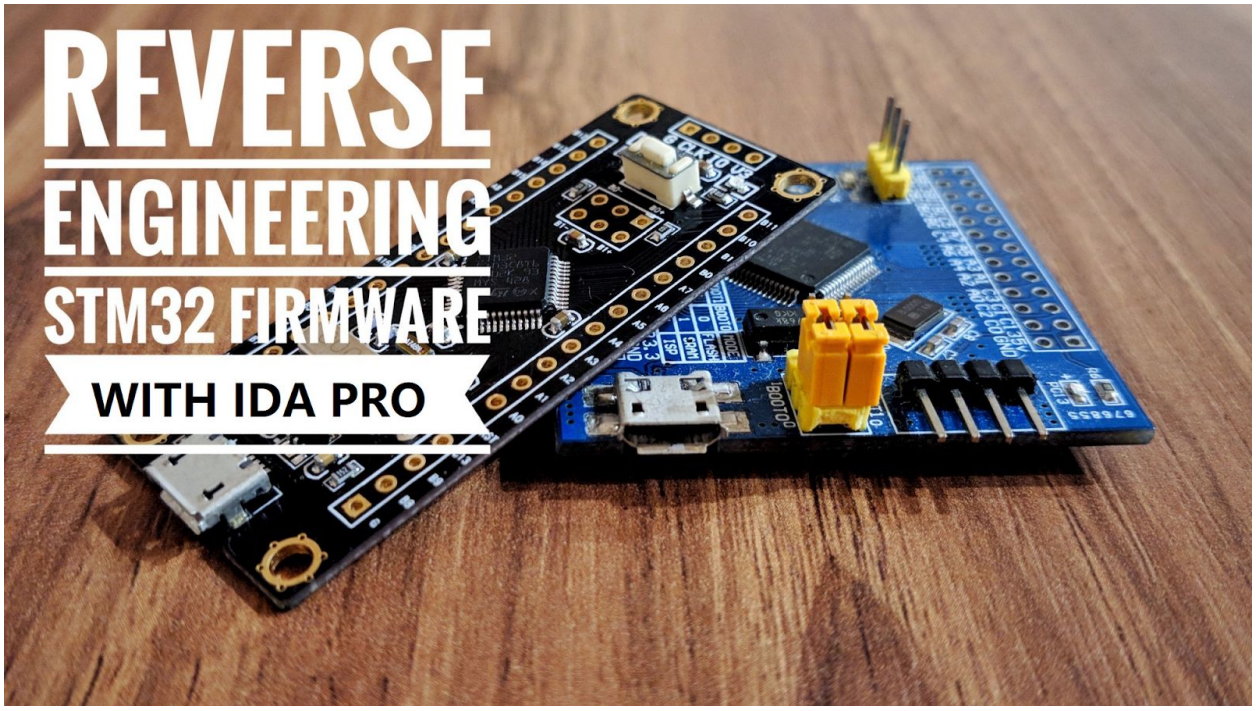


Project Hardware Reverse Engineering

STM32 Firmware

Membres :

- Kevin Minacori
- Yanis Alim
- Mohamed Dhia Layadi
- Lotfi Derri
- Arezki Mehaddi

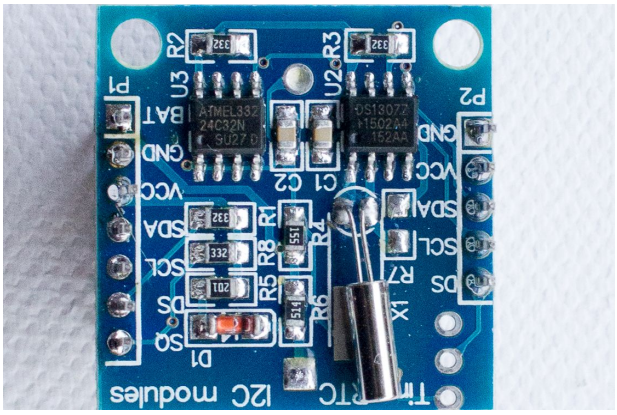
Responsable :

- Mathieu Renard

Analyse du circuit

Les références

Après analyse de la photo du composant, plusieurs déduction se font :



Nous sommes en présence d'un Arduino Tiny I2C Real Time Clock avec deux composants majeurs :

- Une puce ATMEL332 24C32N
- Une puce DS1307

On observe également un oscillateur, plusieurs résistances (8 au total), une diode, ainsi que deux condensateurs. De plus, trois sorties sont présentent juste à côté de l'oscillateur.

Enfin, on observe plusieurs pins : P2 : (DS, SCL, SDA, VCC, GND) d'un côté, et P1 : (SQ, DS, SCL, SDA, VCC, GND, BAT) de l'autre.

Les datasheets

On retrouve ici un datasheet global du module :

https://pdf.direnc.net/upload/tinyrtc-i2c-modul-datasheet.pdf

Après analyse, on observe les composants suivant :

Une mémoire morte : EEPROM AT24C32, avec un datasheet :

