Sistem de autentificare cu parolă pe placa Nucleo-64

**Disciplina:** Sisteme Încorporate

**Student 1:** Gheoace Stelian Valentin

**Student 2:** Gonța Andrei Serafim Daniel

**An:** lll, CTI-Ro

**Caracteristici**

La inițializarea sistemului, acesta se află în starea „locked”, iar pentru deblocare, utilizatorului ii este ceruta, printr-un mesaj afisat pe LCD, introducerea unei parole.

Utilizand tastatura placii, utilizatorul poate itroduce parola pentru a debloca sistemul, avand la dispozitie 3 incercari.

In cazul in care parola introdusa este cea corecta, sistemul trece din starea „locked” in „unlocked”, actiune confirmata si de aparitia unui mesaj corespunzator pe afisazul LCD. Astfel, utilizatorul se poate folosi de functionalitatile placii.

In caz contrar, cele 4 leduri de culoare rosie dispuse deasupra tataturii se vor aprinde iar utilizatorului i se va cere din nou introducerea parolei. Dupa epuizarea celor 3 incercari, sistemul va ramane blocat in starea „locked”.

**Placa Nucleo-64 F446RE** **cu microcontroller STM32F446RE Arm Cortex M4**

Placa Nucleo-64 F446RE este echipată cu un microcontroller STM32F446RE bazat pe nucleul Arm Cortex-M4. Această placă este concepută pentru a oferi o platformă de dezvoltare versatilă și puternică, ideală pentru o gamă largă de aplicații embedded.

Placa Nucleo-64 F446RE include următoarele caracteristici:

1. Patru LED-uri
2. Patru butoane
3. DAC pe 12 biți pentru testarea interfeței I2C și generarea de forme de undă analogice
4. Display LED cu 7 segmente
5. Difuzor controlat de temporizator sau DAC
6. 4 intrări de probă logică
7. Indicatori LED de alimentare pentru 3V și 5V
8. Tastatură 4 x 4
9. LCD cu 16X2 caractere, cu jumper selectabil pentru interfață serială sau paralelă pe 4 biți
10. Potențiometru pentru intrare analogică
11. Senzor de temperatură
12. Senzor de lumină
13. Slot pentru card de memorie MicroSD
14. Breadboard
15. Conector pentru OLED de 0.96", 128x64
16. Conector pentru TFT QVGA de 2.2”, interfața RTC DS3231, CAN, FTDI

Placa poate rula cod scris în diverse limbaje de programare și medii de dezvoltare precum:

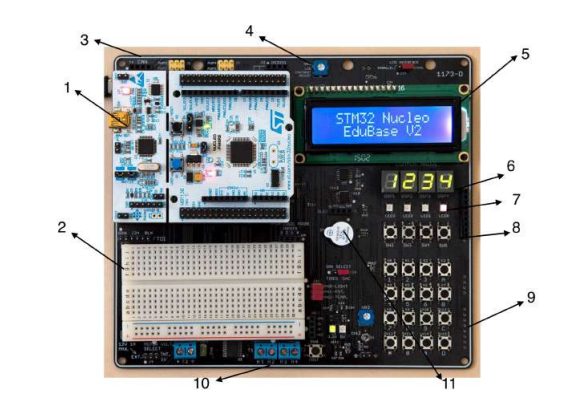
* STM32CubeIDE: IDE oficial de la STMicroelectronics, care oferă suport complet pentru dezvoltarea pe STM32.
* Keil MDK: Mediu de dezvoltare profesional pentru microcontroller-e ARM.
* IAR Embedded Workbench: Un alt mediu de dezvoltare profesional.
* Arduino IDE: Prin intermediul STM32duino, placa poate fi programată folosind limbajul de programare Arduino, ceea ce simplifică foarte mult dezvoltarea pentru utilizatorii mai puțin experimentați.

