

# Python sur STM32



Tuteur: Olivier Richard

XIA Ye (Xavier)

TAO Xinxiu (Isabelle)

RICM4 - 04/2014

1	Introduction.....	3
2	Environment.....	3
3	Programme.....	4
1.	Utilisation.....	4
2.	Flow.....	5
3.	Modification du librairie.....	7
4.	L'affichage du mode Debug.....	7
5.	La structure des fichiers.....	8
4	Évaluation.....	8
5	Problème résolu.....	10
1.	Redirection de USB.....	10
2.	La panne de la carte.....	10

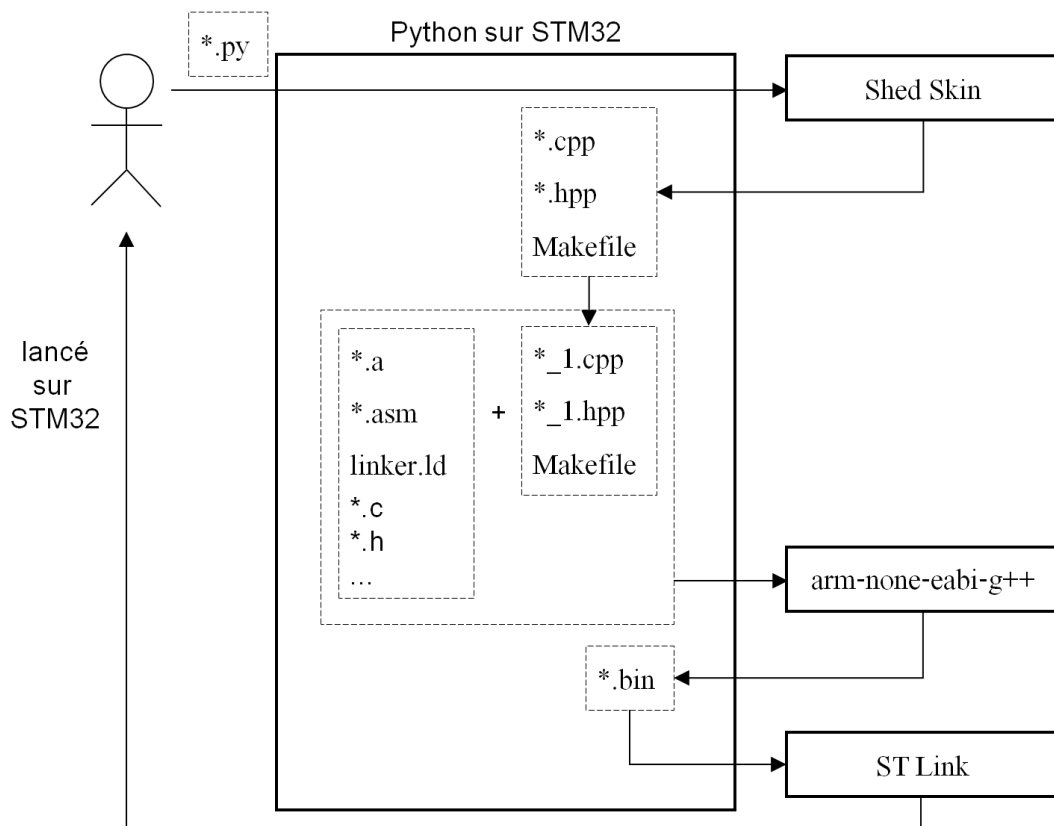
# 1 Introduction

L'objectif de ce projet est de proposer une chaîne de compilation d'un sous ensemble du langage Python pour les microcontrôleurs de la famille STM32F4. Le point de départ est le projet Shedskin un compilateur de python vers C++.

