aux fichiers objet engendrés.
`zBUILDS`	Répertoire contenant les fichiers engendrés
`zPRODUCTS`	Répertoire contenant les fichiers exécutables.



Flashage et exécution du programme

1) Connecter la carte avec le cordon USB : le programme en Flash s'exécute.

2) Double cliquer sur le script **`2-flash-teensy-and-run`** :

(1) ceci lance le programme de flashage ;

```
Teensy Loader, Command Line, Version 2.0
Read "zPRODUCTS/product-internal-flash.ihex": 1164 bytes, 0.9% usage
Waiting for Teensy device...
 (hint: press the reset button)
```

(3) appuyer fugitivement sur le poussoir Teensy pour lancer le flashage ;

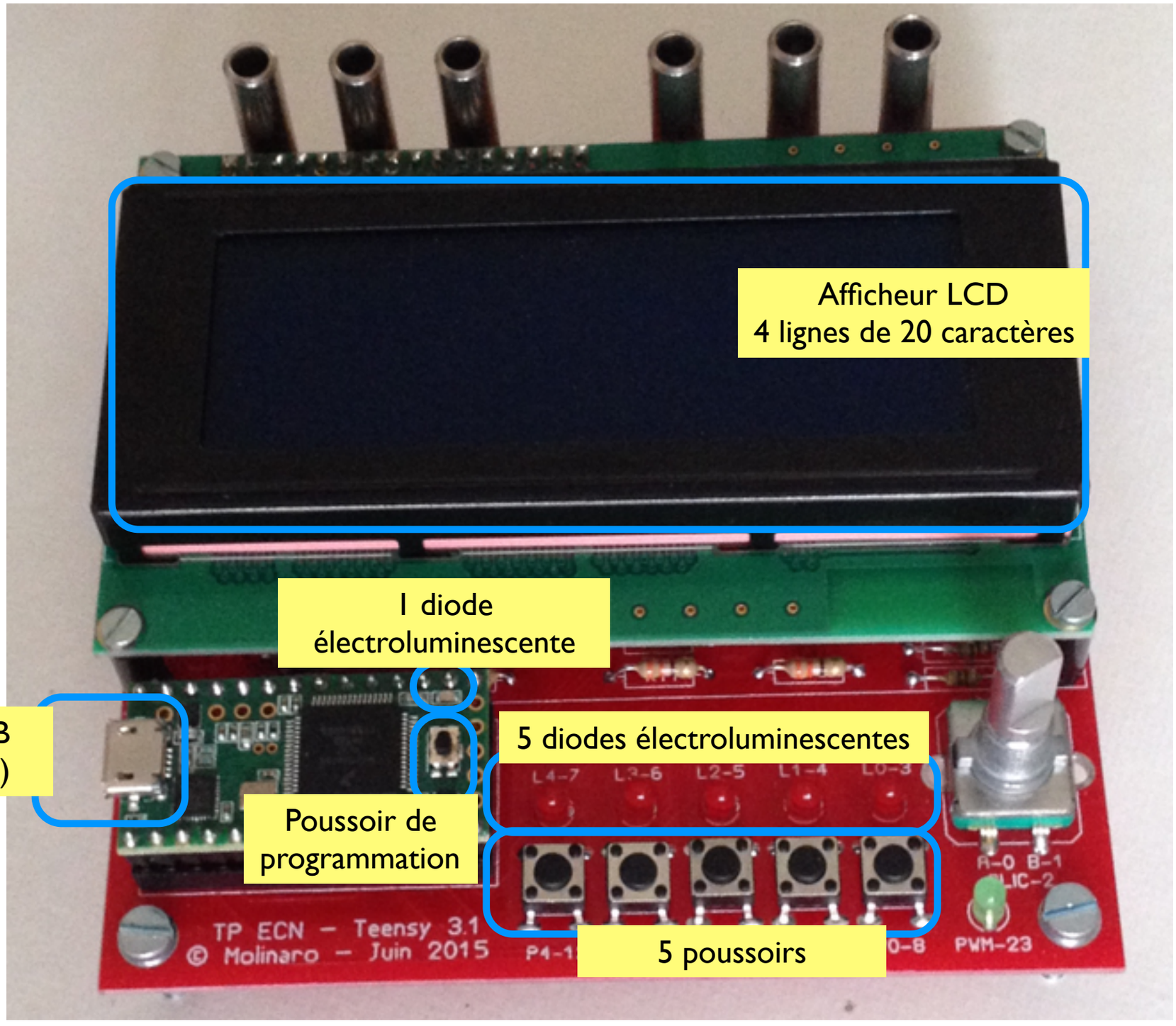
```
Found HalfKay Bootloader
Read "zPRODUCTS/product-internal-flash.ihex": 1164 bytes, 0.9% usage
Programming..
Booting
logout
```

(4) le flashage s'effectue ;

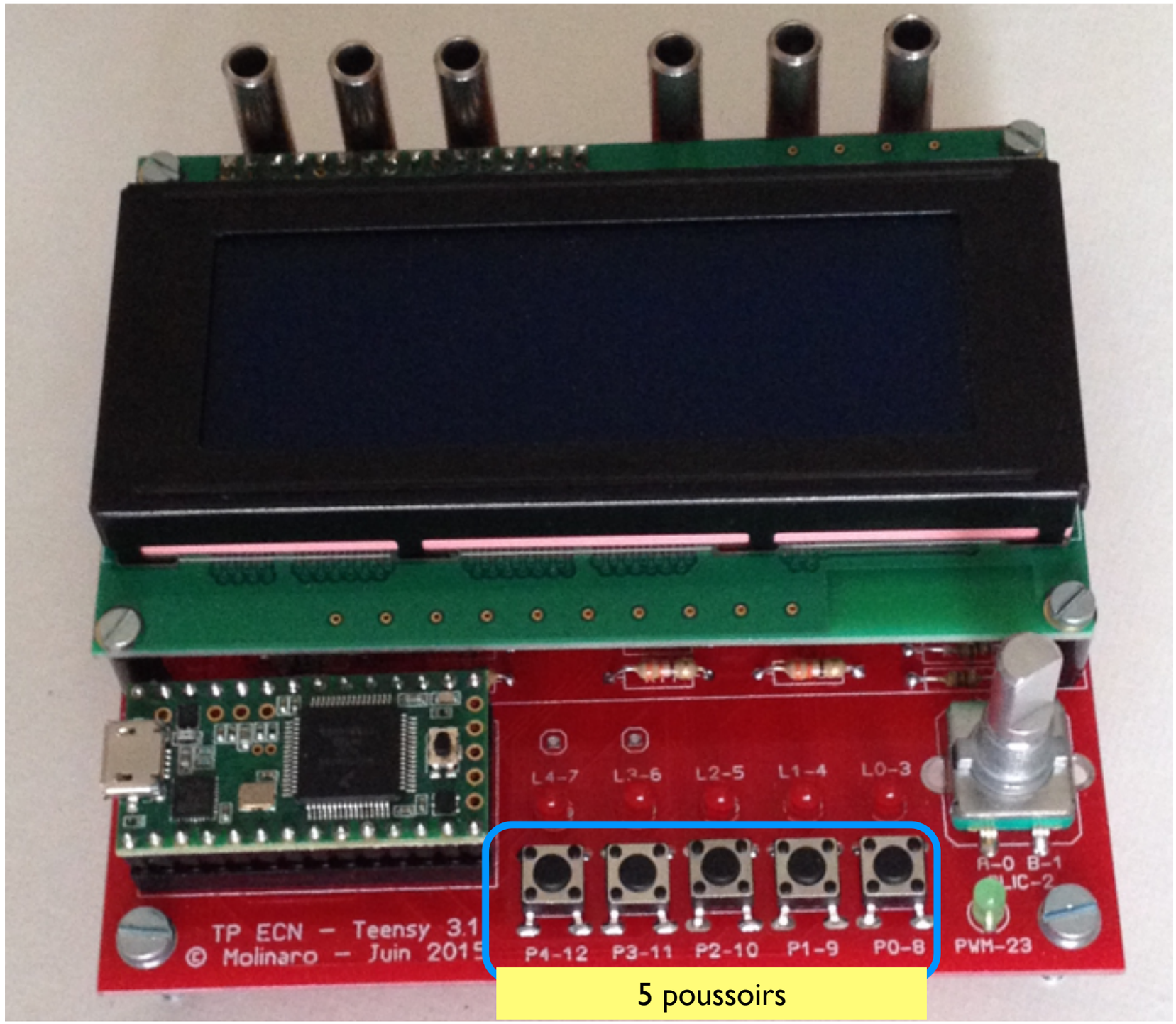
(5) quand il est achevé, le script se termine, et le programme flashé s'exécute.

