

## Documentation du projet

Nom : -Mahmoud ABBOUD  
-Mohamed EL ALAMI

## Communication avec Domoticz.

Notre page web domoticz peut apparaitre sur l'adresse : 172.20.13.30 sur le port 8080.

1- création d'un capteur de température virtuel sur domoticz:

Nous avons créé un capteur de température virtuel sur domoticz et nous l'avons nommé temp0, il est affiché sur le Dashboard de domoticz.

2- donner une valeur à ce capteur via API JSON:

Nous avons utilisé la commande: curl

"http://172.20.13.30:8080/json.htm?type=devices&filter=temp&used=true" pour affiché les informations concernant les capteurs de température existants. Notre capteur a un idx= 5.

```
{
  "AddjMulti" : 1.0,
  "AddjMulti2" : 1.0,
  "AddjValue" : 0.0,
  "AddjValue2" : 0.0,
  "BatteryLevel" : 255,
  "CustomImage" : 0,
  "Data" : "20.0 C",
  "Description" : "",
  "Favorite" : 1,
  "HardwareID" : 3,
  "HardwareName" : "temp0",
  "HardwareType" : "Dummy (Does nothing, use for virtual switches only)",
  "HardwareTypeVal" : 15,
  "HaveTimeout" : false,
  "ID" : "14055",
  "LastUpdate" : "1970-01-01 00:23:25",
  "Name" : "temp0",
  "Notifications" : "false",
  "PlanID" : "0",
  "PlanIDs" : [ 0 ],
  "Protected" : false,
  "ShowNotifications" : true,
  "SignalLevel" : "-",
  "SubType" : "LaCrosse TX3",
  "Temp" : 20.0,
  "Timers" : "false",
  "Type" : "Temp",
  "TypeImg" : "temperature",
  "Unit" : 1,
  "Used" : 1,
  "XOffset" : "0",
  "YOffset" : "0",
  "idx" : "5"
},
```

ensuite, nous avons utilisé la commande: curl

"http://172.20.13.30:8080/json.htm?type=command&param=udevice&idx=5&nvalue=0&svalue=20" pour donner une valeur de 20 degré à notre capteur temp0.

```
u@fipadev ~ $ curl "http://172.20.13.30:8080/json.htm?type=command&param=udevice&idx=5&nvalue=0&svalue=20"
{
  "status" : "OK",
  "title" : "Update Device"
}
u@fipadev ~ $
```

3- un code C qui mettre à jour la valeur mesuré par le capteur tempCap (un autre capteur virtuel) automatiquement:

Nous avons créé un autre capteur de température nommé tempCap. Nous avons écrit un code C en utilisant 'libcurl' pour mettre à jour la valeur mesurée par tempCap.

Le fichier de ce code est nommé 'setTempCapValue.c'. Vous pouvez le trouver dans /scripts. il y a une partie de code commenté pour que l'utilisateur ne saisis pas les paramètres manuellement: ip ,port, idx, value. Donc par défaut nous avons programmé d'une manière que le code connaît notre IP et port et idx et nous avons donné une valeur de 1000, vous pouvez changer ce dernier. De plus, si vous voulez saisir les paramètres lors de compilation, il faut enlever les commentaires et il faut commenter les deux lignes: char url [] et char postdata []. Donc le scanf attend la saisie de l'utilisateur.

```
#include <stdio.h>
#include <curl/curl.h>
#include <string.h>

int main()
{
    /*char id[50];
    char port[50];
    char idx[50];
    char value[50];
    scanf("%s",id);
    scanf("%s",port);
    scanf("%s",idx);
    scanf("%s",value);
    char url[300];
    strcpy (url,"http://");
    strcat (url,id);
    strcat (url,":");
    strcat (url,port);
    strcat (url,"/");
    strcat (url,"json.htm?type=command&param=udevice&idx=");
    strcat (url,idx);
    printf("\n%s",url);
    char postdata[300];
    strcpy (postdata,"svalue=");
    strcat(postdata,value);*/
    char url[]="http://172.20.13.30:8080//json.htm?type=command&param=udevice&idx=7&nvalue=0";
    char postdata[]="svalue=1000";
    printf("\n%s",url);

    CURL *curl;
    CURLcode res;
```

La partie commentée.

