Ceas digital cu STM32F302

Florin Nica

1. Descrierea familiei de microcontrolere utilizate

Procesoarele Arm au o arhitectură de calcul de tip RISC (Reduced Instruction Set Computer) de 32 de biți, ceea ce presupune instrucțiuni reduse într-un singur ciclu de ceas, Load și Store sunt instrucțiuni separate, complexitate redusă (număr mic de tranzistoare pe chip). RISC a monopolizat domeniul embedded datorită costului redus al memoriei, compilatoarelor eficiente, costurilor mici de fabricație și consumului redus de energie.

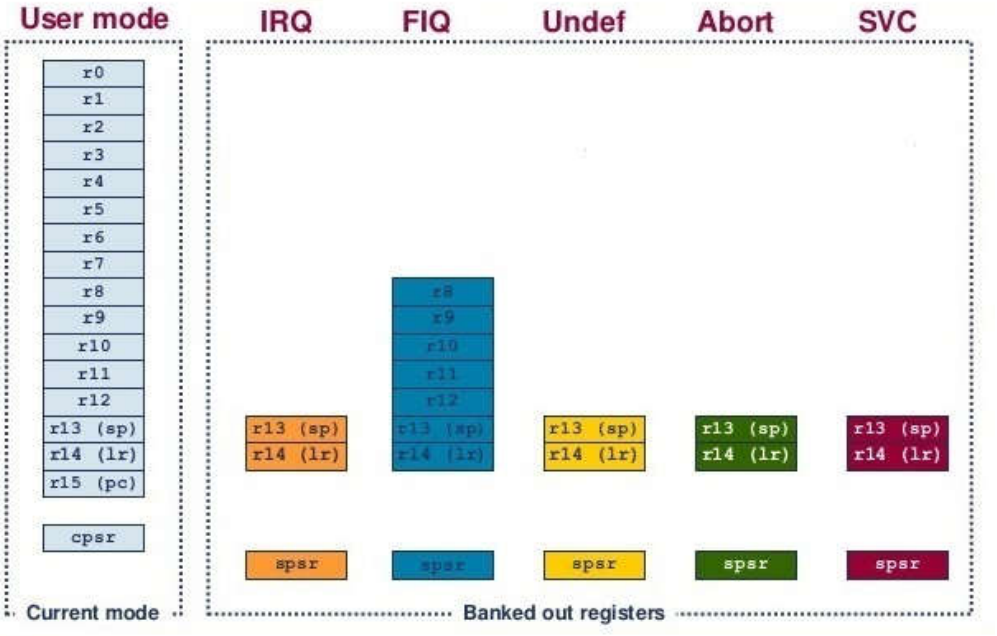
Datorită cerințelor foarte diverse din domeniul sistemelor încorporate, Arm a decis să catalogheze procesoarele în trei familii: Cortex-A, Cortex-M și Cortex-R. Familia Cortex-A este destinată aplicațiilor complexe, conține blocuri suplimentare pentru grafică, managementul memoriei și asigură suport pentru sisteme de operare avansate ca Linux, Android și Windows (de exemplu procesorul de la Raspberry Pi). În seria Cortex-R se află procesoarele folosite în aplicații de timp real sau unde siguranța joacă un rol important, de exemplu în sistemul antiblocare a frânei folosit în industria auto. Seria Arm Cortex-M include procesoare puternic încorporate (deeply embedded) optimizate pentru aplicații de tip microcontroler și de putere scăzută. Datorită prețului în scădere, aceste microcontrolere tind să le înlocuiască pe cele de 8 biți, asigurând o performanță superioară.

Procesoarele Cortex-M sunt proiectate de către Arm dar sunt produse de companiile care cumpără licența, aceasta neincluzând și perifericele - porturile intrare/ieșire, timere, convertoare ADC, interfețe I2C, UART, etc. Companiile își implementează propriile soluții pentru aceste periferice neexistând un standard general, ceea ce face destul de dificilă trecerea de la un tip de microcontroler la altul. Familia Cortex-M4 este prima care a încorporat hardware pentru unitatea de calcul în virgulă mobilă (FPU) și utilizează setul de instrucțiuni ARMv7-M, optimizat pentru aplicații embedded.