3 — Matériel de développement

Entrées / sorties de la carte de TP

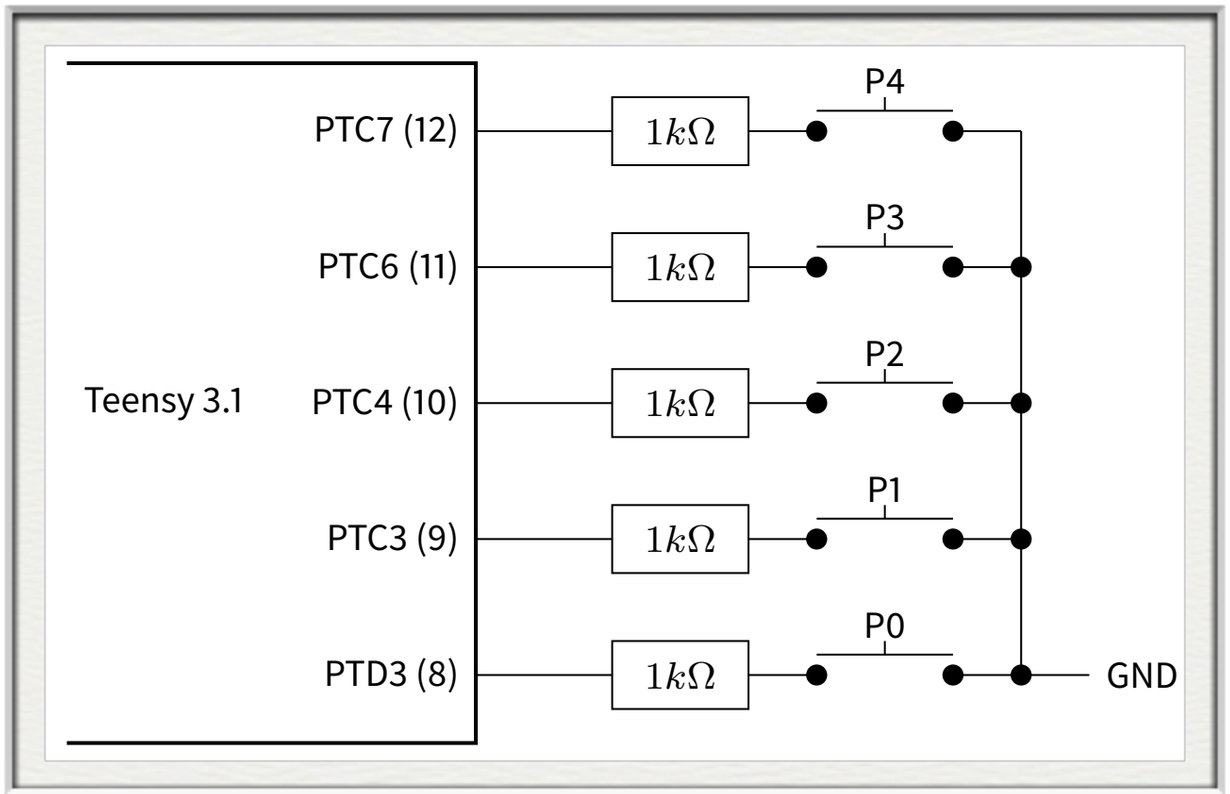


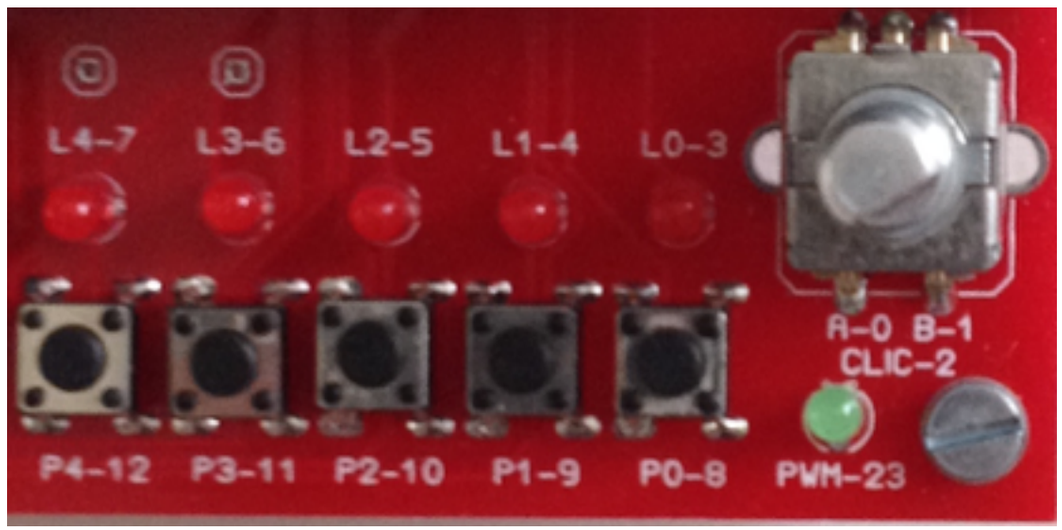
Les entrées logiques de la carte de TP



