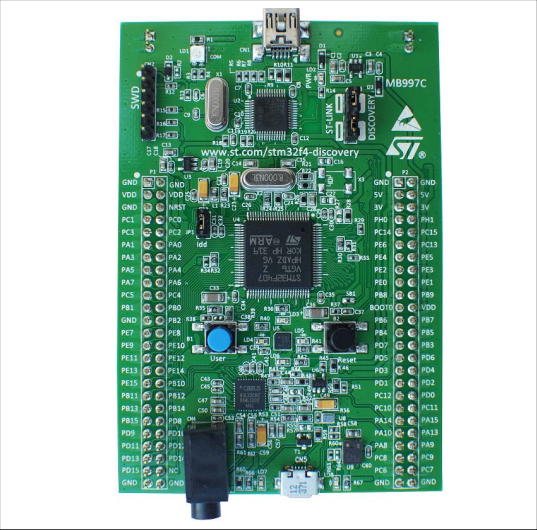
Proiect Microcontrolere

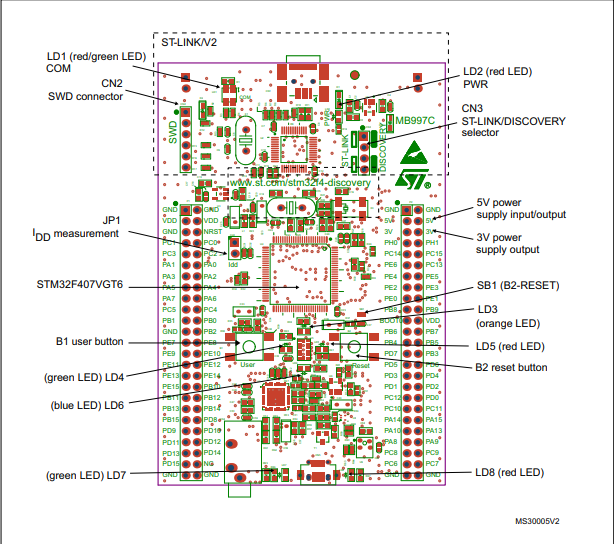
GPIO

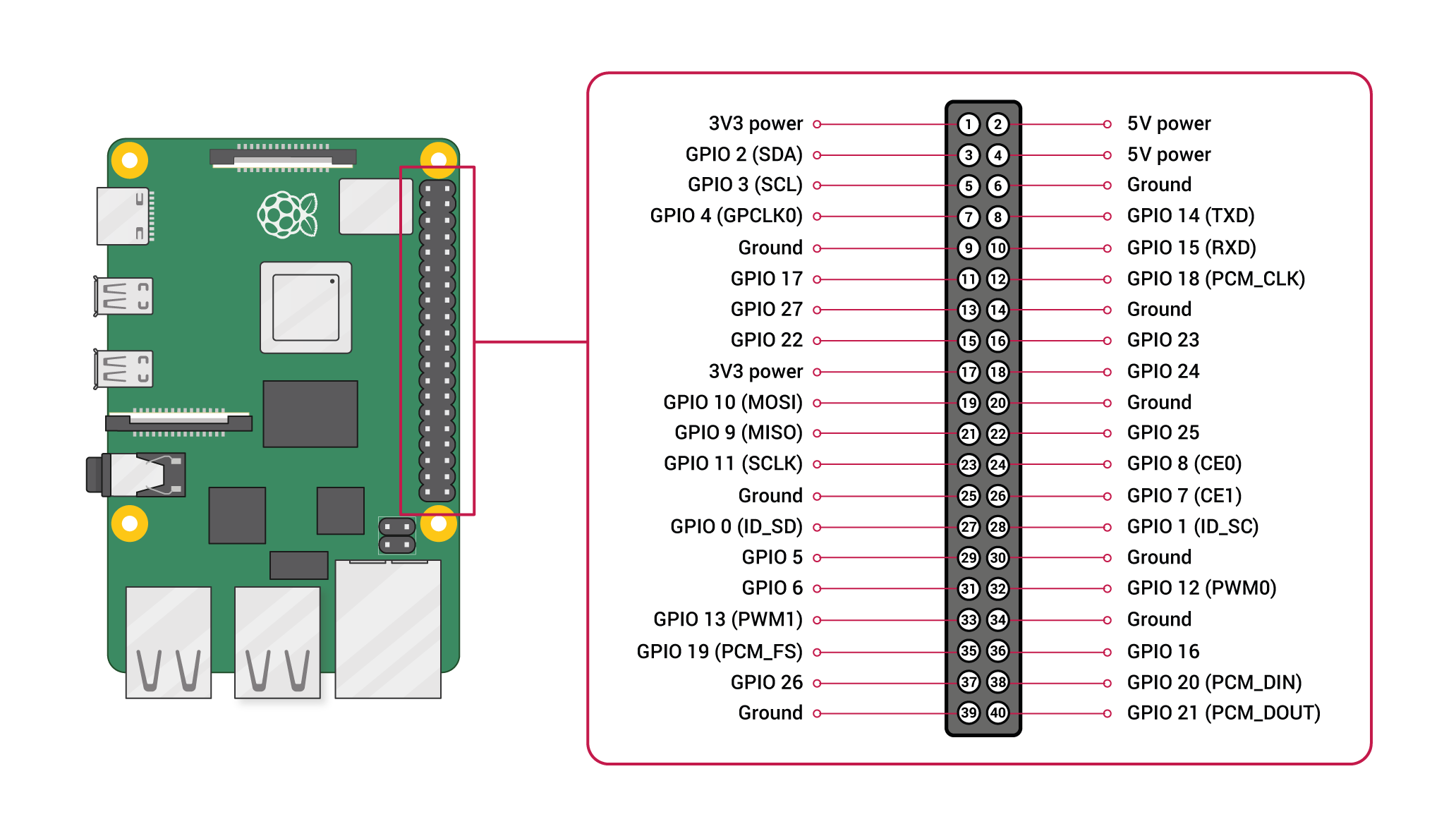
Onofrei Ioana Alexandra

C, an III, grupa 3132b

STM32F4DISCOVERY vă ajută să descoperiți liniile de performanță înaltă STM32F407 și STM32F417 și să vă dezvoltați aplicațiile. Se bazează pe un STM32F407VGT6 și include un instrument de depanare încorporat ST-LINK / V2, accelerometru digital