În general, procesoarele Arm folosesc două seturi de instrucțiuni: un set în care toate sunt de 32 de biți și altul care cuprinde un amestec de instrucțiuni de 16 biți și 32 de biți (setul Thumb-2). În terminologia Arm un byte are 8 biți, un semicuvânt (halfword) 16 biți și un cuvânt (word) 32 de biți.

Un procesor Arm are șapte moduri de operare: *User* (neprivilegiat), *FIQ*, *IRQ*, *Supervisor*, *Abort*, *Undef*, *System*. O aplicație poate rula în modul *User*, iar excepțiile apărute vor fi tratate într-unul din modurile privilegiate.

Procesorul are 37 de registre, care include 30 de registre de uz general (R0-R12), 6 registre de stare și un registru numărător program (PC).



Modul de lucru curent al procesorului stabilește care banc de registre este accesibil. Fiecare mod poate accesa registrele de uz general (R0-R12), indicatorul de stivă (R13), registrul de legătură (R14), numărătorul de program (R15), registrul pentru starea curentă a programului (CPSR). Modurile privilegiate (cu excepția modului System) pot accesa și registrul care conține starea salvată a programului (SPSR).

Seria Cortex-M are implementat un controler pentru deservirea excepțiilor (întreruperilor) - Nested Vector Interrupt Controller (NVIC). Există spațiu pentru un total de 255 de întreruperi și excepții dar, în funcție de producător, nu toate sunt folosite. Controlerul alocă primele 15 pentru uz intern (reset, hard fault), iar restul sunt asignate de producător perifericelor (timere, ADC, etc).

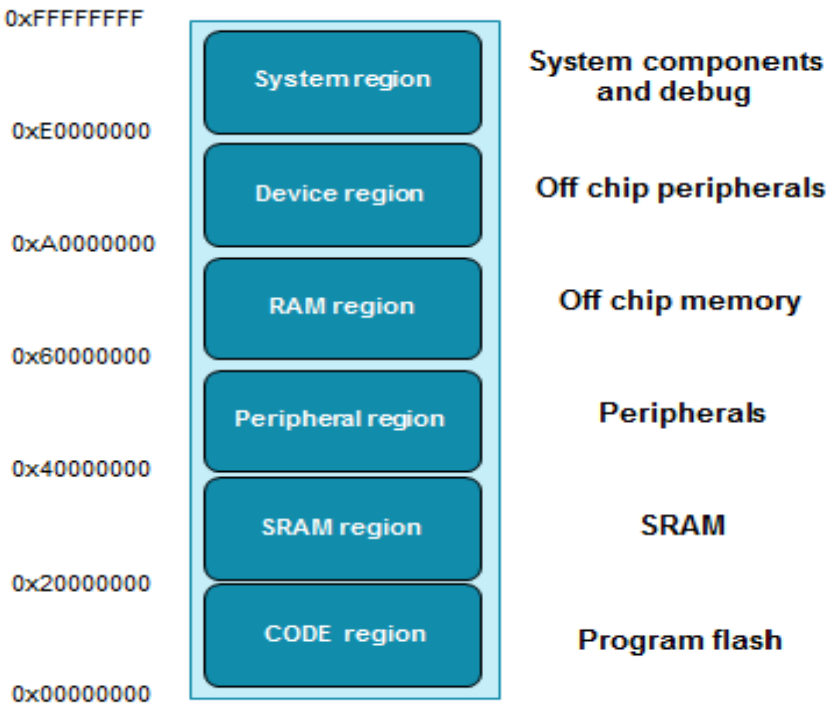
Unitatea de prelucrare în virgulă mobilă (FPU) are încă 32 de registre de 32 de biți. Ele pot fi văzute ca 32 de registre de 32 biți în simplă precizie, 16 registre de 64 de biți în dublă precizie sau o combinație validă a acestora.