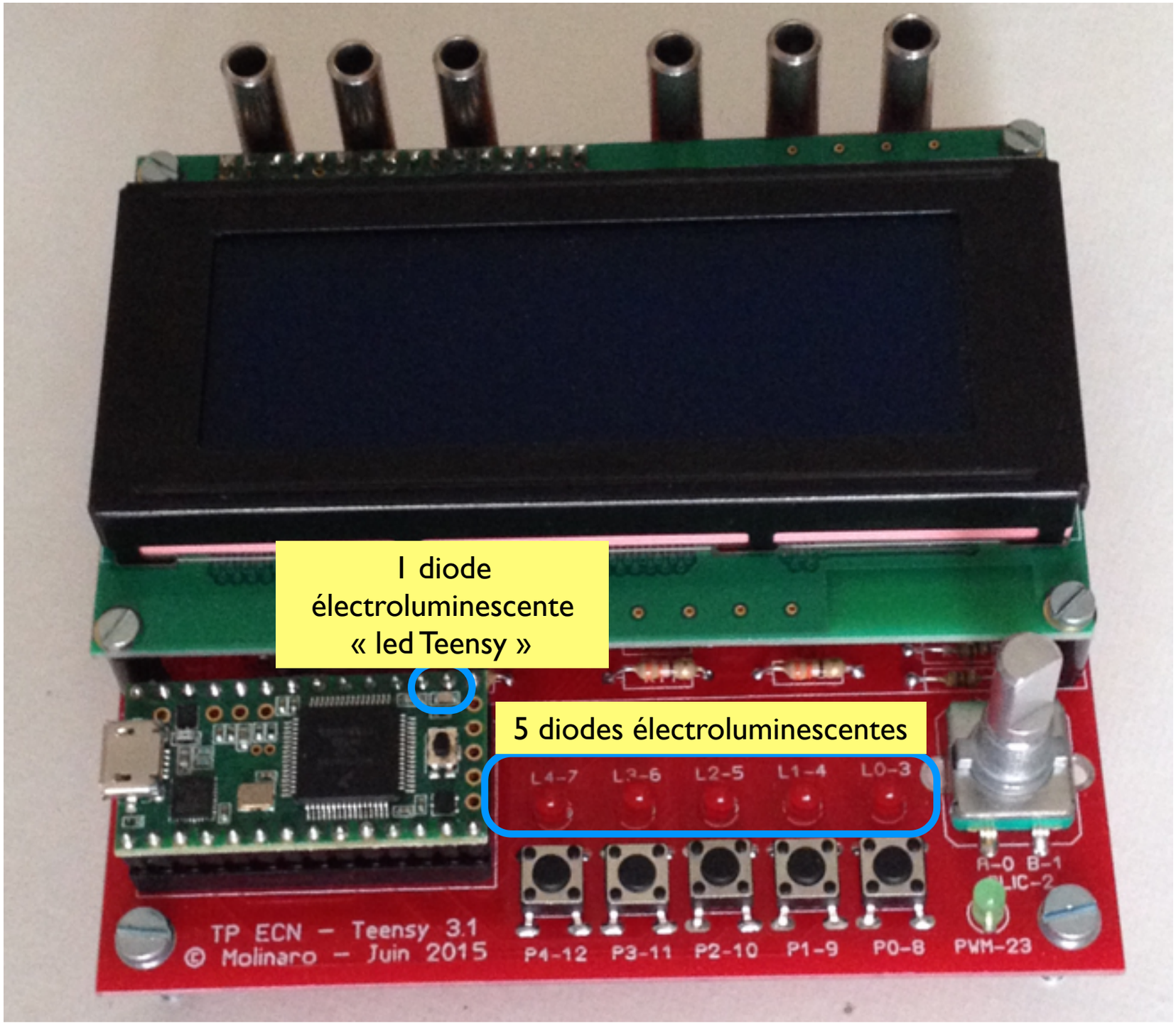
5 poussoirs

Connexion des entrées logiques de la carte de TP

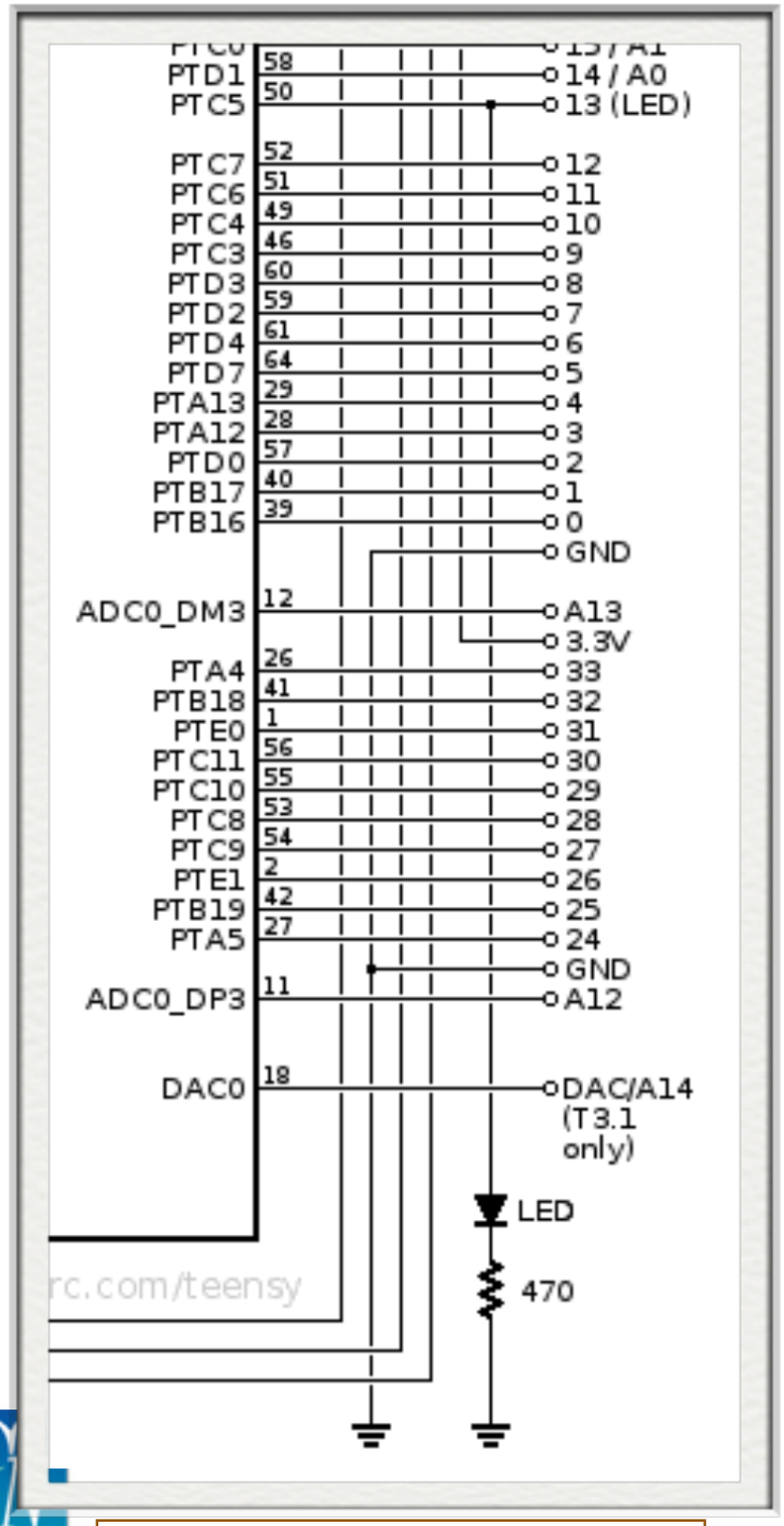