Cross-compiler ce programme :

Nous avons utilisé 'official cross compiling toolchain from Github' *pour cross-compiler le code*

Installation de la chaine de github :

Commande : git clone git://github.com/raspberrypi/tools.git

Après la décompression du dossier cloné nous l'avons mis dans home/u/rpi/.

Donc dans cette destination nous avons un dossier nommé arm-bcm2708/ qui contient 4 tool chain.

Et comme nous utilisons une distribution linux 64 bit nous avons besoin du quatrième tool chain :

gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64. L'étape suivante consiste à ajouter le dossier contenant les fichiers binaires de tool chain à la path de l'environnement de variable dans linux.

Pour ce faire, nous avons édité le fichier ~/.bashrc on rajouter cette ligne :export

```
PATH=$PATH:$home/u/rpi/tools-master/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin
```

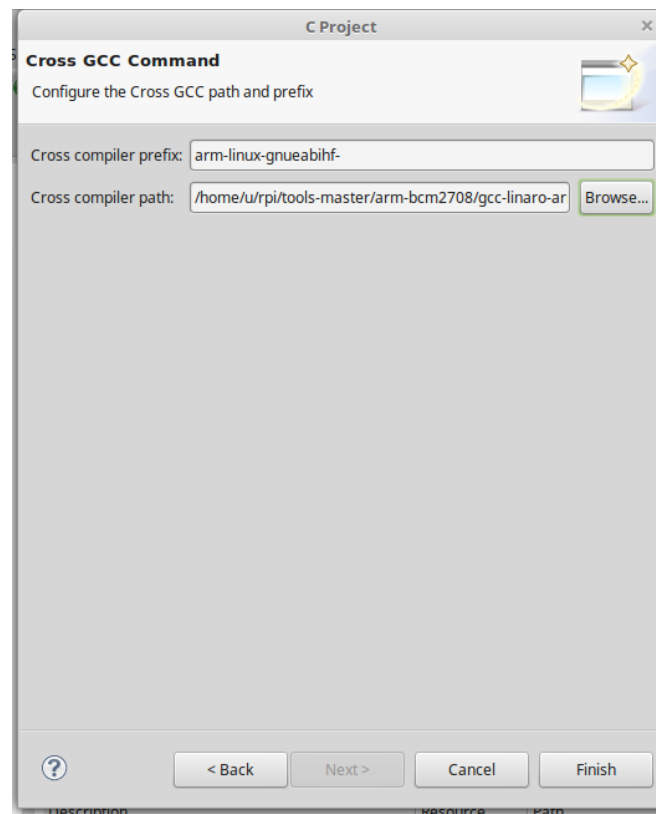
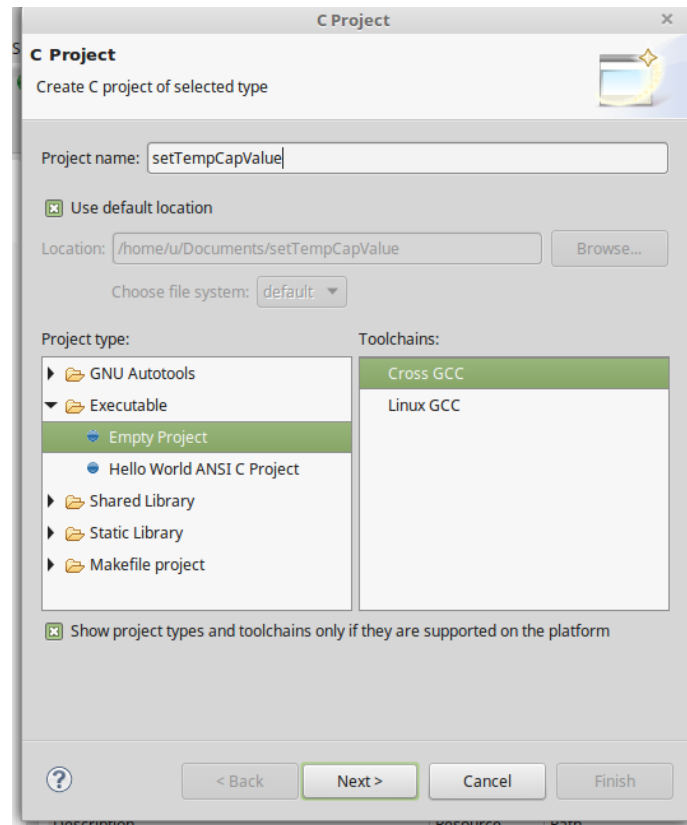
```
if [ -x /usr/bin/mint-fortune ]; then
    /usr/bin/mint-fortune
fi
export PATH=$PATH:$home/u/rpi/tools-master/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin
```

```
u@fipadev ~ $ vi .bashrc
u@fipadev ~ $ source .bashrc
u@fipadev ~ $ arm-linux-gnueabi-hf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabi-hf-gcc
COLLECT_LTO_WRAPPER=/home/u/rpi/tools-master/arm-bcm2708/gcc-linaro-arm-linux-gnueabi-hf-raspbian-x64/bin/./libexec/gcc/arm-linux-gnueabi-hf/4.8.3/lto-wrapper
Target: arm-linux-gnueabi-hf
Configured with: /home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/.build/src/gcc-linaro-4.8-2014.03/configure --build=x86_64-build_unknown-linux-gnu --host=x86_64-build_unknown-linux-gnu --target=arm-linux-gnueabi-hf --prefix=/home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/install --with-sysroot=/home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/install/arm-linux-gnueabi-hf/libc --enable-languages=c,c++,fortran --disable-multilib --enable-multiarch --with-arch=armv6 --with-tune=arm1176jz-s --with-fpu=vfp --with-float=hard --with-pkgversion='crosstool-NG linaro-1.13.1+bzr2650 - Linaro GCC 2014.03' --with-bugurl=https://bugs.launchpad.net/gcc-linaro --enable-__cxa_atexit --enable-libmudflap --enable-libgomp --enable-libssp --with-gmp=/home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/.build/arm-linux-gnueabi-hf/build/static --with-mpfr=/home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/.build/arm-linux-gnueabi-hf/build/static --with-mpc=/home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/.build/arm-linux-gnueabi-hf/build/static --with-isl=/home/zhehe01/work/bzr/pi-build/builds/arm-linux-gnueabi-hf-raspbian-linux/.build/arm-linux-gnueabi-hf/build/static
```