Pentru interconectarea modulelor interne ale microcontrolerului se folosesc două magistrale:

- AHB (Advanced High performance Bus), care are viteză mare de transfer și conectează modulele accesate frecvent (unitatea centrală Arm, module DMA, interfața cu memoriile externe, interfața cu magistrala APB)
- APB (Advanced Peripheral Bus), care are o viteză mai mică, dar poate realiza un consum energetic mai redus și conectează perifericele care nu sunt accesate foarte frecvent

Sistemul care se bazează pe aceste două magistrale este denumit AMBA (Advanced Microcontroller Bus Architecture) și nu este specific doar procesoarelor Arm. El ajută la standardizarea interfețelor la magistrale, ușurează activitatea proiectanților și permite creșterea reutilizării diverselor produse IP în proiecte.

Harta memoriei are 4 Gigabytes și conține regiuni pentru cod, memoria SRAM, periferice interne și externe.



2. Descrierea sistemului de dezvoltare propriu-zis

Am folosit pentru acest proiect o placă de dezvoltare de tipul STM32 Nucleo-64, un shield Arduino cu LCD 2x16 + butoane și un modul Tiny RTC care conține un ceas de timp real DS1307.

Principalele caracteristici ale microcontrolerului STM32F302R8 sunt:

- 64 kB de memorie flash

-16 kB memorie RAM

- tensiune de alimentare între 2-3.6 V

- clock 72 Mhz

-1 ADC cu 15 canale cu rezoluție selectabilă de 12/10/8/6 biți

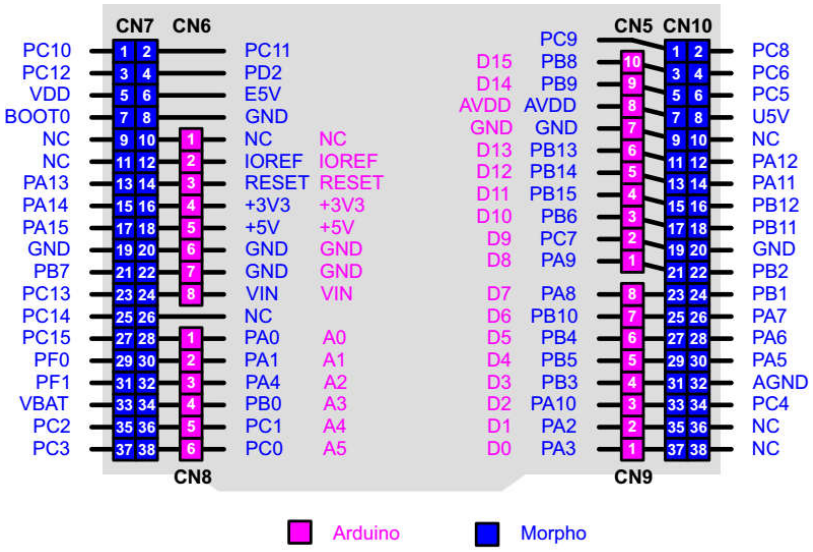
-1 DAC de 12 biți

-3 interfețe I2C

-1 interfață CAN

Placa este compusă din două secțiuni, una care conține debuggerul/programatorul ST-LINK/V2-1 iar cealaltă cu microcontrolerul și conectorii acestuia. Secțiunea cu programatorul conține un conector mini USB prin care se face și alimentarea. Există două tipuri de conectori, unii de tip ST Morpho, care permit conectarea de plăci de expansiune produse de STMicroelectronics oferind facilități suplimentare (WiFi, bluetooth, etc.) și alții care permit utilizarea shield-urilor de tip Arduino. Pe placă mai există două butoane de tip push (unul este pentru reset, celălalt este conectat la pinul PC13) și trei LED-uri (unul tricolor pentru comunicația dintre PC și ST-LINK, unul conectat la pinul PB13 și altul care indică prezența tensiunii de alimentare). Există și opțiunea alimentării dintr-o sursă externă: VIN (7-12V), E5V (5V), sau 3.3V. În cazul alimentării cu 3.3V, ST-LINKul nu este alimentat și de aceea facilitățile de depanare și programare nu sunt disponibile. Este obligatorie folosirea alimentării externe dacă Nucleo-64 și eventualele plăci de extensie consumă mai mult de 300 mA. Jumper-ul J6 (IDD) se poate înlătura pentru conectarea unui ampermetru care să permită măsurarea consumului.