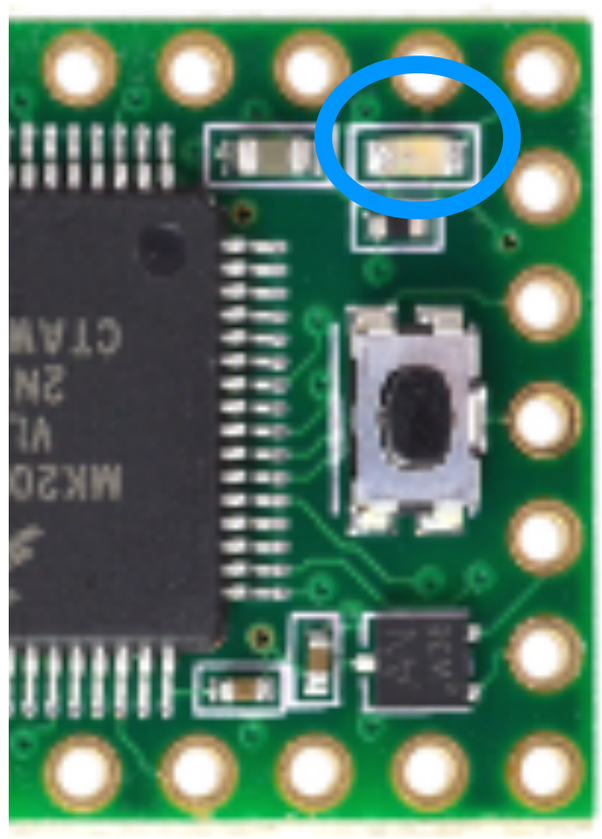
Sorties logiques de la carte de TP



Diode électroluminescente sur la carte Teensy



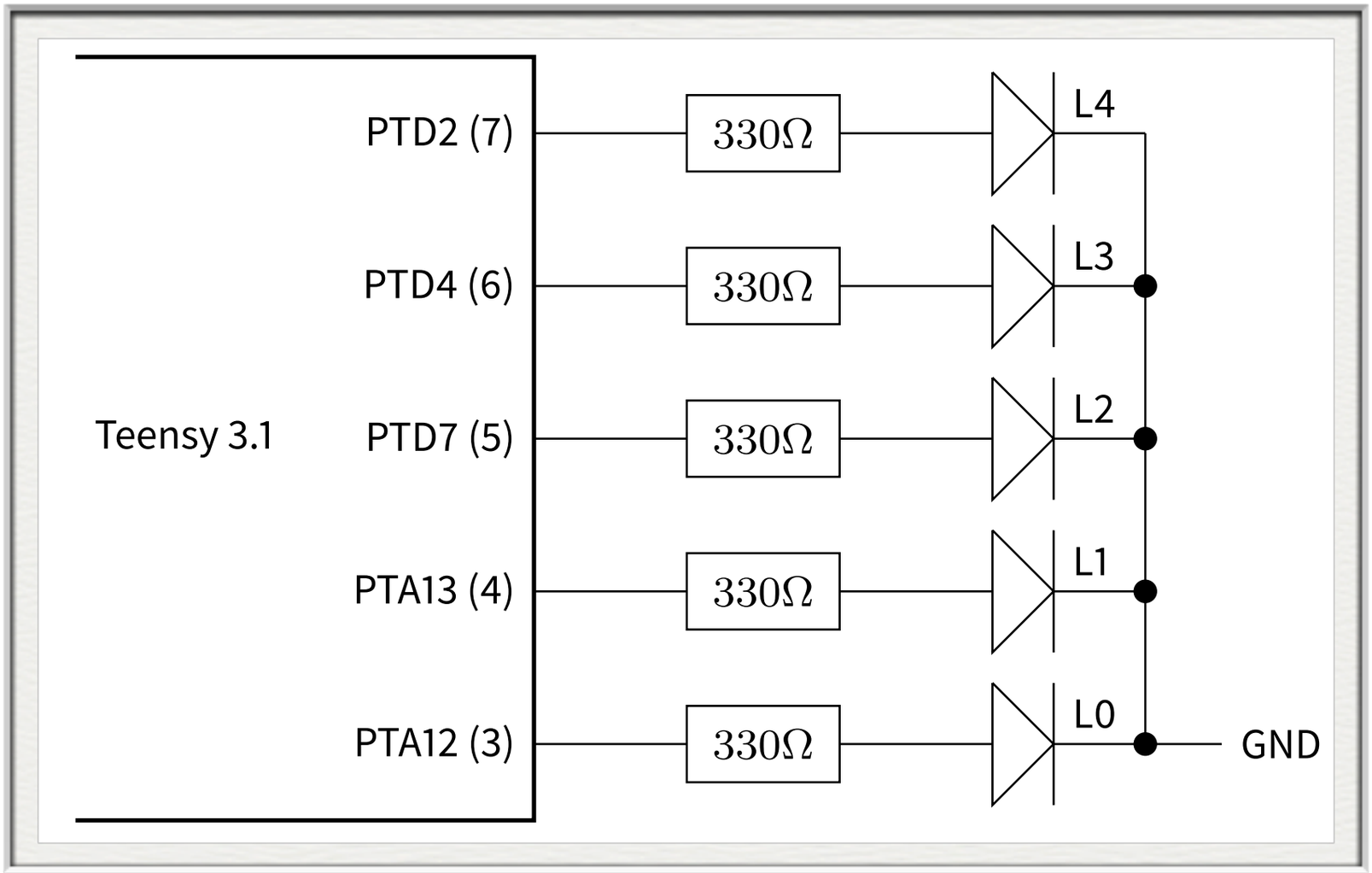
« Led Teensy »