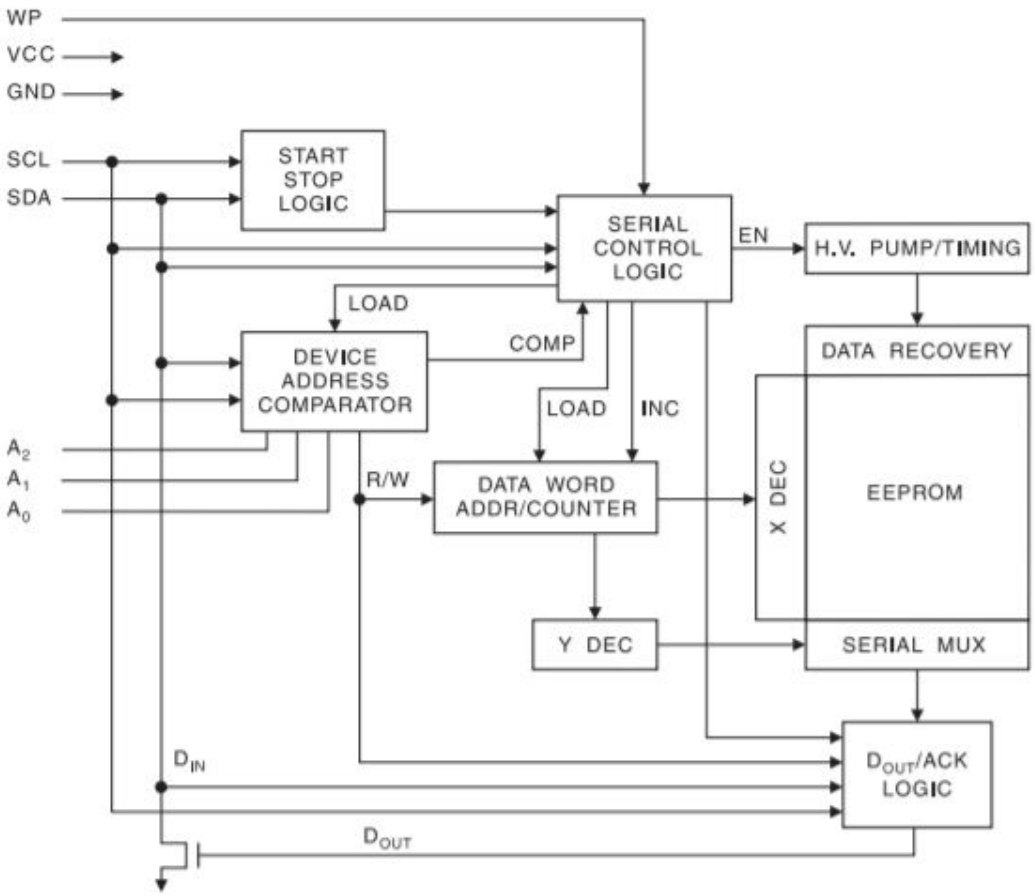
https://ww1.microchip.com/downloads/en/DeviceDoc/doc0336.pdf

Une horloge : DS1307 real-time clock IC, avec un datasheet :

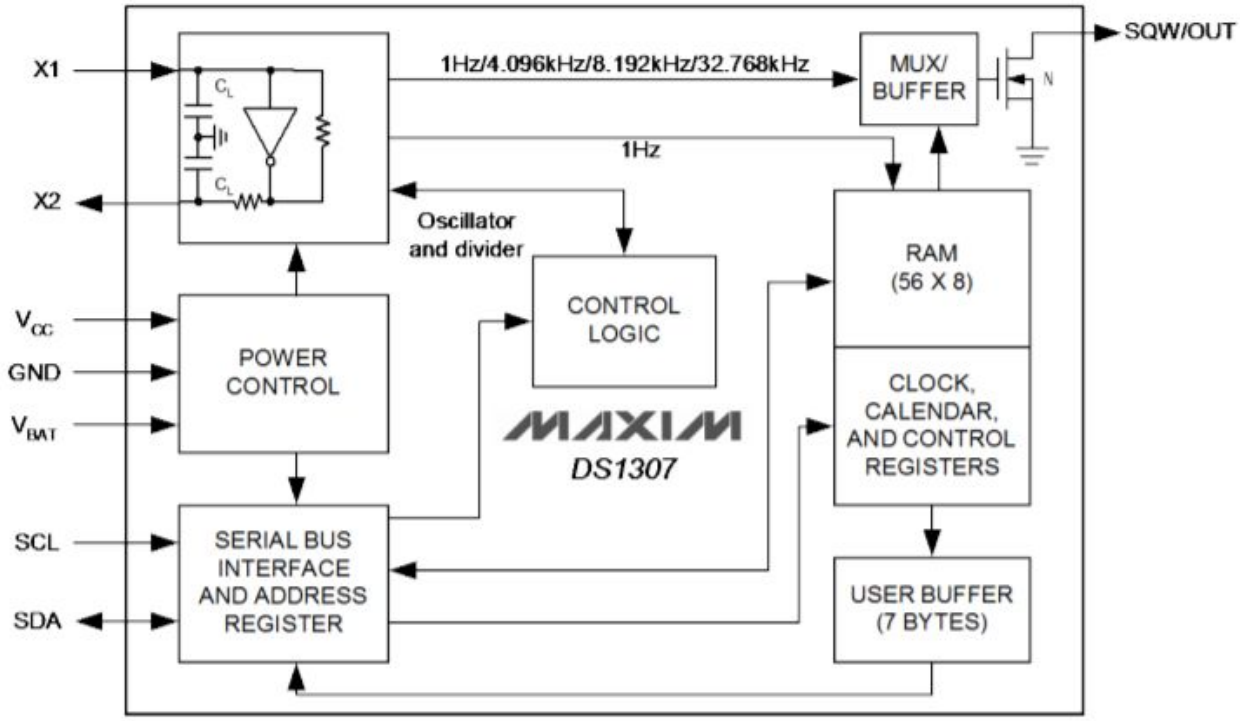
https://datasheets.maximintegrated.com/en/ds/DS1307.pdf

Rôle des composants

Le rôle de la mémoire morte : EEPROM AT24C32 est de garder en mémoire une petite quantité de données tout en nécessitant très peu de courant. Dans notre cas, elle a pour but de garder en mémoire le code de fonctionnement du module, et les fonctions essentielles pour le calcul du temps (années bissextiles, calcul du nombre de jour en février, nombre de jour dans un mois, nombre de mois dans une année, etc..). La mémoire est organisée en 256 pages de 32 bits chacune pour la version AT24C32. Elle est dotée d'un contrôleur logique pour les opérations, d'un WP (Write Protect), pour éviter des problèmes qui peuvent se manifester lors d'une écriture et différents modes d'opération, d'adressages d'écriture et de lecture, reliée à différents ports pour les E/S.