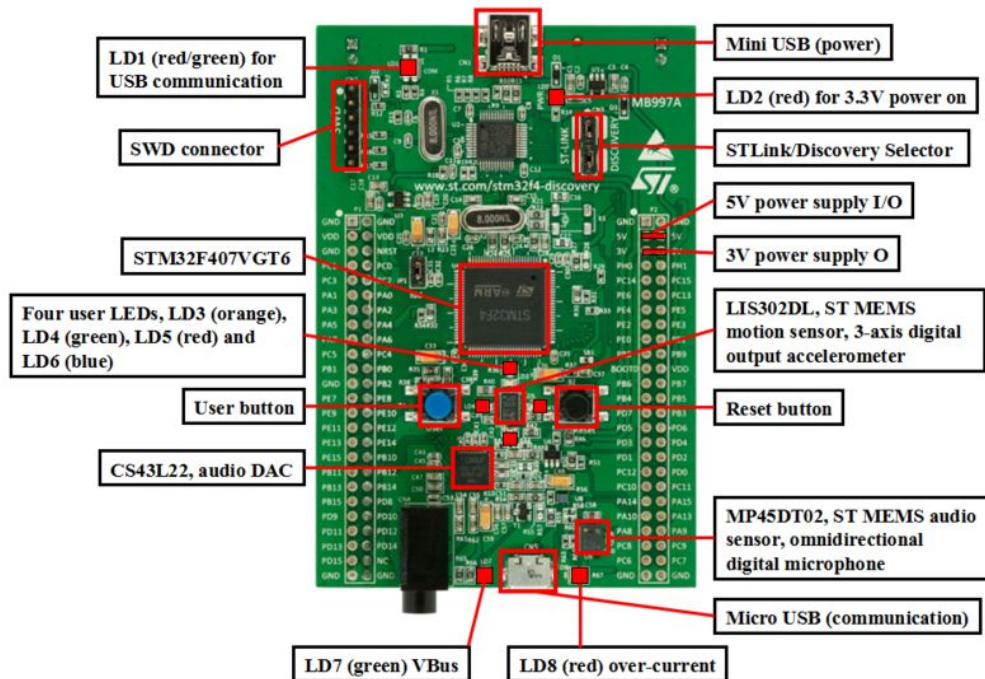
## 2 Environment

Ce projet est développé sous le système Linux. Pour l'environnement, on demande l'utilisateur d'installer les outils suivant: Shedskin (Pour la traduction), chaîne d'outils GNU/ARM (pour la compilation), ST-link (pour le téléchargement vers la carte), le projet STM32PLUS (pour supporter le STL). Les path des outils sont définis au début du script py2stm.tcl

Le diagramme de cas d'utilisation est comme le suivant:



On a appris la structure basique du matériel de la carte, on a fait une graphe pour indiquer les pièces comme le suivant :