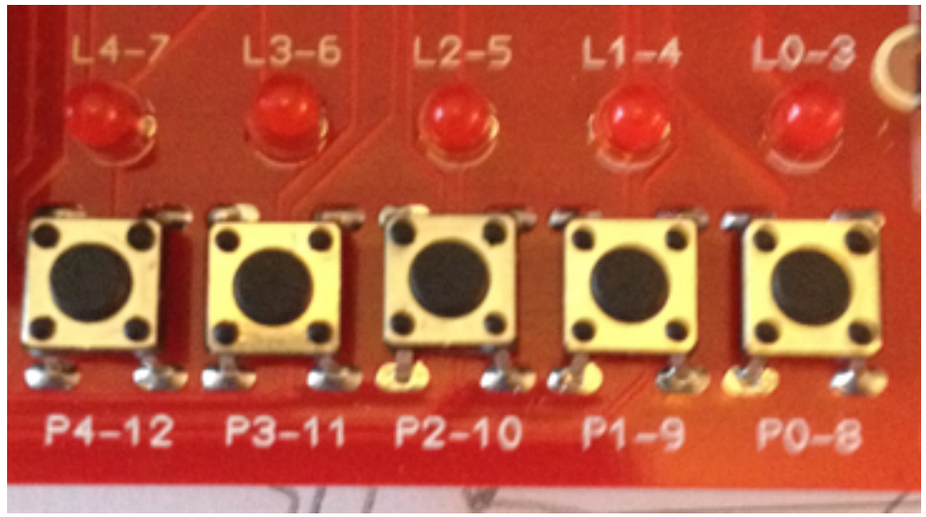


Port n°13 en sortie logique au niveau bas : le micro-contrôleur impose une tension proche de zéro Volt, la led est éteinte.

Port n°13 en sortie logique au niveau haut : le micro-contrôleur impose une tension proche de 3,3V, la led est allumée.

Diodes électroluminescentes sur la carte de TP





Port n°7 en sortie logique au niveau bas : le micro-contrôleur impose une tension proche de zéro Volt, la led `L4` est éteinte.

Port n°7 en sortie logique au niveau haut : le micro-contrôleur impose une tension proche de 3,3V, la led `L4` est allumée.

Carte mémoire du micro-contrôleur

256 Kio de mémoire Flash.
(espace de 128 Mio, seuls les premiers 256 Kio sont utilisés)

16 Kio de mémoire RAM.
(espace de 1 Mio, seuls les derniers 16 Kio sont utilisés)

16 Kio de mémoire RAM.
(espace de 1 Mio, seuls les premiers 16 Kio sont utilisés)

Registres de contrôle

System 32-bit Address Range	Destination Slave	Access
0x0000_0000–0x07FF_FFFF	Program flash and read-only data (Includes exception vectors in first 1024 bytes)	All masters
0x0800_0000–0x0FFF_FFFF	Reserved	—
0x1000_0000–0x13FF_FFFF	FlexNVM	All masters
0x1400_0000–0x17FF_FFFF	FlexRAM	All masters
0x1800_0000–0x1BFF_FFFF	Reserved	—
0x1C00_0000–0x1FFF_FFFF	SRAM_L: Lower SRAM (ICODE/DCODE)	All masters
0x2000_0000–0x200F_FFFF	SRAM_U: Upper SRAM bitband region	All masters
0x2010_0000–0x21FF_FFFF	Reserved	–
0x2200_0000–0x23FF_FFFF	Aliased to SRAM_U bitband	Cortex-M4 core only
0x2400_0000–0x3FFF_FFFF	Reserved	–
0x4000_0000–0x4007_FFFF	Bitband region for peripheral bridge 0 (AIPS-Lite0)	Cortex-M4 core & DMA/EzPort
0x4008_0000–0x400F_EFFF	Bitband region for peripheral bridge 1 (AIPS-Lite1)	Cortex-M4 core & DMA/EzPort
0x400F_F000–0x400F_FFFF	Bitband region for general purpose input/output (GPIO)	Cortex-M4 core & DMA/EzPort

System 32-bit Address Range	Destination Slave	Access
0x4010_0000–0x41FF_FFFF	Reserved	–
0x4200_0000–0x43FF_FFFF	Aliased to peripheral bridge (AIPS-Lite) and general purpose input/output (GPIO) bitband	Cortex-M4 core only
0x4400_0000–0xDFFF_FFFF	Reserved	–
0xE000_0000–0xE00F_FFFF	Private peripherals	Cortex-M4 core only
0xE010_0000–0xFFFF_FFFF	Reserved	–

Registre de contrôle : emplacement mémoire dont la lecture et/ou l'écriture peut modifier le comportement du micro-contrôleur.

La mémoire Flash du micro-contrôleur

Le micro-contrôleur MK20DX256VLH7 intègre une mémoire FLASH de 256 Kio, placée aux adresses **`0x0`** à **`0x3_FFFF`**.

Une mémoire FLASH (http://fr.wikipedia.org/wiki/Mémoire_flash) conserve ses données lors de la mise hors tension : elle est donc utilisée pour conserver le code du programme à exécuter, et les données permanentes. En usage normal, le micro-contrôleur n'y fait que des accès en lecture.

Une mémoire FLASH est divisée en secteurs (128 secteurs de 2Kio chacun).

Il existe deux opérations qui modifient le contenu d'une mémoire FLASH :

- *l'effacement*, qui met tous les bits à 1 ; l'effacement d'une FLASH s'effectue par secteur ; il n'est pas possible d'effacer un octet ou un mot particulier, ce sont tous les octets d'un secteur qui sont mis à 1 ;
- la *programmation*, qui ne peut qu'écrire des zéros ; pour la FLASH interne, l'écriture doit s'effectuer par paquet de 16 octets, mais cette quantification est transparente à partir des outils que nous utiliserons.

La programmation ne peut pas écrire une valeur « 1 » dans un bit à « 0 », mais soit laisser tel quel un bit à « 1 », soit écrire un « 0 » dans un bit à « 1 » : pour ramener un bit à « 1 », il faut effacer tout le secteur auquel il appartient.

Le nombre de programmations est limitée : la technologie utilisée pour le micro-contrôleur MK20DX256VLH7 assure un nombre minimum de **10 000** cycles de programmation, et une rétention d'au moins 50 ans.

Vecteurs d'interruption du micro-contrôleur

Pour le micro-contrôleur MK20DX256VLH7, la table des vecteurs d'interruption est à l'adresse **0** (et donc évidemment dans la Flash).

Les 16 premiers vecteurs sont propres à l'architecture ARM Cortex-M4.

Les 240 suivants sont propres au micro-contrôleur MK20DX256VLH7, qui n'en utilise que 93.

Pour cet enseignement, seuls les premiers vecteurs *ARM Core* nous intéressent.