Le rôle de l'horloge : DS1307 real-time clock IC est d'apporter une valeur du temps réel, et de gérer les heures ainsi que les jours, mois et années en prenant compte du nombre de jour par mois, des années bisectiles. Elle contient une RAM, pour gérer les informations de calcul en temps réel, d'un bus et de registres pour les E/S des données par SDA, et pour les entrées de voltage de la CLOCK par SCL, qui est utilisé pour synchroniser le mouvement des données sur le bus, d'un buffer, d'un oscillateur et d'un contrôleur logique pour les opérations.



Comment sont connectés les composants

Les composant sont soudés à la carte magnétique. Des deux côtés, les composants sont connectés sur les pins : SDA pour le serial data INPUT/OUTPUT, SLC pour le serial clock INPUT. Ils sont également connectés à la masse via le pin GND. Les trois sorties à côté de l'oscillateur sont connecté au pin DS, correspondant à la mesure de la température, grâce à un capteur possiblement connectable via les trois sorties. Le pin VCC est présent pour fournir une charge de +5V au module et charger la pile. Enfin, la pile, est une solution alternative en cas d'une mise hors-tension de l'appareil à laquelle elle est connectée, elle est reliée au pin BAT qui permet de surveiller le voltage de celle-ci. Le dernier pin SQ n'est généralement pas utilisé, car elle est valable que dans le cas ou elle doit renvoyer une valeur (0 ou 1), quand SQW/OUT = 1 (SQuare Wide OUTPUT qui correspond à la fréquence que l'oscillateur envoi et peut avoir 4 valeurs possibles), SQWE = 0 (SQuare Wide Enable,

permet de sélectionner le mode de fonctionnement du SQW/OUT) , et que les valeurs des registres RS1 et RS0 sont nulles, si OUT=0, la sortie est au niveau bas, et si OUT=1, la sortie est au niveau haut. Les conditions pour avoir ce mode de fonctionnement sont très rare et c'est ce qui fait que SQ n'est presque jamais utilisé. Voici un tableau qui résume ceci.

RS1	RS0	SQW/OUT OUTPUT	SQWE	OUT
0	0	1Hz	1	X
0	1	4.096kHz	1	X
1	0	8.192kHz	1	X
1	1	32.768kHz	1	X
X	X	0	0	0
X	X	1	0	1

Les résistances, les condensateurs et la diode servent à réguler l'énergie du courant qui circule entre les différents composants.

Type de BUS

Le type de BUS utilisé ici est le I2C (Inter-Integrated Circuit).

Type de Protocol

Le type de protocole utilisé ici est le protocol I2C.

Analyse du firmware

Réaliser la rétro-ingénierie du circuit imprimé (PCB)

Identifier les composants

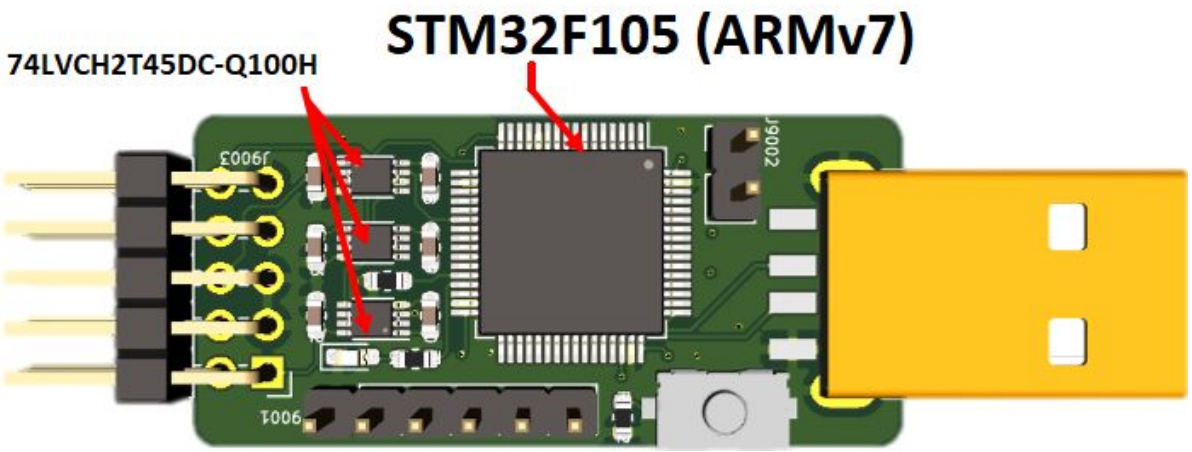


Schéma représentant le rôle des composants de la carte.

- 74LVCH2T45DC-Q100H : emetteur, récepteur bus
- STM32F105 : microcontrôleur ARM

Trouver les documentations techniques (datasheet) et les manuel (référence manual).

STM32F105

Manuel de référence :
https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf

Fiche techniques :
https://www.st.com/en/microcontrollers-microprocessors/stm32f105-107.html#

Manuel de programmation :
https://www.st.com/content/ccc/resource/technical/document/programming_manual/10/98/e8/d4/2b/51/4b/f5/CD00283419.pdf/files/CD00283419.pdf/jcr:content/translations/en.CD00283419.pdf

74LVCH2T45DC-Q100H

Fiche technique :
https://datasheet.octopart.com/74LVCH2T45DC-Q100H-Nexperia-datasheet-87842627.pdf

Identifier le SoC principal et donner sa référence ainsi que ses caractéristiques essentielles

Le SoC principal correspond au composant STM32F105 dont l'architecture est décrit par le schéma suivant :