Maintenant nous pouvons accéder le tool chain de l'invite de commande

### téléchargement et configuration d'Eclipse .

Pour ne pas entrer en Détail, en bref nous avons téléchargé eclipse ide for c/c++ developper en créant un projet c et en configurant le path du compiler (vers notre tool chain compiler).



Configuration de path de tool chain

```

fipadev Debug # ls
makefile      setTempCapValue  setTempCapValue.o  subdir.mk
objects.mk    setTempCapValue.d  sources.mk
fipadev Debug # ls -al
total 100
drwxrwxr-x 2 u u  4096 Jan 31 12:04 .
drwxrwxr-x 4 u u  4096 Jan 31 11:53 ..
-rw-rw-r-- 1 u u  1014 Jan 31 12:04 makefile
-rw-rw-r-- 1 u u   231 Jan 31 12:04 objects.mk
-rwxrwxr-x 1 u u 30353 Jan 31 12:04 setTempCapValue
-rw-rw-r-- 1 u u   40 Jan 31 12:03 setTempCapValue.d
-rw-rw-r-- 1 u u 37260 Jan 31 12:03 setTempCapValue.o
-rw-rw-r-- 1 u u   390 Jan 31 12:04 sources.mk
-rw-rw-r-- 1 u u   701 Jan 31 12:04 subdir.mk
fipadev Debug # ./setTempCapValue
bash: ./setTempCapValue: cannot execute binary file: Exec format error
fipadev Debug # file setTempCapValue
setTempCapValue: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamic
ally linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.26, BuildID
[sha1]=7d002f2547768daf69b88cdb6ba8a91eeeb8120b, not stripped
fipadev Debug #

```

La deuxième étape consiste à écrire le code source du projet et le compiler pour la rpi3.