ST MEMS, microfon digital ST MEMS, DAC audio cu driver de boxe integrat clasa D, leduri, butoane și conector micro-AB USB OTG.







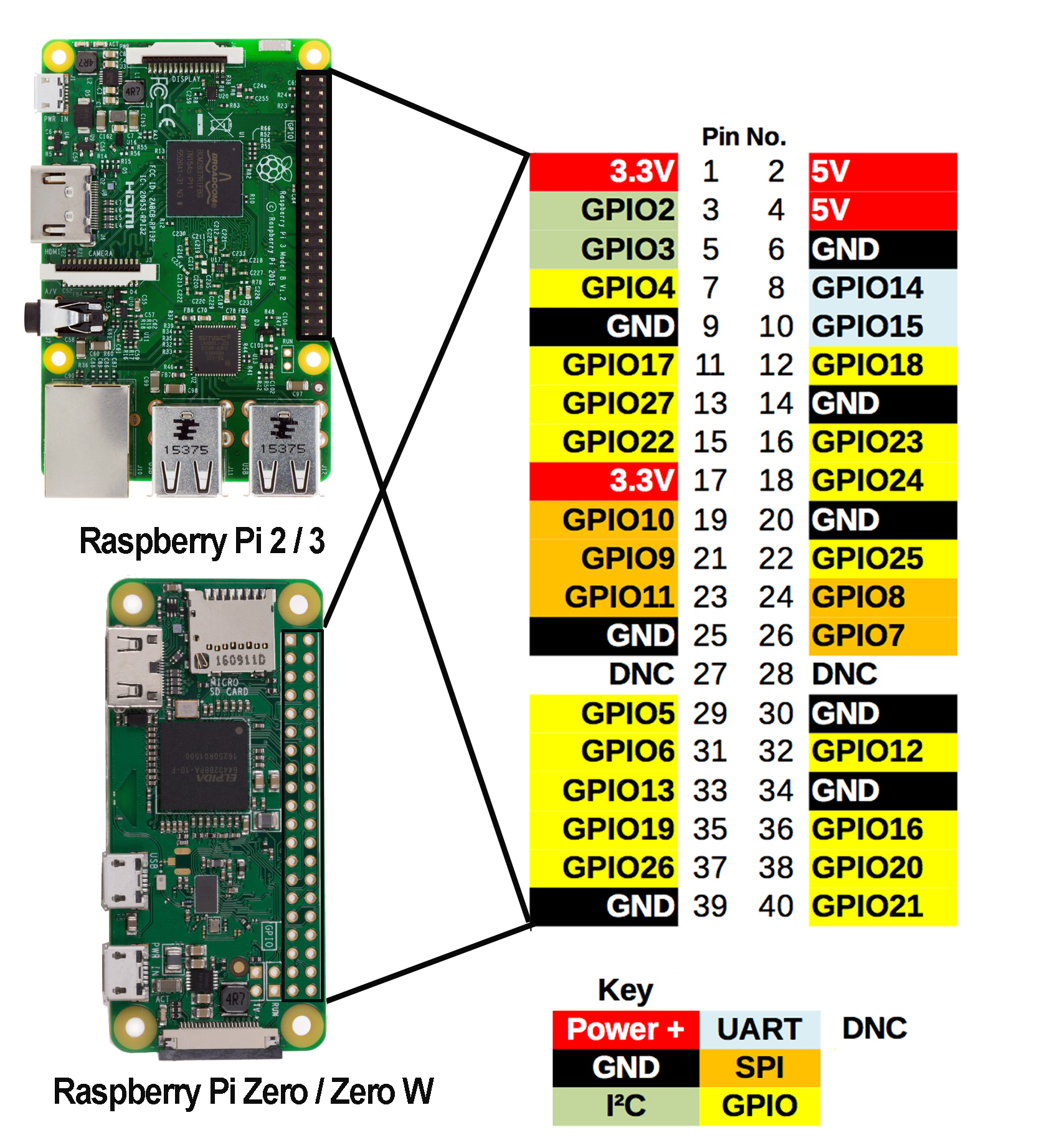
GPIO (General Purpose Input/Outpt) este un pin ce se găsește în circuitele integrate, dar care nu îndeplinește o funcție specifică. Chiar dacă majoritatea pinilor au funcțiile lor predefinite în hard, funcția pinului GPIO poate fi modificata cu ajutorul softului.