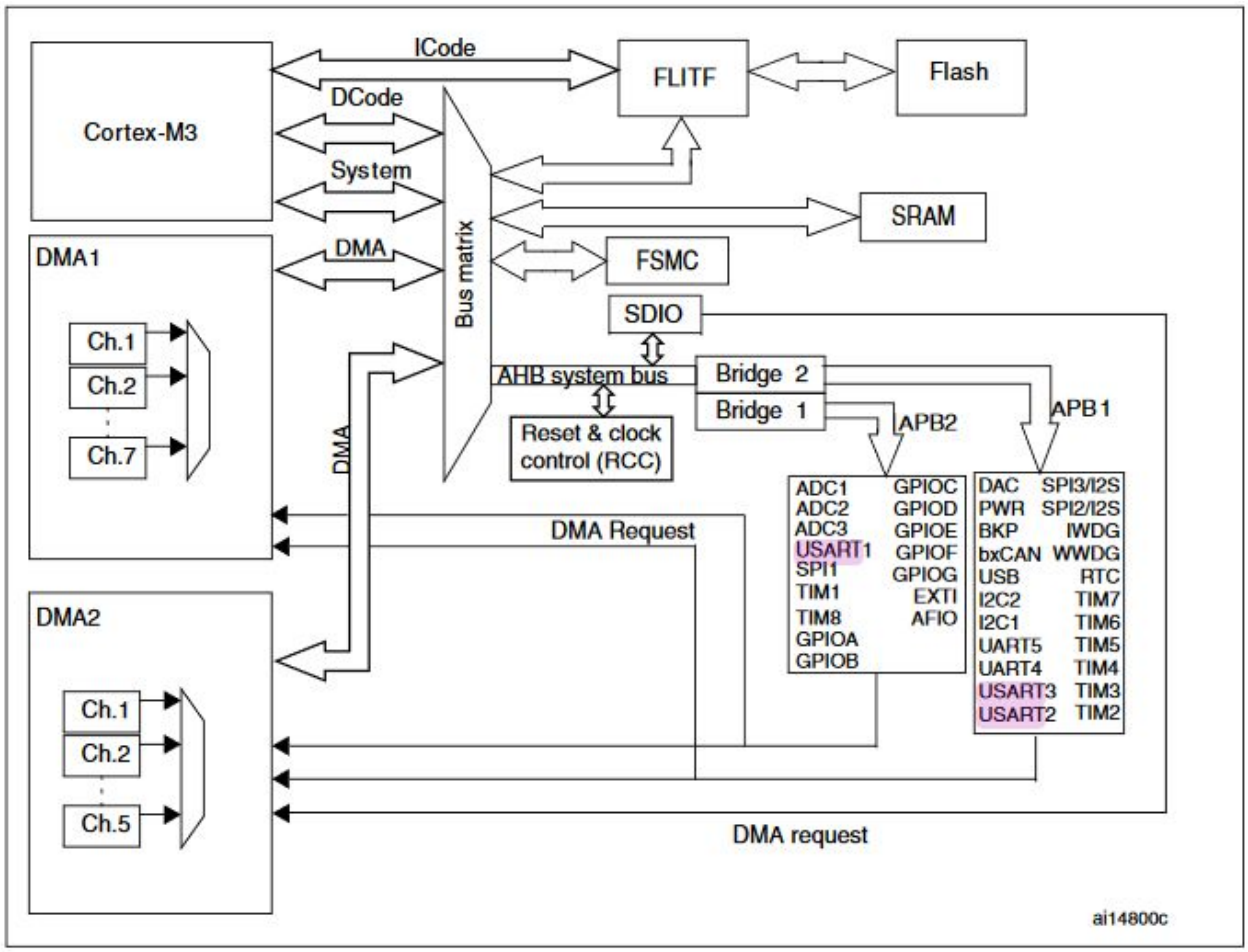
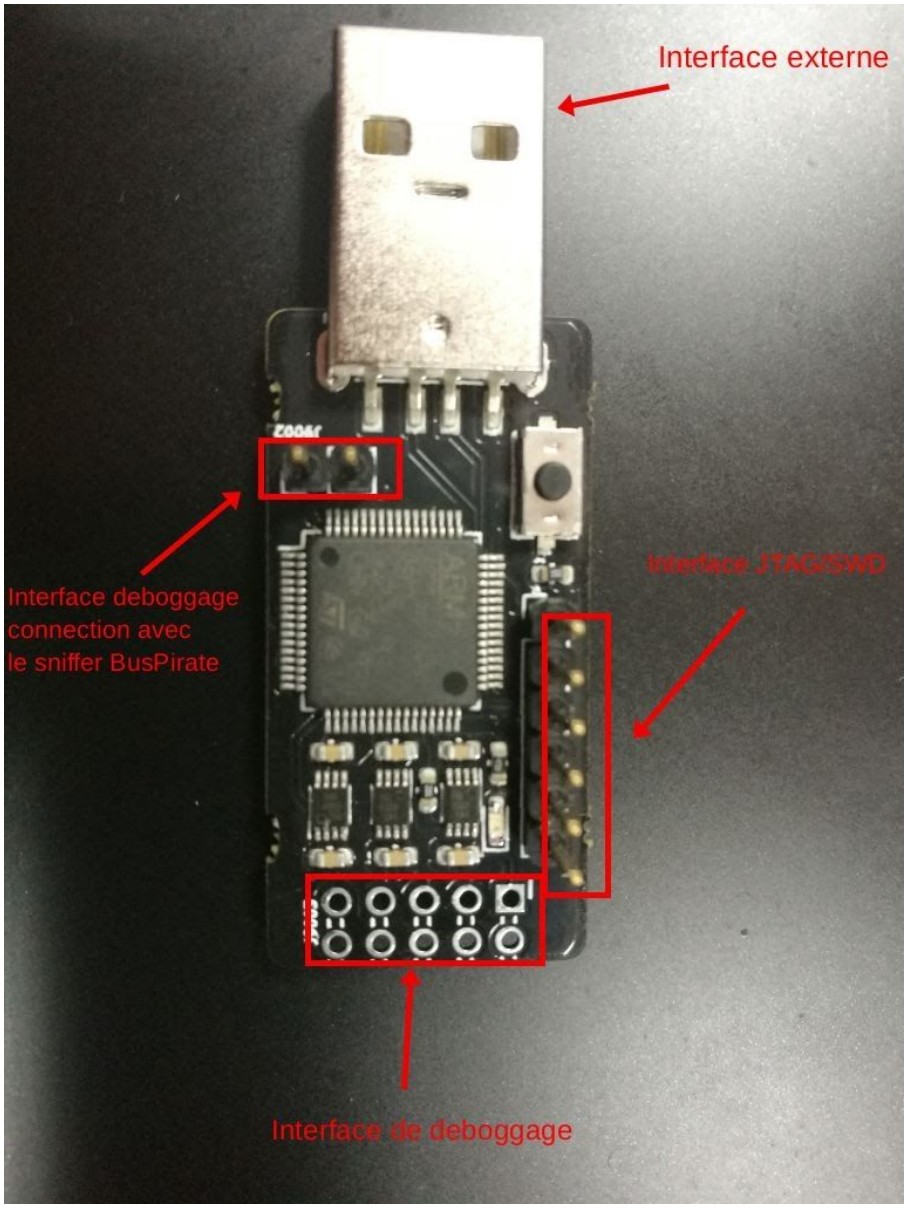