Nous avons fait in build project pour avoir le fichier binaire du code.

```

12:04:41 **** Incremental Build of configuration Debug for project setTempCapValue ****
make all
Building target: setTempCapValue
Invoking: Cross GCC Linker
arm-linux-gnueabi-gcc -o "setTempCapValue" ./setTempCapValue.o
Finished building target: setTempCapValue

12:04:41 Build Finished (took 267ms)

```

Et voilà le binaire exécutable :

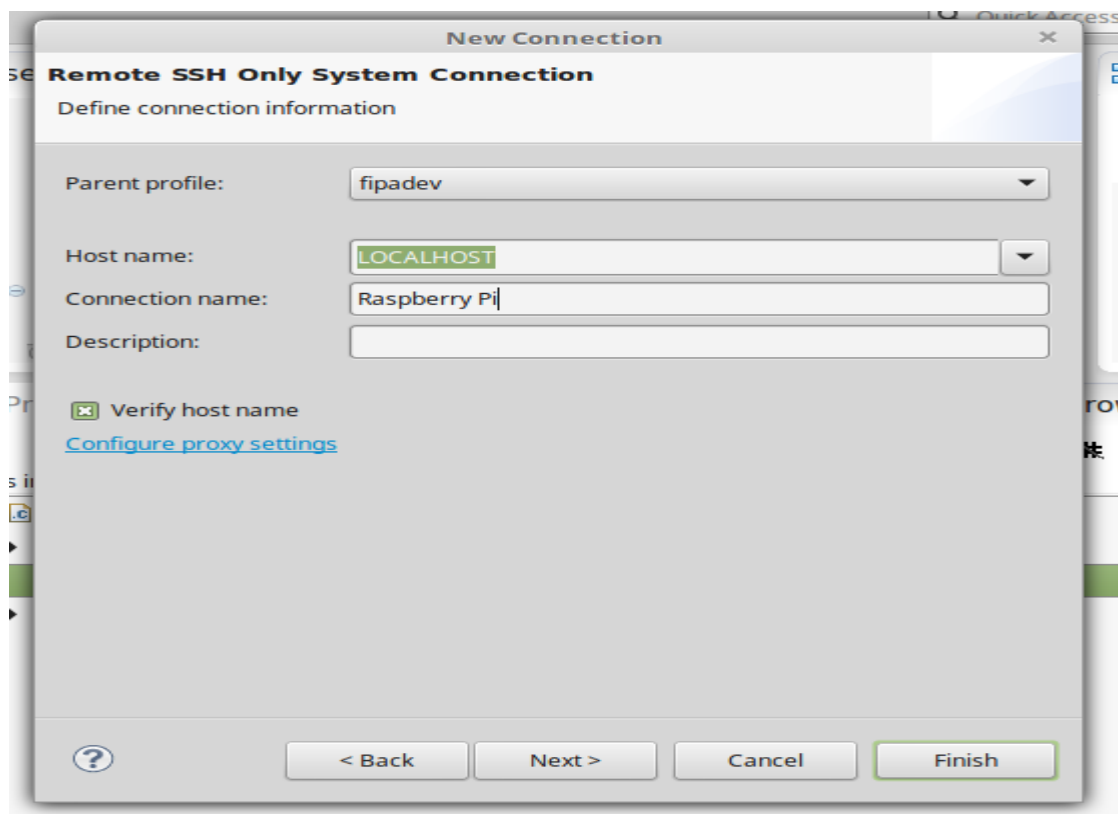
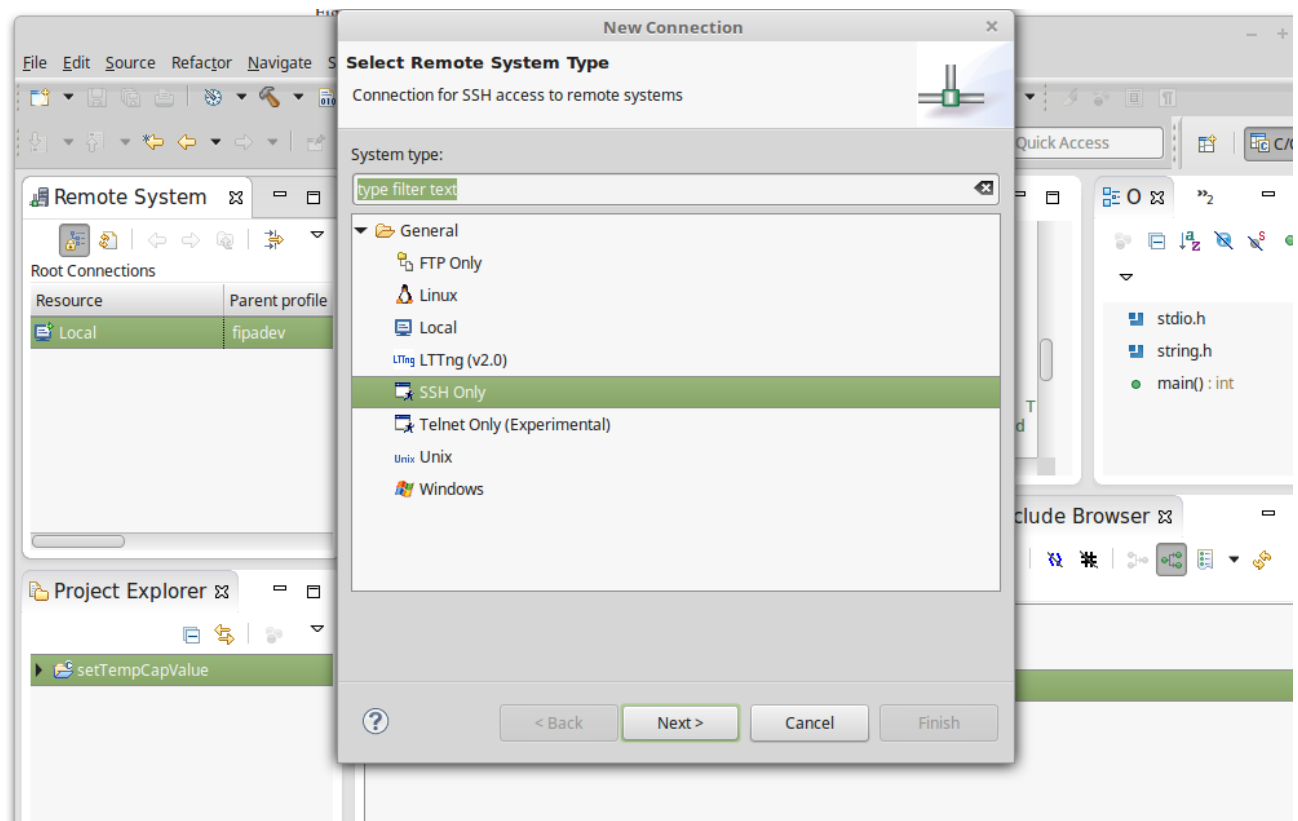
```

RemoteSystemsTempFiles  setTempCapValue

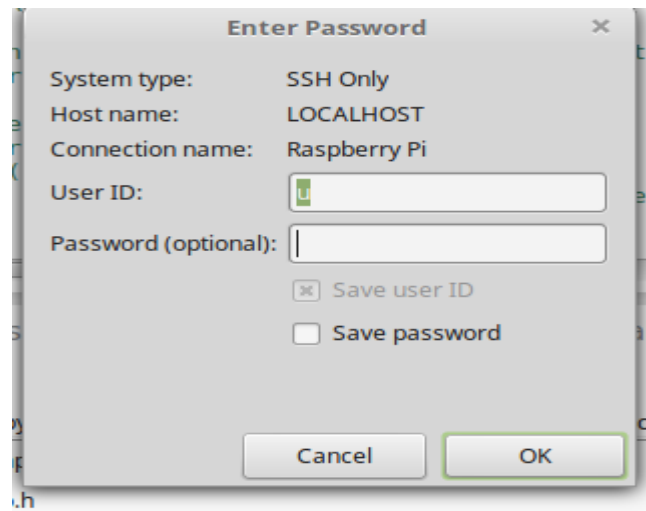
```

Maintenant il suffit juste d'employer les fichiers binaires dans la carte rpi3.

Sur eclipse, nous avons créé une connexion via Remote Systems.