GPIO este alcătuit din :

* 4 regiștrii de configurare de configurare pe 32 de biți
* 2 regiștrii de configurare pe 32 de biți
* Un registru de blocare
* 2 regiștrii alternativi cu funcții de selecție pe 32 de biți

Există mai multe plăci care au pini GPIO, cum ar fi Raspberry Pi, Arduino sau STM32F4xx Discovery. Acești pini ne furnizează interfața dintre plăcuță și alte componente. Pinii acționează ca niște comutatoare care ne dau la ieșire 3,3 - 5 V când sunt setați pe HIGH sau 0 V când sunt setați pe LOW. Cel mai simplu exemplu de utilizare a pinilor GPIO este aprinderea și stingerea unui led.

Exemplu de GPIO folosit în Raspberry Pi



Pinii GPIO ai plăcii STM32F4xx Discovery au 4 regiștrii de configurare pe 32 de biți:

* Regiștrii de date: GPIOx\_IDR, GPIOx\_ODR;
* Registru set/reset: GPIOx\_BSRR;
* Registru de blocare: GPIOx\_LCKR;
* Regiștrii alternativi cu funcție de selecție pe 32 de biți: GPIOx\_AFRH, GPIOxAFRL;

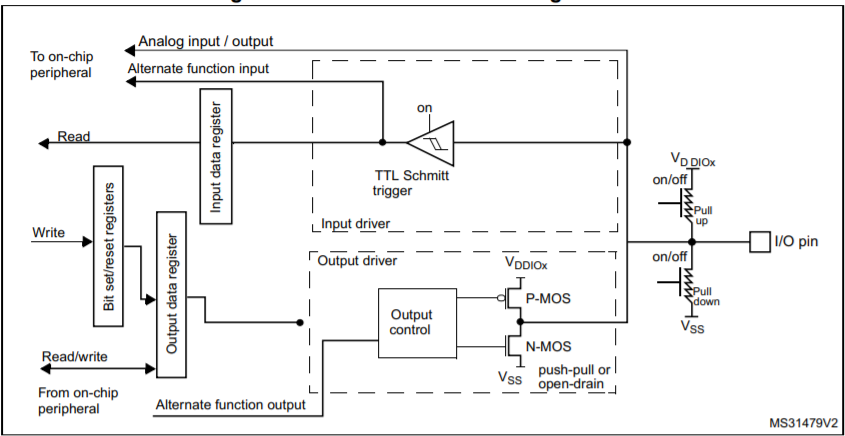
Caracteristicile principale ale pinilor GPIO:

* Putem avea până la 16 I/O-uri de control;
* Putem selecta viteza pentru fiecare port de intrare/ieșire;
* Este un comutator rapid, capabil să se schimbe la fiecare 2 cicluri de ceas;