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
ARM Core System Handler Vectors						
0x0000_0000	0	–	–	–	ARM core	Initial Stack Pointer
0x0000_0004	1	–	–	–	ARM core	Initial Program Counter
0x0000_0008	2	–	–	–	ARM core	Non-maskable Interrupt (NMI)
0x0000_000C	3	–	–	–	ARM core	Hard Fault
0x0000_0010	4	–	–	–	ARM core	MemManage Fault
0x0000_0014	5	–	–	–	ARM core	Bus Fault
0x0000_0018	6	–	–	–	ARM core	Usage Fault
0x0000_001C	7	–	–	–	—	—
0x0000_0020	8	–	–	–	—	—
0x0000_0024	9	–	–	–	—	—
0x0000_0028	10	–	–	–	—	—
0x0000_002C	11	–	–	–	ARM core	Supervisor call (SVCall)
0x0000_0030	12	–	–	–	ARM core	Debug Monitor
0x0000_0034	13	–	–	–	—	—
0x0000_0038	14	–	–	–	ARM core	Pendable request for system service (PendableSrvReq)
0x0000_003C	15	–	–	–	ARM core	System tick timer (SysTick)
Non-Core Vectors						
0x0000_0040	16	0	0	0	DMA	DMA channel 0 transfer complete
0x0000_0044	17	1	0	0	DMA	DMA channel 1 transfer complete
0x0000_0048	18	2	0	0	DMA	DMA channel 2 transfer complete
0x0000_004C	19	3	0	0	DMA	DMA channel 3 transfer complete
0x0000_0050	20	4	0	1	DMA	DMA channel 4 transfer complete
0x0000_0054	21	5	0	1	DMA	DMA channel 5 transfer complete
0x0000_0058	22	6	0	1	DMA	DMA channel 6 transfer complete

• • •

0x0000_01AC	107	91	2	22	Port control module	Pin detect (Port E)
0x0000_01B0	108	92	2	23	—	—
0x0000_01B4	109	93	2	23	—	—
0x0000_01B8	110	94	2	23	Software	Software interrupt[4]

Démarrage du micro-contrôleur

Un processeur ARM Cortex-M4 démarre de la façon suivante :

- la valeur initiale du pointeur de pile est chargée à partir de l'adresse **0** ;
- la valeur initiale du compteur programme est chargée à partir de l'adresse **4**.

Le micro-contrôleur MK20DX256VLH7 ajoute des conditions complémentaires :

- le champ de configuration de la Flash (**0x400** à **0x40F**, voir ci-contre) doit être initialisé ;
- le micro-contrôleur démarre avec une horloge interne de fréquence 8 MHz environ ;
- le *watchdog timer* est par défaut activé.

Un *watchdog timer* est un timer qui rédémarre le micro-contrôleur si il arrive à échéance.

The program flash memory contains a 16-byte flash configuration field that stores default protection settings (loaded on reset) and security information that allows the MCU to restrict access to the flash memory module.

Flash Configuration Field Byte Address	Size (Bytes)	Field Description
0x0_0400 - 0x0_0407	8	Backdoor Comparison Key. Refer to Verify Backdoor Access Key Command and Unsecuring the Chip Using Backdoor Key Access.
0x0_0408 - 0x0_040B	4	Program flash protection bytes. Refer to the description of the Program Flash Protection Registers (FPROT0-3).
0x0_040F	1	Data flash protection byte. Refer to the description of the Data Flash Protection Register (FDPROT).
0x0_040E	1	EEPROM protection byte. Refer to the description of the EEPROM Protection Register (FEPROT).
0x0_040D	1	Flash nonvolatile option byte. Refer to the description of the Flash Option Register (FOPT).
0x0_040C	1	Flash security byte. Refer to the description of the Flash Security Register (FSEC).

4 — Description du programme `01-blinkled`

Détail de la construction (1-verbose-build.py)

```
--- Making /Volumes/dev-svn/info-treel/etapes-realisation-executif-teensy-3-1/01-blinkled

Compiling (thumb) main.c
/Users/pierremolinaro/dev-arm/dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0/bin/arm-none-
eabi-gcc -mcpu=cortex-m4 -mthumb -fomit-frame-pointer -Os -foptimize-register-move -Wall -Werror -Wreturn-
type -Wformat -Wsign-compare -Wpointer-arith -Wparentheses -Wcast-align -Wcast-qual -Wwrite-strings -Wswitch
-Wuninitialized -ffunction-sections -fdata-sections -fno-stack-protector -fshort-enums -std=c99 -Wstrict-
prototypes -Wbad-function-cast -Wmissing-declarations -Wimplicit-function-declaration -Wno-int-to-pointer-
cast -Wno-pointer-to-int-cast -Wmissing-prototypes -c sources/main.c -o zBUILDS/main.c.oo -Isources -MD -MP -
MF zBUILDS/main.c.oo.dep

Building zBUILDS/internal-flash.ld
python /Volumes/dev-svn/dev-lpc2294/dev-files/python-scripts/build-linker-script.py --flash=0:256k --
ram=sram_u:0x20000000:32k --bss=sram_u --data=sram_u --stack=sram_u:1k --other-stacks= --heap=sram_u --
sections= --vector=flash --code=flash --output=zBUILDS/internal-flash.ld

Linking zPRODUCTS/product-internal-flash.elf
/Users/pierremolinaro/dev-arm/dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0/bin/arm-none-
eabi-gcc -mcpu=cortex-m4 -mthumb zBUILDS/main.c.oo -TzBUILDS/internal-flash.ld -Wl,-Map=zPRODUCTS/product-
internal-flash.map -o zPRODUCTS/product-internal-flash.elf -nostartfiles -Wl,--fatal-warnings -Wl,--warn-
common -Wl,--no-undefined -Wl,--cref -lc -lgcc -lm -Wl,-static -Wl,-s -Wl,--gc-sections

Hexing zPRODUCTS/product-internal-flash.ihex
/Users/pierremolinaro/dev-arm/dev-arm-Intel-Darwin-gccarm-5_2-2015q4-20151219-openocd-0.8.0/bin/arm-none-
eabi-objcopy -O ihex zPRODUCTS/product-internal-flash.elf zPRODUCTS/product-internal-flash.ihex
*** Internal FLASH:
  ROM code:    1164 bytes
  ROM data:    0 bytes
  RAM + STACK: 1024 bytes
```

Compilation C

Construction du script d'édition de liens

Édition de liens

Traduction en format HEX

Description de la table des vecteurs d'interruption

```
typedef struct {
  uint32_t * mStackPointer ;
  void (* mResetHandler) (void) ;
  int32_t mUnused [254] ;
  int32_t mFlash [4] ;
} vectorStructTemp ;
```

Le type `vectorStructTemp` décrit composition de la table des vecteurs d'interruption du micro-contrôleur, ainsi que la configuration de la Flash.

Le placement de la table à l'adresse `0` ne peut pas être spécifié ici.