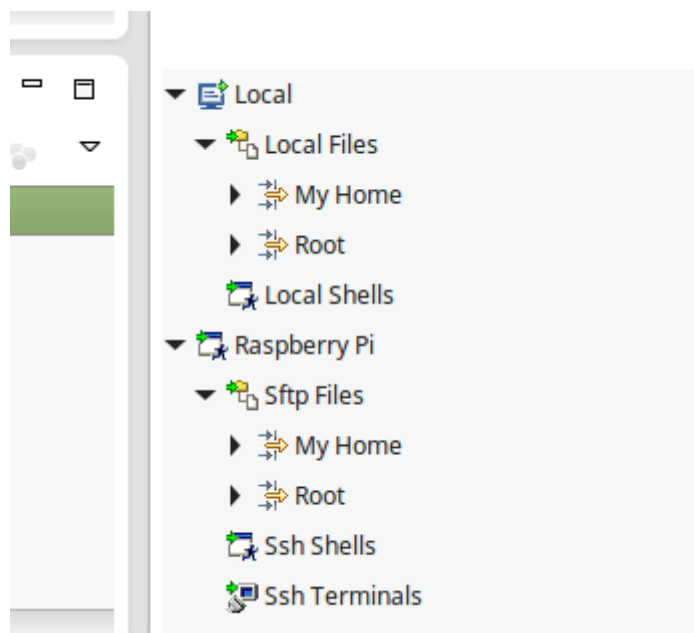


Se connecter sur rpi :



On obtient maintenant les systèmes de fichiers de machine local et de rpi3 :

Un simple copier-coller de local vers la rpi (on copie le fichier binaire de notre code):



Maintenant on ouvre le terminal qui nous connecte avec la carte via ssh :



```

u@fipadev ~ $ ./setTempCapValue
-bash: ./setTempCapValue: cannot execute binary file: Exec format error
u@fipadev ~ $ file setTempCapValue
setTempCapValue: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.26, BuildID[sha1]=7d002f2547768daf69b88cdb6ba8a91eeeb8120b, not stripped
u@fipadev ~ $ █

```

On change la permission du fichier binaire pour qu'il soit exécutable :

```

u@fipadev ~ $ chmod +x setTempCapValue
u@fipadev ~ $ █

```

Et enfin on peut exécuter le fichier binaire sur la carte.


4- création d'un Virtual switch :

Implémentation de deux scripts :  
Destination : scripts/



Name:

Switch Type:

Switch Icon:  **Light/Switch**  
A Lamp or Switch

On Delay:  (Seconds) 0 = Disabled

Off Delay:  (Seconds) 0 = Disabled

On Action:  (Should s

Off Action:  (Should s

Protected: ☐

Description:

```
echo "ACTION OFF $(date)" >> /tmp/switch0.log
```

Contenu de switch0\_off.sh.


5- code C pour allumer/éteindre un led via Virtual switch :

Nous avons fait les codes serveurs en python et les code client en bash. Le switch 1 exécute deux fichiers bash : clientOn.sh et clientOff.sh selon l'état de switch 1. dans chacun de ces deux code bash, il y a un appel aux codes serveurs écrits en python (onAction.py et offAction.py) pour allumer/éteindre le led connecté sur le Pen 37/ gpio26. Ces deux derniers utilisent GPIO.OUT pour allumer/éteindre le led.

Les paths de ces codes (python : /scripts/python/)

Name:

Switch Type:

Switch Icon:  **Light/Switch**  
A Lamp or Switch

On Delay:  (Seconds) 0 = Disabled

Off Delay:  (Seconds) 0 = Disabled

On Action:  (Should start w

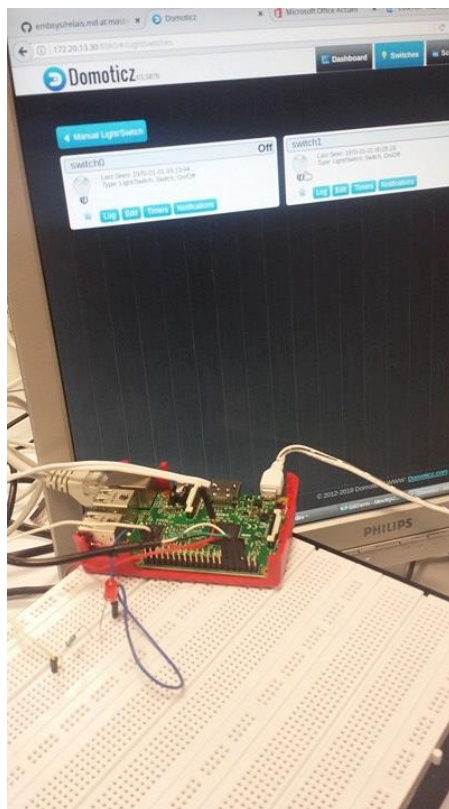
Off Action:  (Should start w

Protected: ☐

Description:

Slave Devices:

Vous pouvez vérifier le bon fonctionnement en connectant un led au gpio26 et en cliquant sur le switch 1 de domoticz.



Les codes fonctionnent sur la carte donc ce n'est pas la peine de cross\_compiler les codes. Pour exécuter les codes serveurs au moment de démarrage de la carte, nous avons écrit un code bash dans /etc/init.d/, il s'appelle serverSwitchBoot.sh. Il sert à exécuter les deux programmes serveurs pendant le démarrage de la carte. Il est exécutable grâce à la commande chmod 755 serverSwitchBoot.sh. Nous l'avons testé grâce à la commande : service serverSwitchBoot.sh start.

Activer le lancement du script au démarrage du système avec la commande update-rc.d : update-rc.d serverSwitchBoot.sh enable.

```
./serverSwitchBoot.sh start
/opt/domoticz/scripts/python/onAction.py:3: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarnings(False) to disable warnings.
  GPIO.setup(37,GPIO.OUT)
/opt/domoticz/scripts/python/offAction.py:3: RuntimeWarning: This channel is already in use, continuing anyway. Use GPIO.setwarning
s(False) to disable warnings.
  GPIO.setup(37,GPIO.OUT)
```

Il a lancé les deux programmes serveurs.

6- allumer un led connecté au gpio17 grâce à sysfs :

Nous avons créé un script bash dans le répertoire scripts/ de la carte. Il est nommé sysfsGpio17.sh. Il sert à accéder sys/class/gpio pour exporter le gpio17 puis il y rentre pour faire la direction en output et mettre la valeur à 1. Vous pouvez le tester en faisant ./sysfsGpio17.sh.

```
#!/bin/bash
echo 17 >/sys/class/gpio/export
echo out >/sys/class/gpio/gpio17/direction
echo 1 > value
echo 17 > /sys/class/gpio/unexport
~
```

7- un code python pour faire l'étape précédente :

Dans le répertoire scripts/ de la carte, nous avons créé un code python (python\_code.py) qui sert à allumer/éteindre le led connecté au gpio17 pour un certain temps. Vous pouvez le tester (python python\_code.py).

8- PWM

Dans cette partie nous avons copié le code de github mais il n'a pas marché. Dans scripts (pwm.py)

9- Relais.

Pour tester l'allumage du led à travers le relais nous avons appliqué les deux dernières étapes sauf que le câblage se diffère et le numéro de gpio également. Et nous avons réussi à allumer le led.

## 10- Code pour relais.

Nous avons écrit un code C qui communique avec les gpio via l'application (sysfs).

Il est nommé sysfsRelais.c. Il se trouve dans scripts/. Il fait mettre le gpio4 en output et mettre à jour sa valeur 0/1. Un code client en bash dans script/ il est nommé clientRelais.sh, il sert à exécuter le code serveur quand le switch 2 de domoticz change d'état. Mais il ne marche pas parce que nous n'avons pas arrivé à le cross-compiler dans la carte.

NB : le code qui sert à mettre à jour le capteur de température n'est pas cross-compileé dans la carte du fait que la librairie libcurl présente un problème d'importation dans eclipse, donc nous n'avons pas arrivé à faire le build binaire. Mais ce que nous avons écrit dans le début de ce rapport c'est la cross-compilation que nous avons réussi à faire pour le code commenté (pas de problème d'importation donc un code exécutable). Et le but était de vous montrer que nous avons trouvé un autre moyen que buildroot parce que nous n'avons pas réussi de faire ça avec buildroot.