Tactul implicit pentru STM32F302R8 este de 8 Mhz, nemodificabil și este livrat de către microcontrolerul (STM32F103) din ST-LINK. Mai există și variantele de a folosi un tact extern pe pinul PF0 sau adăugarea unui cristal de cuarț, a unor condensatoare și rezistoare, pentru care există loc rezervat pe placă.



Shield-ul Arduino conține un LCD cu 2x16 caractere (compatibil HD44780) și este conectat în modul 4 biți la conector:

LCD	Arduino	Nucleo-64 pin
D7	D7	PA8
D6	D6	PB10
D5	D5	PB4
D4	D4	PB5
RS	D8	PA9
E	D9	PC7
R/W	Ground	Ground

Butoanele (SELECT, UP, DOWN, RIGHT, LEFT) sunt conectate toate la același pin (PA0) prin intermediul unui circuit cu rezistoare (divizor) și citește valori de tensiune diferite în funcție de butonul apăsat.

Modulul Tiny RTC comunică prin protocolul I2C, datele fiind livrate pe magistrală în format BCD. Are un acumulator pentru menținerea configurației atunci când nu este alimentat și un circuit pentru încărcarea acestuia. Încărcat complet, acumulatorul asigură o funcționare de aproximativ un an. Modulul include și o memorie EEPROM 24C32, care nu este folosită în acest proiect.

3. Mijloace de dezvoltare și testare a aplicațiilor

Proiectul este realizat în Keil µVision5 IDE cu ajutorul aplicației STM32CubeMX, care generează cod folosind bibliotecile HAL, dezvoltate de STMicroelectronics. Etapele sunt următoarele:

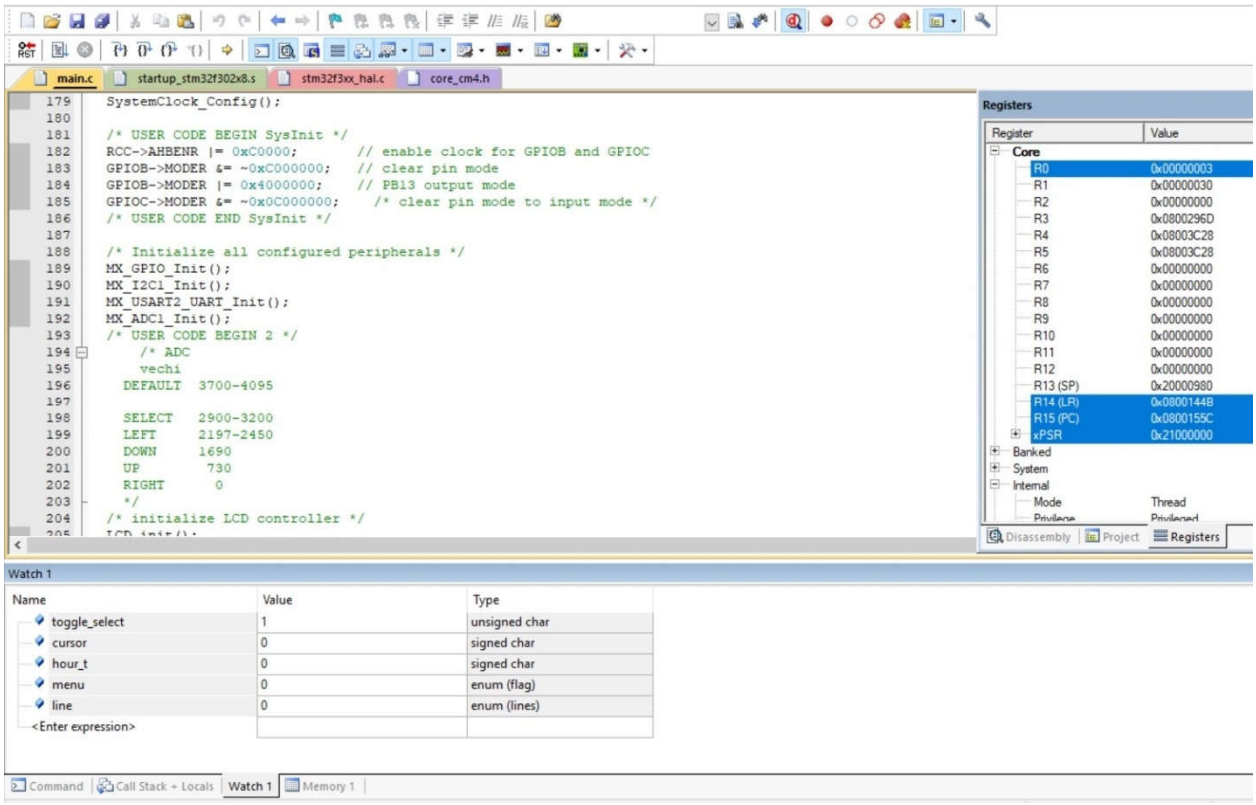
- Crearea unui proiect cu aplicația STM32CubeMX
  - *New project* - *Board Selector* (se alege Nucleo-F302R8) – *Start project*
  - Configurarea unui canal ADC pe pinul PA0 necesar butoanelor de pe shield-ul Arduino (tabul *Pinout*)
  - Configurarea pinilor PB8 și PB9 pentru comunicația I2C cu DS1307 (tabul *Pinout*)
  - Configurarea tactului realizată în tabul *Clock Configuration*
- Generarea fișierelor care vor fi integrate în mediul de dezvoltare µVision (la *Project Settings* se alege la Toolchain/IDE MDK-ARM V5)
- Aplicația µVision se lansează automat după comenzile *Generate Code* - *Open Project*
- În fereastra *Project* sunt listate toate fișierele aplicației, grupate în directoarele *Application/MDK-ARM, Application/User, Drivers/STM32F3xx_HAL_Driver, Driver/CMSIS*
- Fișierul *main.c* cuprinde codul pentru inițializarea tactului, porturilor, perifericelor precum și niște zone comentate de tipul */* USER CODE BEGIN 1 */ /* USER CODE END 1 */* . Numai în aceste intervale trebuie implementat codul aplicației deoarece codul scris în exteriorul acestor zone va fi șters la o eventuală regenerare a proiectului STM32CubeMX.

Arm, ca deținătoare a drepturilor de proprietate intelectuală pentru procesoarele Cortex, a creat o interfață standard pentru a permite o dezvoltare software eficientă pentru medii de programare diferite – CMSIS (Cortex Microcontroller Software Interface Standard). CMSIS constă din următoarele componente principale:

- CMSIS-CORE – interfața standard de programare a aplicației
- CMSIS-Driver – drivere pentru periferice

- CMSIS-DSP – biblioteci optimizate pentru diferite tipuri de date, incluzând tipuri de date în virgulă mobilă și fixă
- CMSIS-RTOS – interfață pentru implementarea unui sistem de operare în timp real
- CMSIS-DAP – debug acces port, un set de rutine care implementează caracteristici de depanare (debugging) optimizate pentru procesoare Cortex

Mediul de dezvoltare µVision dispune și de un debugger integrat pentru depanarea programului. Codul se poate rula linie cu linie, se pot observa valoarile regiștrilor din procesor (fereastra Registers) și a altor variabile folosite în program (fereastra Watch 1).