## 3 Programme

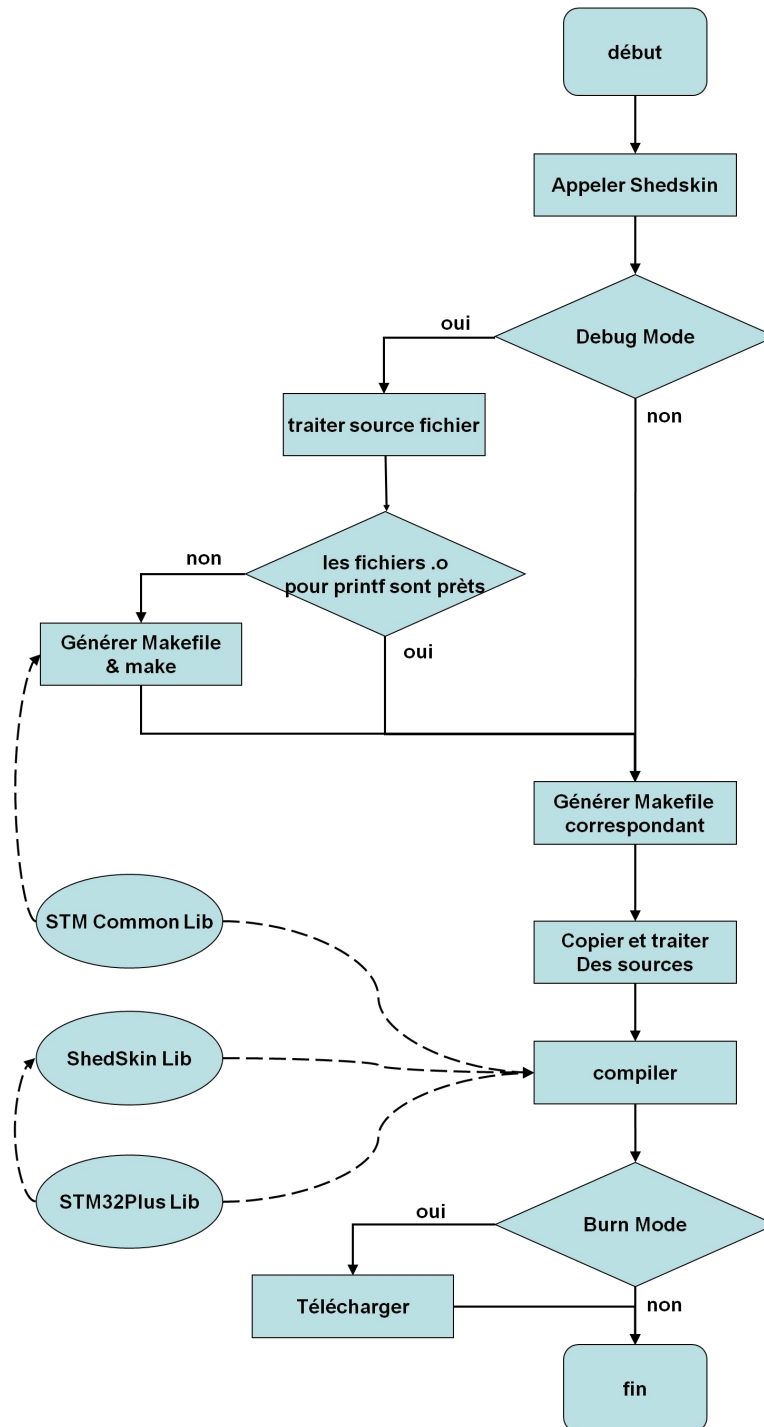
### 1. Utilisation

Pour faciliter les traitements du texte, on a réaliser notre projet avec le scripte tcl. À l'aide de notre projet, l'utilisateur peut finir la traduction, la compilation et le téléchargement avec une seule commande, l'utilisation du projet est comme le suivant:

```
[py2stm]$ ./py2stm.tcl
-----
      RICM Project : Python sur STM32
-----
Usage: py2stm.tcl [-d] [-b] -p py_file
[-d] : Debug Mode, will surpport printf in the project.
[-b] : Burn  Mode, will burn the program on the card when finish compile.
Example: py2stm.tcl -d -p /home/share/test.py
```

## 2. Flow

Le flow chart du projet est comme le suivant:



D'abord, on appelle le shedskin pour faire la traduction depuis le source python vers C++. L'affichage de cette étape est comme le suivant:

```
[py2stm]$ ./py2stm.tcl -p ./test/1.py -b
Translating...
*** SHED SKIN Python-to-C++ Compiler 0.9.4 ***
Copyright 2005-2011 Mark Dufour; License GNU GPL version 3 (See LICENSE)

[analyzing types..]

0%
*****100%
*****100%
*****100%
[generating c++ code..]
[elapsed time: 2.27 seconds]
```

Puis, si l'utilisateur a choisi le Debug mode, on va ajouter du code nécessaire pour supporter la fonction printf. Le code contient d'inclusion des fichiers tête nécessaire et les initialisation de USB, ici on a utilisé le librairie STM Common Lib, ce librairie est en C, on doit le compiler avec arm-none-eabi-gcc, mais le code traduit par shedskin est en C++, on le compile avec arm-none-eabi-g++. À cause de la différence entre C et C++, on a ajouté le mot clé "extern "C" {...}" pour les fichiers tête C dans le C++ code pour résoudre ce problème. Après cette étape, le programme va vérifier si les fichiers nécessaires dans le librairie pour le printf sont déjà compilé ou pas, sinon on va les compiler pour préparer de lier avec les projet traduit.

En suite, le programme va lire le Makefile généré par Shedskin, et générer un nouvel Makefile. Ce Makefile va considérer le mode l'utilisateur a choisi.