Schéma de l'architecture système de la puce

Les caractéristiques essentielles sont :

- Processeur cortex-m3
- Mémoire de type flash
- Mémoire de type SRAM
- Pin USART
- GPIO

Identifier les interfaces



Le JTAG/SWD est il actif ?

Le JTAG est activé.

```
Yan1x0s@1337:~/0x00/2019/Hardware-Re$ openocd -f stm32f4disco1.cfg
Open On-Chip Debugger 0.10.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
Info : The selected transport took over low-level target control. The results might differ compared to plain JTAG/SWD
adapter speed: 1000 kHz
adapter_nsrst_delay: 100
none separate
srst_only separate srst_nogate srst_open_drain connect_deassert_srst
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : Unable to match requested speed 1000 kHz, using 950 kHz
Info : clock speed 950 kHz
Info : STLINK v2 JTAG v25 API v2 SWIM v14 VID 0x0483 PID 0x374B
Info : using stlink api v2
Info : Target voltage: 2.891838
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
Info : accepting 'gdb' connection on tcp/3333
Info : device id = 0x10016418
Info : flash size = 128kbytes
undefined debug reason 7 - target needs reset
undefined debug reason 7 - target needs reset
```

Connection au port jtag à l'aide d'openocd et gdb

Après avoir lancé OpenOCD avec :

openocd -f stm32f4disco1.cfg

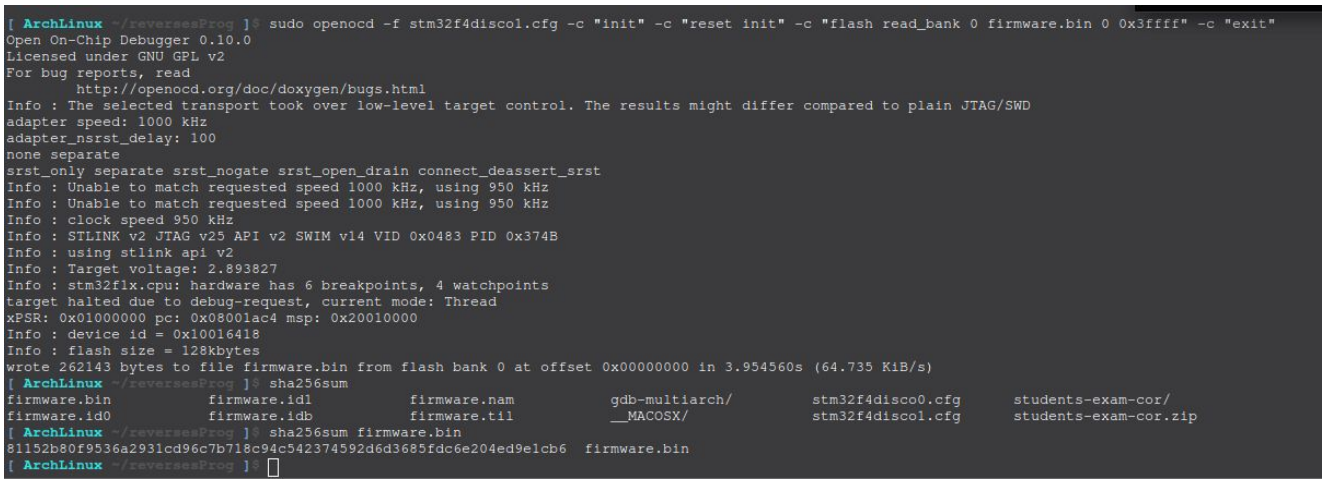
On se connecte au JTAG avec gdb et on spécifie que la hôte et le port où attaché gdb:

```
root@1337:/home/Yan1x0s/0x00/2019/Hardware-Re# gdb-multiarch
GNU gdb (Debian 8.3.1-1) 8.3.1
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word".
(gdb) set arch arm
The target architecture is assumed to be arm
(gdb) target extended-remote localhost:3333
Remote debugging using localhost:3333
warning: No executable has been specified and target does not support
determining executable automatically.  Try using the "file" command.
0x00000000 in ?? ()
(gdb) x/20wx
Argument required (starting display address).
(gdb) x/20wx 0
0x0:    0x20010000      0x08001ac5      0x08001ac3      0x08001ac1
0x10:   0x08001ac1      0x08001ac1      0x08001ac1      0x00000000
0x20:   0x00000000      0x00000000      0x00000000      0x08001ac3
0x30:   0x08001ac3      0x00000000      0x08001ac3      0x08000c89
0x40:   0x08001ac1      0x08001ac1      0x08001ac1      0x08001ac1
(gdb) _
```

Extraction du firmware

Dump du firmware dans un fichier binaire



Sh256 du binaire

81152b80f9536a2931cd96c7b718c94c542374592d6d3685fdc6e204ed9e1cb6 firmware.bin

Analyse du binaire (Statique / dynamique)

Identification du point d'entrée

Nous savons que le boot à lieu depuis la flash. D'après la documentation[1], section 3.4 Boot configuration, le contenu de la flash est mappé à l'adresse 0x08000000. Le point d'entrée correspond au reset handler, qui est à la seconde place dans la table[2]. Le point d'entrée est donc 0x08000000 + 0x4 = 0x08000004.