Fiecare bit al portului de intrare/ieșire poate fi configurat individual prin software în diferite moduri:

* Intrare floating, pull-up, pull-down;
* Analog;
* Ieșire open-drain cu capacitatea pull-up/pull-down;
* Ieșire push-pull cu capacitatea pull-up/pull-down;
* Funcție alternativă push-pull cu capacitatea pull-up sau pull-down;
* Funcție alternativă open-drain cu capacitatea pull-up sau pull-down;

Putem programa fiecare bit al portului I/O, însă regiștrii acestuia trebuie să fie accesați ca și cuvinte pe 32 biți, jumătate de cuvânt. Scopul registrului set/reset este de a permite accesarea atomică de citire sau modificare la oricare dintre regiștrii GPIO. Astfel, nu există risc privind întreruperea IRQ ce apare între citire și modificarea accesului.



*Figura 1. Configurarea bitului de intrare/ieșire (Funcția alternativă)*

În timpul resetării și imediat după, funcțiile alternative nu sunt active, iar porturile de I/O sunt configurate in modul de intrare floating.

**Jocuri de lumini**

Folosind placa sau **STM32F429** Discovery am realizat mai multe jocuri de lumini cu ajutorul butonului care a servit ca semnal de intrare, și ledurile puse la dispoziție de plăcuță, care au servit ca semnal de ieșire.

Placa **STM32F429** Discovery are 4 leduri care pot fi programate de către user:

* LD3: Un led verde, care este conectat la pinul PG13;
* LD4: Un led roșu conectat la pinul de intrare/ieșire PG14;
* LD5: Un led verde care ne indică daca VBUS este prezent pe CN6 și este conectat la PB13;
* LD6: Un led roșu care ne indică supracurentul prezent de la VBUS, CN6 și este conectat la PC5;

Din aceste 4 leduri enumerate, putem folosi doar ledul trei și patru. De asemenea, această placă ne pune la dispoziție 2 butoane: unul de reset, și altul care poate fi programat de către user, și care este conectat la pinul GPIO – PA0.

Din cauza faptului că placa **STM32F429** Discovery are doar 2 leduri ce pot fi programate, am folosit o altă placă - **STM32F407** Discovery, care la fel conține cele 2 butoane, însă pe lângă aceasta, are și leduri ce chiar pot fi programate de către user:

* LD3: Un led de culoare portocalie, care este conectat la pinul PD13;
* LD4: Un led verde conectat la pinul de intrare/ieșire PD12;
* LD5: Un led roșu, care este conectat la pinul PD14;
* LD6: Un led albastru conectat la pinul de intrare/ieșire PD15;

Pentru început am inițializat ledurile, setând pentru fiecare led pinul căruia îi corespunde. Apoi, în programul principal, am apelat funcțiile din „Board\_LED.h”, și am setat pe ON / OFF ledurile, cu un mic delay între ele, pentru a observa efectul de toggle.

uint32\_t led\_cnt = LED\_GetCount();

uint32\_t led\_num = 0;

while (1)

{

for (led\_num = 0; led\_num < led\_cnt; led\_num++)

{

LED\_On(led\_num);

}

osDelay(1000);

for (led\_num = 0; led\_num < led\_cnt; led\_num++)

{

LED\_Off(led\_num);

}

}

Un alt joc de lumini ar fi acela prin care ledurile se aprind circular, astfel încât primul led se va seta pe ON, va aștepta câteva clipe, după care va fi setat pe OFF, și se va continua procedura cu alt led.

uint32\_t led\_cnt = LED\_GetCount();

uint32\_t led\_num = 0;

while (1) {

osSignalWait(1U, osWaitForever); // Wait for signal

LED\_On(led\_num); // Turn specified LED on

osSignalWait(1U, osWaitForever); // Wait for signal

LED\_Off(led\_num); // Turn specified LED off

led\_num++; // Change LED number

if (led\_num >= led\_cnt) {

led\_num = 0; // Restart with first LED

}

}

Pentru a folosi și semnalul de intrare, vom utiliza buton. În continuare este secvența de cod în care am setat ledurile pe ON doar dacă butonul este apăsat.

uint32\_t led\_cnt = LED\_GetCount();

uint32\_t led\_num = 0;

while (1)

{

if (Buttons\_GetState() & 1U)

{

for (led\_num = 0; led\_num < led\_cnt; led\_num++)

{

LED\_On(led\_num);

}

}

else

{

for (led\_num = 0; led\_num < led\_cnt; led\_num++)

{

LED\_Off(led\_num);

}

}

}

În acest mod, între blocul de la instrucțiunea if, putem pune orice joc de lumini. Însă acest joc de lumini va rula doar dacă butonul este ținut apăsat (în starea ON) mereu. Astfel, am pus în evidență și modul de intrare a pinilor GPIO.