Après, le programme va copier les sources traduit par Shedskin et faire des traitements nécessaires. Ici on a supprimé les codes correspondent à l'exception(try catch throw ASSERT) et l'affichage (print2, une fonction supporté par shedskin, mais impossible pour l'exécution sur la carte).

Puis, le programme va faire la compilation à l'aide des 2 librairie Shedskin Lib et STM23Plus Lib. L'affichage est comme le suivant:

```
Compiling...
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-g++ -o ./re.o -c -pedantic-errors -fno-rtti -std=gnu++0x -fno-threadsafe-statics -pipe -ffunction-sections -fdata-sections -fno-exceptions -mthumb -gdwarf-2 -pipe -DHSE_VALUE=8000000 -mcpu=cortex-m4 -DSTM32PLUS_F4 -O5 -I/home/xavier/project/stm32/stm32plus/lib/include -I/home/xavier/project/stm32/stm32plus/lib/include/stl -I/home/xavier/project/stm32/stm32plus/lib -I. -I/home/xavier/ricm/mul_projet/py2stm/shedskin_lib -I/home/xavier/ricm/mul_projet/py2stm/shedskin_lib/builtin /home/xavier/ricm/mul_projet/py2stm/shedskin_lib/re.cpp
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-g++ -o ./builtin.o -c -pedantic-errors -fno-rtti -std=gnu++0x -fno-threadsafe-statics -pipe -ffunction-sections -fdata-sections -fno-exceptions -mthumb -gdwarf-2 -pipe -DHSE_VALUE=8000000 -mcpu=cortex-m4 -DSTM32PLUS_F4 -O5 -I/home/xavier/project/stm32/stm32plus/lib/include -I/home/xavier/project/stm32/stm32plus/lib/include/stl -I/home/xavier/project/stm32/stm32plus/lib -I. -I/home/xavier/ricm/mul_projet/py2stm/shedskin_lib -I/home/xavier/ricm/mul_projet/py2stm/shedskin_lib/builtin /home/xavier/ricm/mul_projet/py2stm/shedskin_lib/builtin.cpp
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-g++ -o ./5.o -c -pedantic-errors -fno-rtti -std=gnu++0x -fno-threadsafe-statics -pipe -ffunction-sections -fdata-sections -fno-exceptions -mthumb -gdwarf-2 -pipe -DHSE_VALUE=8000000 -mcpu=cortex-m4 -DSTM32PLUS_F4 -O5 -I/home/xavier/project/stm32/stm32plus/lib/include -I/home/xavier/project/stm32/stm32plus/lib/include/stl -I/home/xavier/project/stm32/stm32plus/lib -I. -I/home/xavier/ricm/mul_projet/py2stm/shedskin_lib -I/home/xavier/ricm/mul_projet/py2stm/shedskin_lib/builtin /5.cpp
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-g++ -o LibraryHacks.o -c -pedantic-errors -fno-rtti -std=gnu++0x -fno-threadsafe-statics -pipe -ffunction-sections -fdata-sections -fno-exceptions -mthumb -gdwarf-2 -pipe -DHSE_VALUE=8000000 -mcpu=cortex-m4 -DSTM32PLUS_F4 -O5 -I/home/xavier/project/stm32/stm32plus/lib/include -I/home/xavier/project/stm32/stm32plus/lib/include/stl -I/home/xavier/project/stm32/stm32plus/lib -I. -I/LibraryHacks.cpp
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-gcc -o System.o -c -ffunction-sections -fdata-sections -fno-exceptions -mthumb -gdwarf-2 -pipe -DHSE_VALUE=8000000 -mcpu=cortex-m4 -DSTM32PLUS_F4 -O5 -I/home/xavier/project/stm32/stm32plus/lib/include -I/home/xavier/project/stm32/stm32plus/lib/include/stl -I. -I/System.c
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-as -mcpu=cortex-m4 -I. -o Startup.o Startup.asm
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-g++ -o ./5.elf -Xlinker -gc-sections -mthumb -g3 -gdwarf-2 -mcpu=cortex-m4 -Wl,-wrap,__aeabi_unwind_cpp_pr0 -Wl,-wrap,__aeabi_unwind_cpp_pr1 -Wl,-wrap,__aeabi_unwind_cpp_pr2 -Tstm32_flash.ld re.o builtin.o LibraryHacks.o Startup.o System.o 5.o /home/xavier/project/stm32/stm32plus/lib/build/small-f4-8000000/libstm32plus-master-f4-small.a
~/Lib/gcc-arm-none-eabi-4.7.2013q2/bin/arm-none-eabi-objcopy -O binary ./5.elf ./5.bin
```