Address	Vector	IRQ[1]	NVIC non-IPR register number [2]	NVIC IPR register number [3]	Source module	Source description
ARM Core System Handler Vectors						
0x0000_0000	0	–	–	–	ARM core	Initial Stack Pointer
0x0000_0004	1	–	–	–	ARM core	Initial Program Counter
0x0000_0008	2	–	–	–	ARM core	Non-maskable Interrupt (NMI)
0x0000_000C	3	–	–	–	ARM core	Hard Fault
0x0000_0010	4	–	–	–	ARM core	MemManage Fault
0x0000_0014	5	–	–	–	ARM core	Bus Fault
0x0000_0018	6	–	–	–	ARM core	Usage Fault
0x0000_001C	7	–	–	–	—	—
0x0000_0020	8	–	–	–	—	—
0x0000_0024	9	–	–	–	—	—
0x0000_0028	10	–	–	–	—	—
0x0000_002C	11	–	–	–	ARM core	Supervisor call (SVCall)
0x0000_0030	12	–	–	–	ARM core	Debug Monitor
0x0000_0034	13	–	–	–	—	—
0x0000_0038	14	–	–	–	ARM core	Pendable request for system service (PendableSrvReq)
0x0000_003C	15	–	–	–	ARM core	System tick timer (SysTick)
Non-Core Vectors						
0x0000_0040	16	0	0	0	DMA	DMA channel 0 transfer complete
0x0000_0044	17	1	0	0	DMA	DMA channel 1 transfer complete
0x0000_0048	18	2	0	0	DMA	DMA channel 2 transfer complete
0x0000_004C	19	3	0	0	DMA	DMA channel 3 transfer complete
0x0000_0050	20	4	0	1	DMA	DMA channel 4 transfer complete
0x0000_0054	21	5	0	1	DMA	DMA channel 5 transfer complete
0x0000_0058	22	6	0	1	DMA	DMA channel 6 transfer complete
•••						
0x0000_01AC	107	91	2	22	Port control module	Pin detect (Port E)
0x0000_01B0	108	92	2	23	—	—
0x0000_01B4	109	93	2	23	—	—
0x0000_01B8	110	94	2	23	Software	Software interrupt[4]

Description de la table des vecteurs d'interruption

La constante `vector`, de type `vectorStructTemp`, décrit la table des vecteurs d'interruption du micro-contrôleur, ainsi que la configuration de la Flash :

- `__system_stack_end` est défini lors de l'édition de liens ;
- `ResetISR` est le nom de la routine de démarrage.

Le placement de la table à l'adresse `0` sera résolu à l'édition de liens grâce à la déclaration :
`__attribute__((section (".isr_vector")))`

```
extern uint32_t __system_stack_end ;

const vectorStructTemp vector
__attribute__ ((section (".isr_vector"))) = {
  & __system_stack_end,
  ResetISR,
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {-1, -1, -1, -2}
} ;
```

La routine ResetISR

Inhibition du *watchdog*

Activation du port C

Programmation de PTC5 (led *Teensy*) en sortie logique

PTC5 au niveau haut : la led *Teensy* s'allume

Boucle d'attente

PTC5 au niveau bas : la led *Teensy* s'éteint

```
static void ResetISR (void) {
  WDOG_UNLOCK = 0xC520 ;
  WDOG_UNLOCK = 0xD928 ;
  WDOG_STCTRLH = 0x0010 ;

  SIM_SCGC5 |= 1 << 11 ;

  PORTC_PCR5 = (1 << 8) ;
  GPIOC_PDDR |= (1 << 5) ;
  while (1) {
    GPIOC_PSOR = 1 << 5 ;
    volatile uint32_t j ;
    for (j=0; j<200000 ; j++) {}
    GPIOC_PCOR = 1 << 5 ;
    for (j=0; j<200000 ; j++) {}
  }
}
```

Les registres de contrôle (`WDOG_UNLOCK`, …) sont écrits en majuscule pour être plus facilement repérés. La page suivante montre comment si ils sont définis.

Les boucles d'attente imposent que `j` soit déclaré `volatile`.

Définition du registre de contrôle WDOG_UNLOCK

Absolute address (hex)	Register name	Width (in bits)	Access	Reset value	Section/ page
4005_2000	Watchdog Status and Control Register High (WDOG_STCTRLH)	16	R/W	01D3h	23.7.1/474
4005_2002	Watchdog Status and Control Register Low (WDOG_STCTRLL)	16	R/W	0001h	23.7.2/475
4005_2004	Watchdog Time-out Value Register High (WDOG_TOVALH)	16	R/W	004Ch	23.7.3/476
4005_2006	Watchdog Time-out Value Register Low (WDOG_TOVALL)	16	R/W	4B4Ch	23.7.4/476
4005_2008	Watchdog Window Register High (WDOG_WINH)	16	R/W	0000h	23.7.5/477
4005_200A	Watchdog Window Register Low (WDOG_WINL)	16	R/W	0010h	23.7.6/477
4005_200C	Watchdog Refresh register (WDOG_REFRESH)	16	R/W	B480h	23.7.7/478
4005_200E	Watchdog Unlock register (WDOG_UNLOCK)	16	R/W	D928h	23.7.8/478
4005_2010	Watchdog Timer Output Register High (WDOG_TMROUTH)	16	R/W	0000h	23.7.9/478
4005_2012	Watchdog Timer Output Register Low (WDOG_TMROUTL)	16	R/W	0000h	23.7.10/479
4005_2014	Watchdog Reset Count register (WDOG_RSTCNT)	16	R/W	0000h	23.7.11/479
4005_2016	Watchdog Prescaler register (WDOG_PRESC)	16	R/W	0400h	23.7.12/480

WDOG_UNLOCK est un registre de 16 bits à l'adresse 0x4005_200E.

Son accès est défini par la macro :

```
#define WDOG_UNLOCK  (*((volatile uint16_t *) 0x4005200E))
```

La déclaration `volatile`

Une variable volatile est une variable sur laquelle aucune optimisation de compilation n'est appliquée (https://fr.wikipedia.org/wiki/Variable_volatile).

Référence :

https://en.wikipedia.org/wiki/Volatile_(computer_programming)

```
while (1) {
  GPIOC_PSOR = 1 << 5 ;
  volatile uint32_t j ;
  for (j=0; j<200000 ; j++) {}
  GPIOC_PCOR = 1 << 5 ;
  for (j=0; j<200000 ; j++) {}
}
```

```
.L6:
    str  r2, [r4]
    str  r1, [sp, #4]
.L2:
    ldr  r3, [sp, #4]
    cmp  r3, r0
    bhi  .L8
    ldr  r3, [sp, #4]
    adds r3, r3, #1
    str  r3, [sp, #4]
    b    .L2
.L8:
    str  r2, [r5]
    str  r1, [sp, #4]
.L4:
    ldr  r3, [sp, #4]
    cmp  r3, r0
    bhi  .L6
    ldr  r3, [sp, #4]
    adds r3, r3, #1
    str  r3, [sp, #4]
    b    .L4
```

```
while (1) {
  GPIOC_PSOR = 1 << 5 ;
  uint32_t j ;
  for (j=0; j<200000 ; j++) {}
  GPIOC_PCOR = 1 << 5 ;
  for (j=0; j<200000 ; j++) {}
}
```

```
.L3:
    ldr  r3, .L6+24
    str  r2, [r0]
.L2:
    subs r3, r3, #1
    bne  .L2
    str  r2, [r1]
    b    .L3
```