O soluție mai eficientă de a crea un astfel de joc de lumini, este să creeam un fișier în care să ne creăm treduri, ce va conține codul de stingere și aprindere a ledurilor în diferite moduri.

Introducerea tredurilor în proiect este necesară pentru performanță, deoarece apelul funcției jocului de lumini va fi executat în paralel cu programul principal. În continuare este fișierul în care am creat următorul tred:

#include "cmsis\_os.h" // ARM::CMSIS:RTOS:Keil RTX

#include "Board\_LED.h" // ::Board Support:LED

/\*-------------------------------------------------------------

\* Thread 1 'Thread\_LED': LED thread

\*------------------------------------------------------------\*/

void Thread\_LED (void const \*argument); // thread function

osThreadId tid\_Thread\_LED;

osThreadDef(Thread\_LED, osPriorityNormal, 1, 0); // thread object

int Init\_Thread\_LED (void) {

tid\_Thread\_LED = osThreadCreate(osThread(Thread\_LED), NULL);

if (!tid\_Thread\_LED) return(-1);

return(0);

}

Iar în fișierul main am apelat aceste treduri în funcție de starea butonului:

int main (void)

{

osKernelInitialize(); /\* initialize CMSIS-RTOS \*/

HAL\_Init(); /\* Initialize the HAL Library\*/

SystemClock\_Config(); /\* Configure the System Clock\*/

LED\_Initialize(); /\* Initialize LED \*/

Buttons\_Initialize(); /\* Initialize Buttons \*/

Init\_Thread\_LED(); /\* Create LED thread \*/

osKernelStart(); /\* start thread execution \*/

while (1)

{

if (Buttons\_GetState() & 1U)

{

osSignalSet(tid\_Thread\_LED1, 1U);

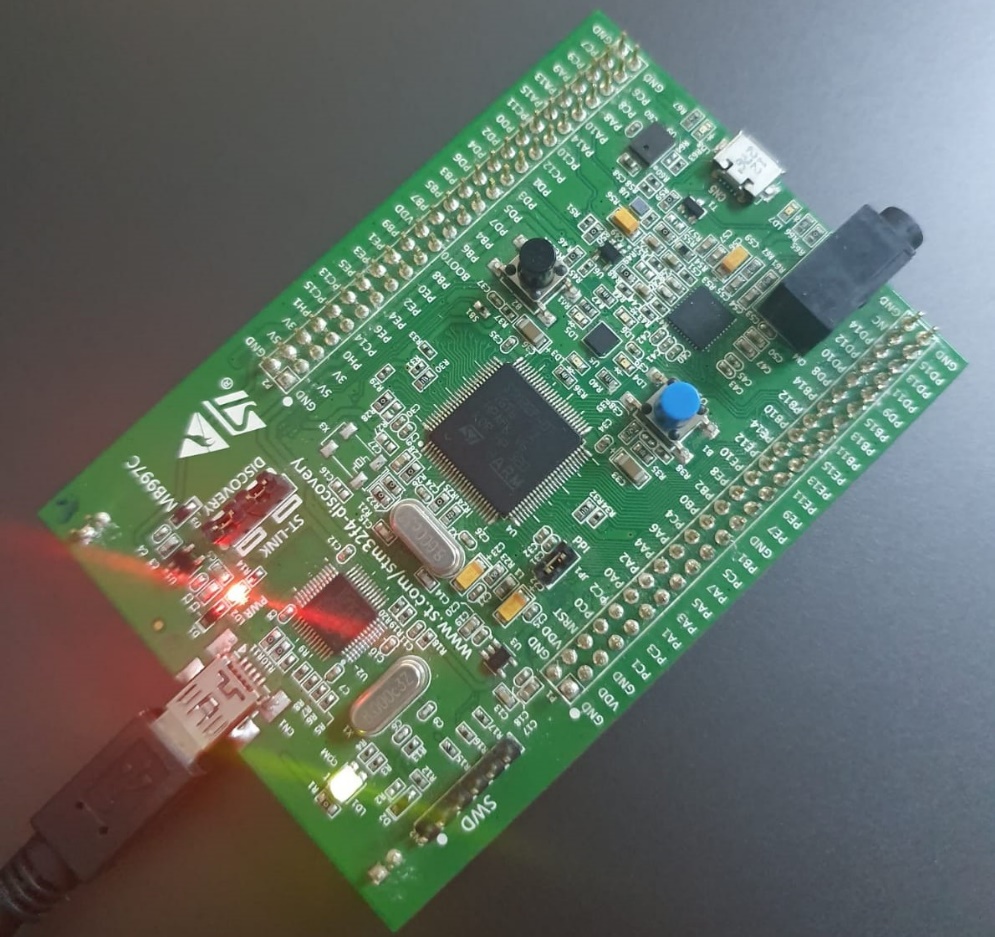
osDelay(100);