Si l'utilisateur a choisi le mode burn, après la compilation, le fichier exécutable va être téléchargé dans la carte à l'aide de st-link. L'affichage est comme le suivant:

```
Burning...

/home/xavier/lib/stlink/st-flash write 1.bin 0x8000000

Flash page at addr: 0x08000000 erased
Flash page at addr: 0x08004000 erased
Flash page at addr: 0x08008000 erased
size: 32768
size: 8128
2014-03-09T09:52:46 INFO src/stlink-common.c: Loading device parameters...
2014-03-09T09:52:46 INFO src/stlink-common.c: Device connected is: F4 device, id 0x20006411
2014-03-09T09:52:46 INFO src/stlink-common.c: SRAM size: 0x30000 bytes (192 KiB), Flash: 0x100000 bytes (1024 KiB) in pages of 16384 bytes
2014-03-09T09:52:46 INFO src/stlink-common.c: Attempting to write 40896 (0x9fc0) bytes to stm32 address: 134217728 (0x8000000)
EraseFlash - Sector:0x0 Size:0x4000
EraseFlash - Sector:0x1 Size:0x4000
EraseFlash - Sector:0x2 Size:0x4000
2014-03-09T09:52:47 INFO src/stlink-common.c: Finished erasing 3 pages of 16384 (0x4000) bytes
2014-03-09T09:52:47 INFO src/stlink-common.c: Starting Flash write for F2/F4
2014-03-09T09:52:47 INFO src/stlink-common.c: Successfully loaded flash loader in sram
2014-03-09T09:52:48 INFO src/stlink-common.c: Starting verification of write complete
2014-03-09T09:52:49 INFO src/stlink-common.c: Flash written and verified! jolly good!
```

### 3. Modification du librairie

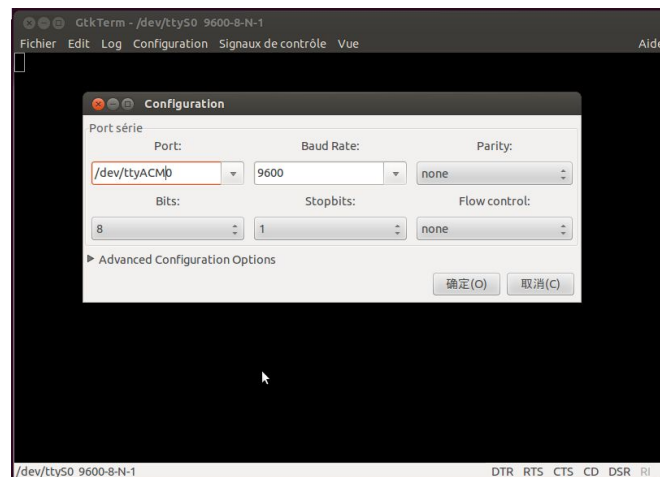
Pour les 3 librairies ce qu'on a utilisé, on a tous modifié. Pour STM Common Lib et STM32Plus, il n'y a pas de modification sur les fonctionnalités, on a juste ajouté des fichiers tête nécessaires pour éviter les erreurs de la compilation. Pour Shedskin Lib, on a supprimé les codes correspondant à l'exception, gc lib, pcre lib, des opérations sur stdin, stdout et quelques fonctions correspondant aux des fonctions en bas niveau (par exemple exit(), etc).