Identification des tables des interruptions

D'après la documentation, la table d'interruption est mappé à l'adresse 0x08000000.

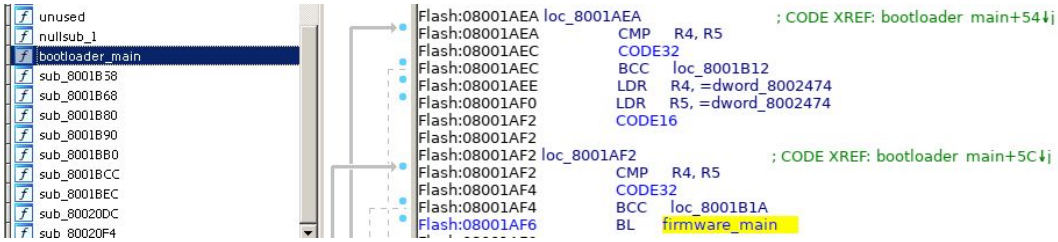
Identification de l'adresse de la pile

[1] https://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf

[2] Table 61,

Décrire le fonctionnement global du binaire

Une fois tout est prêt, on essaye de trouver le main et pour cela on va vers le reset_handler et on sait bien qu'après avoir initialiser les dépendances il y aura un appel au main à la fin.



Dans le main :

```
39      do
40      {
41        if ( v3 <= cpt )
42          break;
43        v6 = *((unsigned __int8 *)off_8000200 + cpt++);
44        if ( v6 != v5 )
45          goto LABEL_14;
46        v5 = *(unsigned __int8 *)(cpt + v1);
47      }
48      while ( *(_BYTE *)(cpt + v1) );
49      if ( cpt == 21 )
50        sub_80011A4(*(_DWORD *)off_8000204, 3, ptr_welldone, 11);
51      else
52  LABEL_14:
53        sub_80011A4(*(_DWORD *)off_8000204, 3, ptr_looser, 9);
54      v7 = *(_DWORD *)off_8000204;
55      *(_WORD *)ptr_GPIOB_BRR = 4;
56      sub_80011A4(v7, 3, ptr_password[0], 11);
57      v8 = off_80001F0;
58      *off_80001FC = 0;
59      *v8 = v12;
```

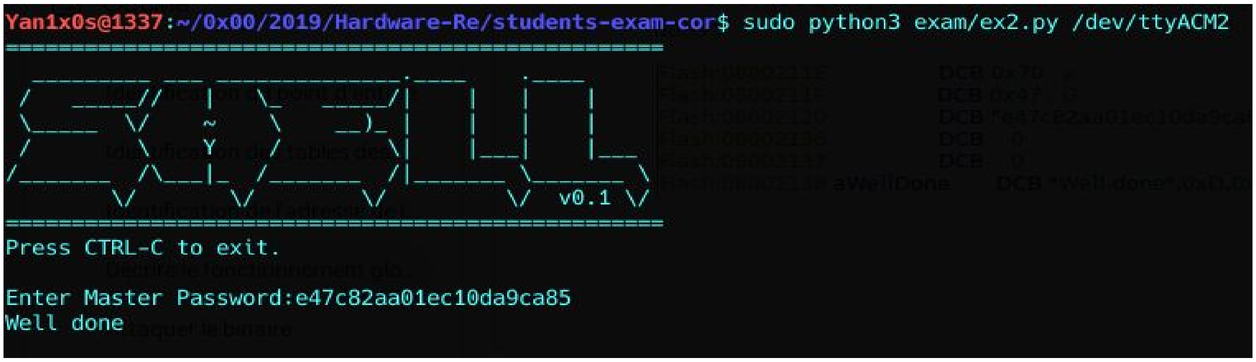
On remarque que le programme cherche une entrée à la ligne 43.
Si on a une entrée, on vérifie la taille à la ligne 49: ***Si la taille de l'entrée est 21*** on passe à la vérification de l'entrée sinon on affiche un message de perte.

```
Flash:080001AC loc_80001AC                ; CODE XREF: firmware_main+4A↑j
Flash:080001AC          CMP     R3, #0x15
Flash:080001AE          BNE     loc_80001E0
Flash:080001B0          LDR     R2, =dword_200005C4
Flash:080001B2          MOVS    R3, #0xB
Flash:080001B4          LDR     R0, [R2]
Flash:080001B6          MOVS    R1, #3
Flash:080001B8          LDR     R2, =aWellDone  ; "Well done\r\n"
Flash:080001BA          BL      sub_80011A4
```

On remarque qu'il y a une vérification char par char pour le mot de passe qui est situé dans :