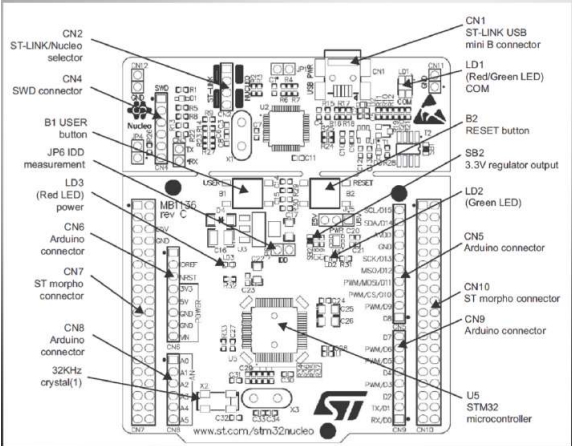
**Avantaje:**

* **Performanță ridicată**: Microcontroller-ul STM32F446RE oferă o frecvență de până la 180 MHz și are capacități avansate de procesare.
* **Versatilitate**: Suport pentru numeroase interfețe de comunicare și periferice, ceea ce o face potrivită pentru o gamă largă de aplicații.
* **Compatibilitate Arduino:** Permite utilizarea shield-urilor Arduino, extinzând funcționalitățile plăcii.
* **Debugging integrat**: Facilități de debugging și programare ușoare prin ST-LINK/V2-1

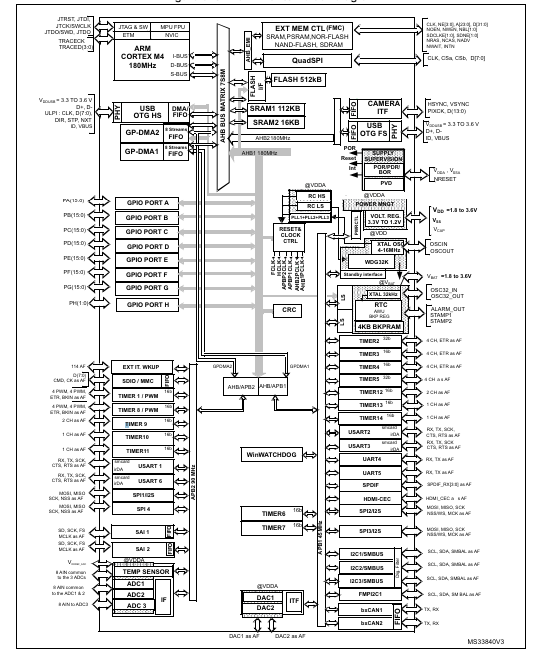
**Dezavantaje**

* Complexitate: Pentru utilizatorii începători, configurarea și utilizarea poate fi mai complicată comparativ cu alte platforme mai simple.
* Dimensiune: Placa poate fi mai mare comparativ cu alte plăci de dezvoltare mai compacte, ceea ce poate fi un dezavantaj în anumite aplicații cu constrângeri de spațiu.

****

****

**STM32F446 Block Diagram**

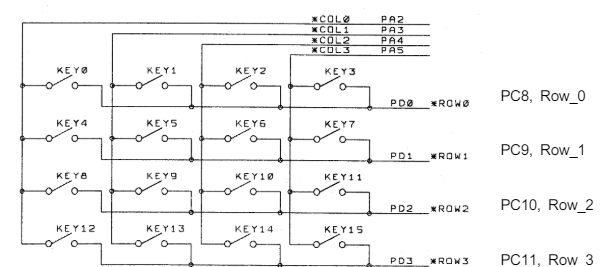
****

**Module utilizate:**

* **Tastatura 4x4**

Registrii PC8-PC11 și PB12-PB15 sunt folosiți pentru tastatură 4X4. De asemenea, daca utilizatorul nu doreste utilizarea tastaturii, acestia pot fi utilizați ca pini I/O (intrare/ieșire) de uz general.

****

****

* **LEDs**

Fiecare pin din PB4-PB7 este conectat la un LED prin intermediul unui buffer (IC U8). Pentru a aprinde un LED, trebuie ca pinul corespunzător al portului B sa fie setat ca ieșire și apoi activat prin setarea valorii “1”.