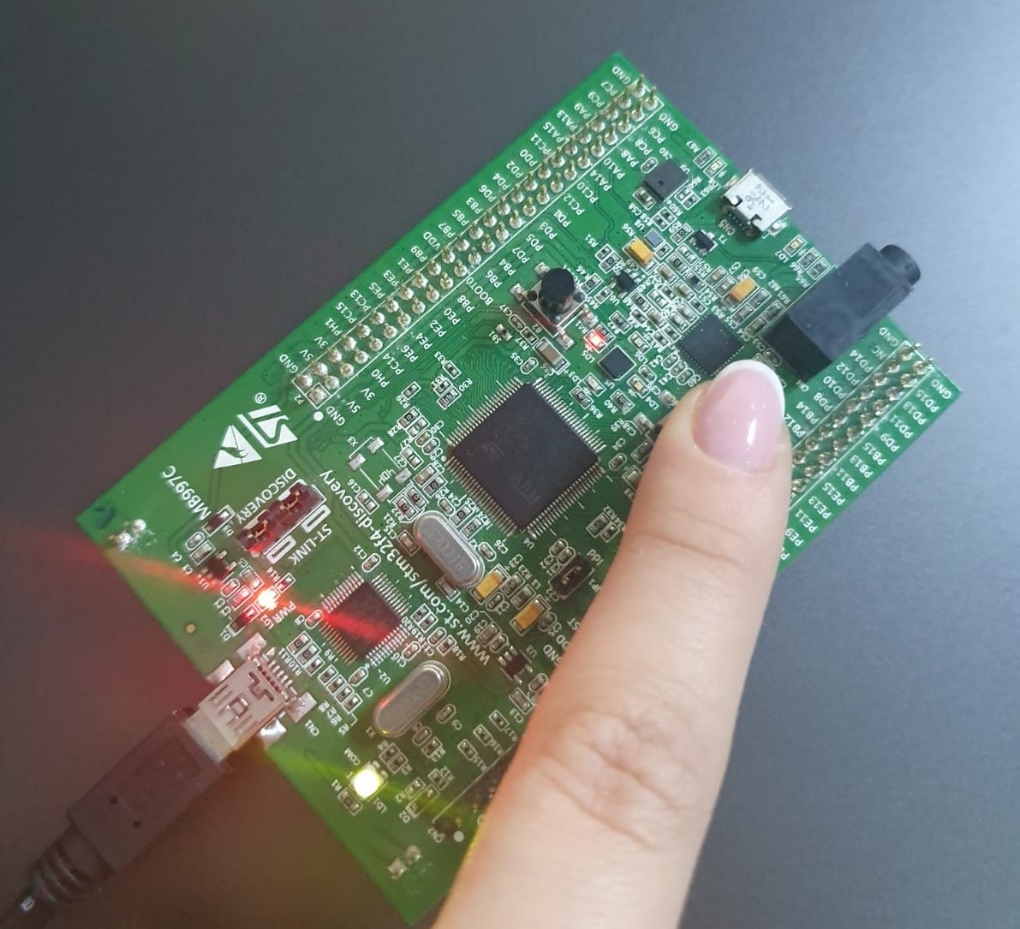
}

}

}

Astfel dacă facem o singură apăsare scurtă a butonului se va aprinde un led, dacă mai apăsăm o data, se va seta pe ON următorul led. Dacă menținem butonul apăsat (apăsare lungă), ledurile se vor aprinde circular unul după altul.





Bibliografie

<https://www.google.com/>

<https://ro.wikipedia.org/>

<http://www.eed.usv.ro/~cristinag/LabMC/STM32F407_TechRefManual.pdf>

<http://www.eed.usv.ro/~cristinag/LabMC/Lab2_GPIO_STM32.pdf>