### 4. L'affichage du mode Debug

Quand on utilise le mode Debug pour faire un affichage à l'aide de la fonction printf, la fonction printf doit être dans une boucle infinie, sinon, il y a deux situations, soit on ne peut pas trouver la carte sur l'ordinateur, soit l'affichage est toujours 0. Si l'affichage est correct, quand on connecte la carte avec l'ordinateur, on peut le trouver dans le répertoire /dev, son nom est ttyACM0. Comme le suivant:

```
[dev]$ ls
alarm          dvdrw          loop6           ram0            rtc             tty1            tty26           tty42           tty59           ttyS15          ttyS31          vcsa
ashmem          encryptfs       loop7           ram1            rtc0            tty10           tty27           tty43           tty6            ttyS16          ttyS4           vcsa1
autofs          fb0             loop-control    ram10           sda             tty11           tty28           tty44           tty60           ttyS17          ttyS5           vcsa2
binder          fd              mapper          ram11           sda1            tty12           tty29           tty45           tty61           ttyS18          ttyS6           vcsa3
block           full            mcelog          ram12           sda2            tty13           tty3            tty46           tty62           ttyS19          ttyS7           vcsa4
bsg             fuse            mei             ram13           sda3            tty14           tty30           tty47           tty63           ttyS2            ttyS8           vcsa5
btrfs-control  hidraw0         mem             ram14           serial           tty15           tty31           tty48           tty7            ttyS20          ttyS9           vcsa6
bus             hidraw1         net             ram15           sg0             tty16           tty32           tty49           tty8            ttyS21          uinput          vga_arbiter
cdrom           hpet            network_latency ram2            sg1             tty17           tty33           tty5            tty9            ttyS22          urandom          vhost-net
cdwr            input           network_throughput ram3            shm             tty18           tty34           tty50           ttyACM0         ttyS23          usb              video0
char            kmsg           null            ram4            snapshot         tty19           tty35           tty51           ttyprntk        ttyS24          v4l              zero
console         log             nvram           ram5            snd              tty2            tty36           tty52           ttyS0            ttyS25          vcs              vcs
core            loop0           oldmem          ram6            sr0              tty20           tty37           tty53           ttyS1            ttyS26          vcs1              vcs2
cpu             loop1           port            ram7            stderr           tty21           tty38           tty54           ttyS10          ttyS27          vcs2              vcs3
cpu_dma_latency loop2           ppp             ram8            stdin            tty22           tty39           tty55           ttyS11          ttyS28          vcs3              vcs4
disk            loop3           psaux           ram9            stdout           tty23           tty4            tty56           ttyS12          ttyS29          vcs4              vcs5
dri             loop4           ptmx            random          tty              tty24           tty40           tty57           ttyS13          ttyS3            vcs5              vcs6
dvd             loop5           pts             rkill           tty0             tty25           tty41           tty58           ttyS14          ttyS30          vcs6
```

Comme ça on peut l'ouvrir à l'aide de l'outil gterm, comme le suivant:



Donc on peut voir l'affichage de la carte sur l'ordinateur.



## 5. La structure des fichiers

La structure des fichiers du projet est comme le suivant:

```
[projet]$ ls
LibraryHacks.cpp  print  py2stm.tcl  README  shedskin_lib  Startup.asm  stm32_flash.ld  System.c
```

Dans notre projet on a ajouté 2 librairie Shedskin Lib et STM Common Lib, ils sont dans les répertoires print et shedskin\_lib respectivement. Le fichier script py2stm.tcl est le fichier exécutable du projet. Les autres fichiers supporte des fonctions de bas niveau pour chaque projet généré.

## 4 Évaluation

Pour évaluer le projet, on a lancé notre projet sur tous les exemples fourni par le projet Shedskin. Les exemples sont 166 source fichiers du python, il contient des caractéristiques différents du langage python.



On a donc obtenu le résultat statistique comme le suivant:

Pour les 117 fichiers (.py) :

0,1,2,3,4,5,6,7,8,9,10,11,12,13,15,16,18,19,20,21,22,23,24,25,26,28,31,32,33,36,38,40,43,44,45,46,50,52,53,54,55,56,59,60,64,69,71,75,76,77,78,79,81,82,83,84,86,88,89,93,94,95,97,98,99,100,102,103,104,105,106,107,108,109,110,111,112,113,114,115,118,119,120,123,124,126,127,129,130,132,133,134,137,138,139,140,141,142,143,144,146,147,148,152,155,156,157,158,161,162,167,168,170,175,176,183,187

On a réussi pour les compiler.