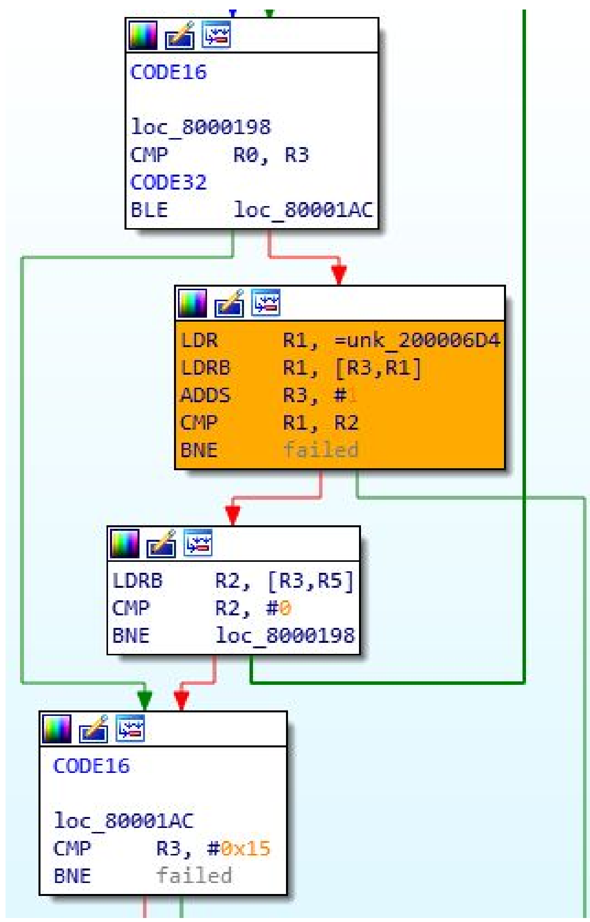
```
Flash:0800211E          DCB 0x70 ; p
Flash:0800211F          DCB 0x47 ; G
Flash:08002120          DCB "e47c82aa01ec10da9ca85",0
Flash:08002136          DCB   0
Flash:08002137          DCB   0
Flash:08002138 aWellDone    DCB "Well done",0xD,0xA,0
```

On teste l'input :



Et on a le bon le mot de passe.

Attaquer le binaire



Code vulnérable à une timing-attack

Le basic bloc orange montre une boucle de comparaison caractère par caractère. Une des conditions de sortie est lorsque les deux caractères sont différents, peu importe si la

comparaison est terminé ou non. Cette boucle de comparaison est donc vulnérable à une timing attack.

```python

```

```
import serial
import sys
from signal import signal, SIGINT
from time import sleep
import timeit
import string
import time
from itertools import product



def check4python3():
    if not sys.version_info.major == 3:
        print("Python 3 or higher is required.")
        print("You are using Python
{}.{}.".format(sys.version_info.major, sys.version_info.minor))
        sys.exit(1)



def handler(signal_received, frame):
    print('SIGINT or CTRL-C detected. Exiting gracefully')
    exit(0)



if __name__ == '__main__':
    signal(SIGINT, handler)
    check4python3()
    print('================================================')
```

```
print('  _________ ___ _________________.____     .____       ')
print(' /   _____//   |   \_   _____/|    |   |    |      ')
print(' \_____  \/    ~    \    __)_ |    |   |    |      ')
print(' /        \    Y    /        \|    |___|    |___   ')
print('/_______  /\___|_  /_______  /|_______ \_______ \ ')
print('        \/       \/        \/         \/  v0.1 \/ ')
print('=================================================')
print('Press CTRL-C to exit.')
print('')

# Open the serial port.
serial_port = serial.Serial(sys.argv[1],
                            baudrate=115200,
                            timeout=None,
                            parity=serial.PARITY_NONE,
                            bytesize=serial.EIGHTBITS,
                            stopbits=serial.STOPBITS_ONE,
                            xonxoff=False)
serial_port.flushInput()

charset = string.digits + string.ascii_lowercase
guess = bytearray(b"X" * 21)
li = list(combinations(charset,4))

# bruteforce par bloc de 4
for i in range(0,21,4):
    charset_iter = product(charset, repeat=4)
    max = (0, (, ))
    for c1, c2, c3, c4 in charset_iter:
```

```
            new = guess
            new[i] = ord(c1)
            new[i+1] = ord(c2)
            new[i+2] = ord(c3)
            new[i+3] = ord(c4)

            t = time.time()
            serial_port.write(new)

            # Give the line a small amount of time to receive data
            sleep(.5)
            bytes_to_read = serial_port.inWaiting()

            while bytes_to_read < serial_port.inWaiting():
                bytes_to_read = serial_port.inWaiting()
                sleep(.5)

            data = serial_port.read(bytes_to_read).decode()
            tt = time.time() - t
            if tt > max[0] :
                max[0] = tt
                max[1] = c1, c2, c3, c4
            print(data)
            print("{} => {}".format(new,tt))

        guess[i] = c1
        guess[i+1] = c2
        guess[i+2] = c3
        guess[i+3] = c4
```