Le fichier `main.c`

```c
#include <stdint.h>

#define PORTC_PCR5 (*((volatile uint32_t *) 0x4004B014))

#define GPIOC_PDDR (*((volatile uint32_t *) 0x400FF094))
#define GPIOC_PSOR (*((volatile uint32_t *) 0x400FF084))
#define GPIOC_PCOR (*((volatile uint32_t *) 0x400FF088))

#define SIM_SCGC5  (*((volatile uint32_t *) 0x40048038))

#define WDOG_STCTRLH (*((volatile uint16_t *) 0x40052000))
#define WDOG_UNLOCK  (*((volatile uint16_t *) 0x4005200E))

static void ResetISR (void) {
  WDOG_UNLOCK = 0xC520 ;
  WDOG_UNLOCK = 0xD928 ;
  WDOG_STCTRLH = 0x0010 ;

  SIM_SCGC5 |= 1 << 11 ;

  PORTC_PCR5 = (1 << 8) ;
  GPIOC_PDDR |= (1 << 5) ;
  while (1) {
    GPIOC_PSOR = 1 << 5 ;
    volatile uint32_t j ;
    for (j=0; j<200000 ; j++) {}
    GPIOC_PCOR = 1 << 5 ;
    for (j=0; j<200000 ; j++) {}
  }
}

typedef struct {
  uint32_t * mStackPointer ;
  void (* mResetHandler) (void) ;
  int32_t mUnused [254] ;
  int32_t mFlash [4] ;
} vectorStructTemp ;

extern uint32_t __system_stack_end ;

const vectorStructTemp vector
__attribute__ ((section (".isr_vector"))) = {
  & __system_stack_end,
  ResetISR,
  { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {-1, -1, -1, -2}
} ;
```

Le fichier d'édition de liens `internal-flash.ld`

Le fichier d'édition de liens décrit comment l'édition des liens est réalisée. Dans la chaîne de compilation utilisée, un script Python construit ce fichier.

```
Building zBUILDS/internal-flash.ld
python /Volumes/dev-svn/dev-lpc2294/dev-files/python-scripts/build-linker-script.py --
flash=0:256k --ram=sram_u:0x20000000:32k --bss=sram_u --data=sram_u --stack=sram_u:1k --other-
stacks= --heap=sram_u --sections= --vector=flash --code=flash --output=zBUILDS/internal-flash.ld
```

Dans la suite, nous allons uniquement voir les constructions qui sont utiles dans ce premier programme.

Le deuxième programme utilisera toutes les constructions du fichier d'éditions de liens.

Le fichier d'édition de liens `internal-flash.ld`

Description de la carte mémoire

```
/*-----------------------------------------------------*/
/*                      Memory                         */
/*-----------------------------------------------------*/

MEMORY {
  flash (rx) : ORIGIN = 0, LENGTH = 256k
  sram_u (rwx) : ORIGIN = 0x20000000, LENGTH = 32k
}

/*-----------------------------------------------------*/

__sram_u_end = 0x20000000 + 32k ;
```

Le fichier d'édition de liens `internal-flash.ld`

Placement de la table des vecteurs d'interruption

```
/*-----------------------------------------------------*/
/*                  ISR Vectors                        */
/*-----------------------------------------------------*/

SECTIONS {
  .vectors : {
    __vectors_start = . ;
    KEEP (*(.isr_vector)) ;
    __vectors_end = . ;
  } > flash
}
```

Le fichier d'édition de liens `internal-flash.ld`

Pile système

```
/*-----------------------------------------------------*/
/*           System stack                              */
/*-----------------------------------------------------*/

SECTIONS {
  .system_stack :{
    . = ALIGN (4) ;
    __system_stack_start = . ;
    . += 1k ;
    . = ALIGN (4) ;
    __system_stack_end = . ;
  } > sram_u
}
```

5 — Complément : opérateurs du langage C

L’opérateur « et bit-à-bit » : « & »

Attention, ne pas confondre « & » et « && ».

L’opérateur « & » effectue une opération *et* bit-à-bit sur des nombres entiers.

Voici un exemple sur des nombres de 8 bits :

	7	6	5	4	3	2	1	0
A	1	0	0	0	1	0	0	0
B	1	0	0	0	0	0	1	0
A & B	1	0	0	0	0	0	0	0

L'opérateur « et logique » : « && »

L'opérateur && : en C, cet opérateur n'évalue pas l'opérande de droite si celui de gauche est faux.

Ne pas confondre avec l'opérateur `&` (page précédente).

Par exemple :

```
if (a && b) {
   X
}else{
   Y
}
```

est équivalent à :

```
if (a) {
   if (b) {
      X
   }else{
      Y
   }
}else{
   Y
}
```

L'opérateur « ou bit-à-bit » : « | »

Attention, ne pas confondre « | » et « || ».

L'opérateur « | » effectue une opération *ou* bit-à-bit sur des nombres entiers.

Voici un exemple sur des nombres de 8 bits :

	7	6	5	4	3	2	1	0
A	1	0	0	0	1	0	0	0
B	1	0	0	0	0	0	1	0
A | B	1	0	0	0	1	0	1	0

L'opérateur « ou logique » : « || »

L'opérateur || : en C, cet opérateur n'évalue pas l'opérande de droite si celui de gauche est vrai.

Ne pas confondre avec l'opérateur | (page précédente).

Par exemple :

```
if (a || b) {
   X
}else{
   Y
}
```

est équivalent à :

```
if (a) {
   X
}else if (b) {
   X
}else{
   Y
}
```

L'opérateur de décalage à gauche : « << »

Cet opérateur décale à gauche en insérant des zéros.

Un exemple sur des nombres de 8 bits :

	7	6	5	4	3	2	1	0
`A`	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
`A << 1`	a_6	a_5	a_4	a_3	a_2	a_1	a_0	0
`A << 2`	a_5	a_4	a_3	a_2	a_1	a_0	0	0

Conséquence : `1 << n` est le nombre dont le seul bit à 1 est le bit n°*n*.

	7	6	5	4	3	2	1	0
`1 << 0`	0	0	0	0	0	0	0	1
`1 << 1`	0	0	0	0	0	0	1	0
`1 << 2`	0	0	0	0	0	1	0	0

L'opérateur de décalage à droite : « >> »

Sur des nombres non signés, cet opérateur insère des zéros.

A est un nombre non signé	7	6	5	4	3	2	1	0
A	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
A >> 1	0	a_7	a_6	a_5	a_4	a_3	a_2	a_1
A >> 2	0	0	a_7	a_6	a_5	a_4	a_3	a_2

Sur des nombres signés, cet opérateur conserve le bit de poids fort.

A est un nombre signé	7	6	5	4	3	2	1	0
A	a_7	a_6	a_5	a_4	a_3	a_2	a_1	a_0
A >> 1	a_7	a_7	a_6	a_5	a_4	a_3	a_2	a_1
A >> 2	a_7	a_7	a_7	a_6	a_5	a_4	a_3	a_2