4. Descrierea aplicației demonstrative

Ceasul afișează pe linia de sus timpul și pe cea de jos data. Cu ajutorul butoanelor (SELECT, UP, DOWN, RIGHT, LEFT) se realizează setarea tuturor parametrilor: oră, minut, secundă, zi, lună, an. Apăsând butoanele pentru direcție se parcurg câmpurile, cu SELECT se selectează unul dintre ele, apoi din UP/DOWN se realizează reglajul. Se apasă din nou SELECT pentru a

confirma valoarea introdusă, și din acest moment, dacă nu se mai apasă nimic, după aproximativ 8 secunde se iese automat din modul de configurare.

Funcția main cuprinde secvența de inițializare și bucla principală a programului. În secvența de inițializare se realizează resetarea perifericelor, inițializarea memoriei flash, configurarea tactului, a perifericelor folosite, a porturilor folosite pentru LCD și setarea variabilelor cu valorile de start.

Bucla principală a programului execută următoarele operațiuni:

- execută o citire a convertorului analogic-digital (variabila *adc_raw*)
- testează dacă s-a apăsat butonul user de pe placa Nucleo, care realizează setarea inițială a ceasului, cu ajutorul funcției *setRTC()*
- face o citire a datelor din RTC
- identifică butonul apăsat cu ajutorul funcției *select_button()*
- execută instrucțiunile specifice butonului apăsat
- testează dacă se află în meniul de configurare și dacă au trecut mai mult de 8 secunde în care nu s-a apăsat nimic; în cazul în care s-au realizat modificări se salvează

Funcția HAL pentru transmitere I2C este de forma:

*HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c,* ***uint16_t*** *DevAddress,* ***uint8_t*** **pData,* ***uint16_t*** *Size,* ***uint32_t*** *Timeout)*

unde **hi2c1* este un pointer către instanța structurii unde este definit perifericul I2C, *DevAddress* este adresa dispozitivului slave, *pData* este pointerul către șirul, de lungime *Size,* care conține secvența de octeți care trebuie transmiși, iar *Timeout* reprezintă timpul maxim, în milisecunde, în care ar trebui să aibă loc transmisia. Dacă transmisia nu se termină în timpul alocat, funcția returnează HAL_TIMEOUT, în caz contrar returnează HAL_OK.

Pentru recepție funcția este

*HAL_StatusTypeDef HAL_I2C_Master_Receive(I2C_HandleTypeDef *hi2c,* ***uint16_t*** *DevAddress,* ***uint8_t*** **pData,* ***uint16_t*** *Size,* ***uint32_t*** *Timeout)*

În interiorul funcțiilor de citire și de setare a ceasului, *getRTC(),* respectiv *setRTC()* se apelează funcțiile HAL de transmitere și recepție și se face conversia BCD – zecimal, deoarece modulul RTC lucrează cu date în format BCD.

Pentru a crește lizibilitatea codului am folosit variabile de tip enumerare pentru cele două linii ale afișorului (*LINE*), pentru butoane (*PRESSED_BUTTON*) și pentru modificările operate (*clock_changes*). Variabila *cursor* ia doar valorile 0, 1, 2 și indică posibilitatea modificării orei, minutului și secundei atunci când *LINE = LINE1*, respectiv a datei, lunii și anului atunci când *LINE = LINE2*. Pentru a memora selecția unui anumit câmp am folosit variabila *toggle_select*, adică atunci când e 0, cursorul clipește (blink) și se poate modifica variabila din butoanele UP/DOWN.