Pour des 10 fichiers (.py):

122,131,135,150,159,160,163,164,172,173

Ils ont appelé correspond aux temps du système, il n'y a pas de système d'exploitation sur la carte, donc on peut pas les supporter. Les affichages d'erreur de la compilation sont sur 3 fonction de bas niveau `_times()` , `_times_r()` et `_gettimeofday()`.

Pour les 14 fichiers (.py):

29,30,37,125,128,136,151,153,154,165,166,171,190,195

Ils ont appelé des fonctions sur le iostream du fichier, donc pour le moment, on ne peut pas les supporter.

Pour les 13 fichiers (.py):

39,51,177,180,181,182,184,192,193,194,196,198,199

Shedskin ne supporte pas des caractéristiques utilisé dans ces fichiers, la traduction n'est pas fini, donc on n'obtient pas de résultat. L'affichage est comme le suivant:

```
[py2stm]$ ./py2stm.tcl -p test/39.py
Translating...
*** SHED SKIN Python-to-C++ Compiler 0.9.4 ***
Copyright 2005-2011 Mark Dufour; License GNU GPL version 3 (See LICENSE)

*ERROR* 39.py:2: cannot locate module: testdata
child process exited abnormally
```

Pour les 12 fichiers (.py):

169,174,179,185,186,188,189,191,197,200,201,202

On n'a pas pu supporté les caractéristiques utilisé dans notre projet.

Il y a quelques points ce qu'on n'a pas fait, on n'a pas supporté le gc (garbage collection), on a bien essayé le gc lib et le tiny gc lib, pour le gc lib, on n'a pas pu l'ajouté dans notre projet à cause de la différence entre g++ et arm-none-eabi-g++, arm-none-eabi-g++ ne supporte pas l'option `-l_nom_lib`.

On n'a pas supporté le printf traduit, tous les utilisations du printf doit être manuel. Parce que quand on utilise printf, on doit détecter le type de la variable ce qu'on veut afficher. Dans ce projet cette fonctionnalité sera très compliqué à cause des nouveaux type supporté par shedskin et la situation d'afficher les listes, etc.

# 5 Problème résolu

Dans ce projet, on a eu 2 problèmes sur le matériel, on explique les situations ce qu'on a rencontré

## 1. Redirection de USB

Dans notre projet, on a supporté le printf, c'est réalisé à l'aide de la redirection de USB. (Normalement, la direction est de l'ordinateur vers la carte, comme on fait le téléchargement. Quand on a besoin de printf, la direction est de la carte vers l'ordinateur, on a fait notre initialisation de USB dans le code.) Ça cause un problème, quand on refait le téléchargement, la carte ne peut pas être trouvée par l'outil ST-link. La solution est de lancer st-util en cliquant le bouton de RESET(noir) sur la carte. Comme le suivant:

```
[stlink]$ ./st-util
2014-03-09T10:10:35 INFO src/stlink-common.c: Loading device parameters....
2014-03-09T10:10:35 INFO src/stlink-common.c: Device connected is: F4 device, id 0x20006411
2014-03-09T10:10:35 INFO src/stlink-common.c: SRAM size: 0x30000 bytes (192 KiB), Flash: 0x100000 bytes (1024 KiB) in pages of 16
384 bytes
Chip ID is 00000413, Core ID is 2ba01477.
KARL - should read back as 0x03, not 60 02 00 00
init watchpoints
Listening at *:4242...
```

Après cette opération, on peut télécharger une autre fois.

## 2. La panne de la carte

Il y a de la situation que la carte tombe en panne à cause du code mauvais (par exemple les opérations sur la mémoire en bas niveau), st-util ne peut plus trouver la carte même si on fait les opérations avant. Il y a 2 solutions, on a essayé de cliquer le bouton RESET pendant le téléchargement. S'il ne marche pas, on peut quand même de reinitialise la carte en bas niveau à l'aide de l'outil arm-none-eabi-gdb.