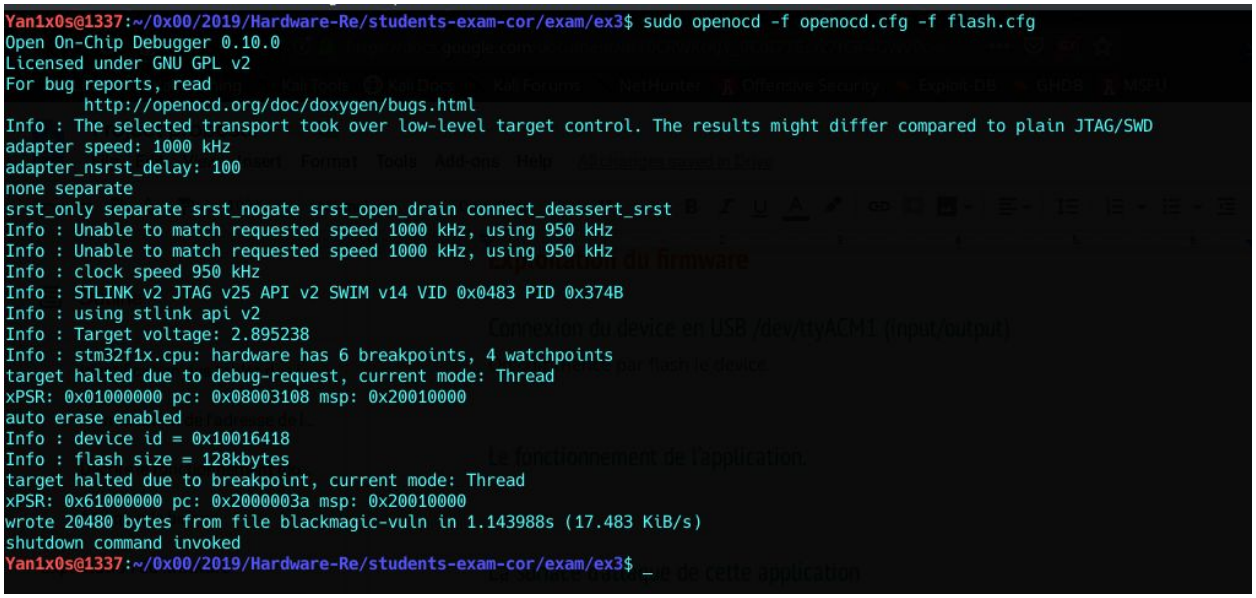
```
    serial_port.close()
```

```

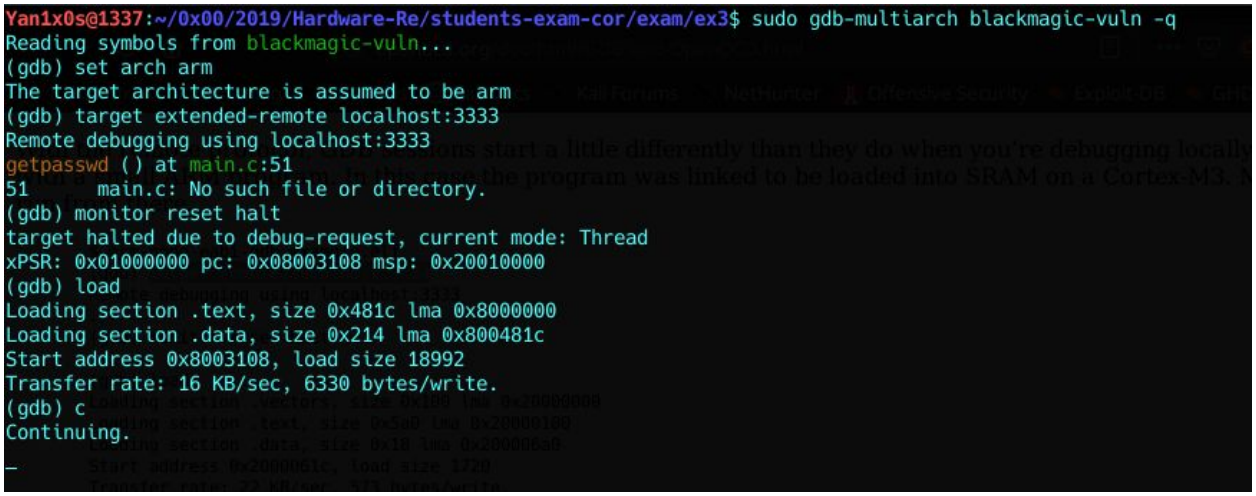
# Exploitation du firmware

## Connexion du device en USB /dev/ttyACM1 (input/output)

On commence par flash le device :




On configure gdb pour qu'il se connecte au device :


```

On envoie des données via le script python :

```
Yan1x0s@1337:~/0x00/2019/Hardware-Re/students-exam-cor/exam/ex3$ sudo python3 ex3.py /dev/ttyACM2
================================================
  _________ ___ ______________.____     .____
 /   _____//   |   \_   _____/|    |    |    |
 \_____  \/    ~    \    __)_ |    |    |    |
 /        \    Y    /        \|    |___ |    |___
/_______  /\___|_  /_______  /|_______ \|_______ \
        \/       \/        \/         \/  v0.1 \/
================================================
Press CTRL-C to exit.
```

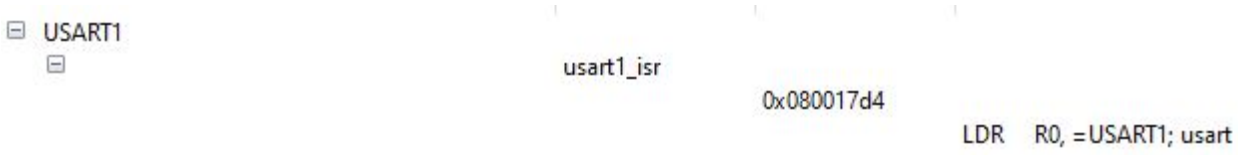
Sur gdb, on peut voir où le programme s'y arrêter :

```
(gdb) c
Continuing.
^C
Program received signal SIGINT, Interrupt.
0x0800057e in getpasswd () at main.c:51
51      in main.c
(gdb) i r
r0             0x14                20
r1             0x2000ff23          536936227
r2             0x20000354          536871764
r3             0x0                 0
r4             0xa                 10
r5             0x41                65
r6             0x20000988          536873352
r7             0x0                 0
r8             0x1c6d7c6e          476937326
r9             0xe83234b9          -399362887
r10            0xfeb88c15          -21459947
r11            0x57df5a15          1474255381
r12            0x2000ff68          536936296
sp             0x2000ff10          0x2000ff10
lr             0x8000655           134219349
pc             0x800057e           0x800057e <getpasswd+14>
xPSR           0x61000000          1627389952
msp            0x2000ff10          0x2000ff10
psp            0xd7538ae8          0xd7538ae8
primask        0x0                 0
basepri        0x0                 0
faultmask      0x0                 0
control        0x0                 0
(gdb) _
```

Le fonctionnement de l'application.

La fonction principale récupère, via la fonction *getpasswd()*, l'entrée de l'utilisateur qui est envoyée depuis l'USART. L'entrée de l'utilisateur est ensuite comparée avec la chaîne *e18496197305510df*.

La surface d'attaque de cette application



Handles pertinents et présents dans l'application

Le port USART permet d'envoyer une donnée à la plateforme. C'est une entrée pour un attaquant.

Analyser le Firmware pour identifier une vulnérabilité.

```
v8 = &buffer_in[63];
v9 = tmp_buf;
do
{
  v10 = (v9++)[1];
  (v8++)[1] = v10;
}
while ( v9 != &tmp_buf[68] );
```

Bout de code qui semble vulnérable dans la fonction getpasswd()

Exploitation de la vulnérabilité identifiée.