Afișorul LCD dispune de următoarele funcții: display on/off, display clear, character blink or shift, cursor blink, shift și home. Controlerul HD44780 are doi regiștri, unul pentru instrucțiuni, celălalt pentru date, care sunt selectați cu ajutorul semnalului RS. În modul de lucru pe 4 biți datele sunt repartizate în felul următor: cuvintele de cod de lungime 8 biți sunt împărțite în două secvențe a câte 4 biți. Pentru a trimite date sau instrucțiuni mai întâi sunt trimiși primii 4 biți pe magistrală, se emite un impuls de ceas (E), apoi sunt trimiși restul de 4 biți, apoi un impuls de ceas. Cele două secvențe de câte 4 biți sunt procesate intern de către controler pentru a forma un cuvânt de cod de 8 biți, apoi este executată operația corespunzătoare acestui cuvânt de cod. Controlerul dispune de un indicator (flag) BUSY care trebuie citit pentru a verifica dacă s-a terminat de executat instrucțiunea anterioară dar acest mod nu poate fi utilizat în acest caz deoarece pinul R/W este conectat la masă (ground) în shield-ul Arduino, deci afișorul se poate folosi doar în mod scriere. Din acest motiv, pentru a fi siguri că nu există operațiuni în desfășurare, după fiecare instrucțiune trimisă se introduce o mică întârziere de o durată mai mare sau egală cu timpul de execuție al instrucțiunii, timp care este precizat în foaia de catalog a controlerului.

În funcția *PORTS_init()* se realizează configurarea pinilor care se conectează la LCD ca output prin intermediul regiștrilor MODER asociați fiecărui port și se activează tactul pentru aceștia (registrul RCC_AHBENR, biții 17, 18, 19 pentru porturile A, B, C). Pentru fiecare pin există doi biți asociați în registrul MODER, în funcție de a căror combinație pinul va fi configurat în modul input, output, alternate sau analog.

În interiorul funcției *LCD_init()* se apelează funcția *PORTS_init()* și se realizează setările inițiale, adică selectarea modului 4 biți, lcd pe 2 linii, font de 5x7 pixeli, cursor incrementat automat către dreapta.

Funcția *LCD_nibble_write(char data, uint16_t control)* primește ca argumente octetul care trebuie transmis către LCD și bitul pentru selectare instrucțiuni/date RS. Pentru a putea transmite tot octetul este apelată de două ori în interiorul funcțiilor *LCD_command(unsigned char command)* și *LCD_data(char data)*. Pentru a seta/reseta pini într-un port se utilizează registrul BSRR asociat fiecărui port. Registrul are funcție doar pentru scriere, orice încercare de a face citire returnând 0. Biții 31-16 sunt pentru setare, iar 15-0 pentru resetare, modificările realizându-se doar prin scrierea de biți cu valoarea 1, scrierea de 0 neavând niciun efect asupra pinilor portului. În cazul setării atât a bitului de set cât și de reset, bitul de set are prioritate. Deoarece pinii de date (D7-D6) fac parte din regiștrii diferiți, este nevoie de operații pe biți pentru transmiterea corespunzătoare a datelor/comenzilor către LCD:

```
GPIOA->BSRR = (data << 1) & 0x100;   /* bit PA8    D7 */
GPIOB->BSRR = (data << 4) & 0x400;   /* bit PB10  D6 */
GPIOB->BSRR = (data >> 1) & 0x10;    /* bit PB4    D5 */
GPIOB->BSRR = (data << 1) & 0x20;    /* bit PB5    D4 */
```

Fiecare din cele patru instrucțiuni de mai sus selectează un bit din primii 4 cei mai semnificativi ai octetului care trebuie transmis și setează pinii prin intermediul registrului BSRR. Deoarece se execută doar operații de setare de biți, pentru a nu altera datele la următoarea apelare a funcției, pinii sunt resetați înainte de aceste patru instrucțiuni cu instrucțiunile:

```
GPIOA->BSRR = 0x1000000;      /* clear data bit PA8 */
GPIOB->BSRR = 0x4300000;      /* clear data bits PB4 PB5 PB10 */
```

Bibliografie:

https://www.youtube.com/watch?v=z3VMKQKJIbA
http://www.microdigitaled.com/ARM/STM_ARM_books.html
http://www.thomasclausen.net/en/walking-through-the-1602-lcd-keypad-shield-for-arduino/
https://www.st.com/content/ccc/resource/technical/document/reference_manual/16/bf/04/cd/f6/25/44/a7/DM00094349.pdf/files/DM00094349.pdf/jcr:content/translations/en.DM00094349.pdf