* **Display LCD**

Placa EduBase-V2 include un registru de deplasare 74HCT595 (U6) pentru a controla afișajul LCD.

Selectarea cipului pentru HCT595 este PA12.

Ieșirile U6 QA-QH sunt folosite ca biți de control și biti date D0-D1, D4-D7 pentru LCD.

Pinout-ul conectorului J1 este următorul:

* Pin 1: GND
* Pin 2: VCC (5V)
* Pin 3: Conectat la GND prin VR1 pentru ajustarea contrastului
* Pin 4: QA (D0) - pinul RS pentru modulul LCD
* Pin 5: GND - Scriere doar pentru modulul LCD
* Pin 6: QB (D1) - pinul EN pentru modulul LCD
* Pin 7: Nu este folosit
* Pin 8: Nu este folosit
* Pin 9: Nu este folosit
* Pin 10: Nu este folosit
* Pin 11: QE (D4) - pinul DB4 pentru modulul LCD
* Pin 12: QF (D5) - pinul DB5 pentru modulul LCD
* Pin 13: QG (D6) - pinul DB6 pentru modulul LCD
* Pin 14: QH (D7) - pinul DB7 pentru modulul LCD
* Pin 15: Printr-un rezistor de 22 Ohmi la VCC - iluminarea de fundal a LED-ului pentru modulul LCD
* Pin 16: GND pentru iluminarea de fundal EN/DIS pentru iluminarea LED

Registrul de deplasare 74HCT595 este conectat la controlerul LCD astfel:

* QE ~ QH la DB4 ~ DB7
* QA la RS
* QB la enable
* QC și QD nu sunt folosite.

**Cod aplicatie**

int main(void)

{

U8 status = 0;

PeriphInit();

keypad\_init();

LCD\_init();

RCC->AHB1ENR |= 2; /\* enable GPIOB clock \*/

GPIOB->MODER &= ~0x0000ff00; /\* clear pin mode \*/

GPIOB->MODER |= 0x00005500; /\* set pins to output mode \*/

while(1)

{

LCD\_data('P');

LCD\_data('A');

LCD\_data('S');

LCD\_data('S');

LCD\_data('W');

LCD\_data('O');

LCD\_data('R');

LCD\_data('D');

LCD\_data(':');

delayMs(1000);

if(tries>0)

{

citire(); // functie pentru citirea parolei folosind tastatura 4x4

if(count==4)

{

/\* clear LCD display \*/

LCD\_command(1);

delayMs(1000);

writeLEDs(0x0); // stingere leduri

LCD\_data('U');

LCD\_data('N');

LCD\_data('L');

LCD\_data('O');

LCD\_data('C');

LCD\_data('K');

LCD\_data('E');

LCD\_data('D');

LCD\_data('!');

k=0; // resetare parametrii

count=0;

}

else

{

/\* clear LCD display \*/

LCD\_command(1);

delayMs(1000);

tries--;

LCD\_data('W');

LCD\_data('R');

LCD\_data('O');

LCD\_data('N');

LCD\_data('G');

writeLEDs(0xF);

k=0; // resetare parametrii si reintroducere parola

count=0;

delayMs(1000);

LCD\_command(1);

citire();

}

}

else

{

LCD\_data('L');

LCD\_data('O');

LCD\_data('C');

LCD\_data('K');

LCD\_data('E');

LCD\_data('D');

LCD\_data('!');

delayMs(1000);

LCD\_command(1);

}

}

}

void LCD\_init(void)

{

RCC->AHB1ENR |= 1; /\* enable GPIOA clock \*/

RCC->AHB1ENR |= 4; /\* enable GPIOC clock \*/

RCC->APB2ENR |= 0x1000; /\* enable SPI1 clock \*/

/\* PORTA 5, 7 for SPI1 MOSI and SCLK \*/

GPIOA->MODER &= ~0x0000CC00; /\* clear pin mode \*/

GPIOA->MODER |= 0x00008800; /\* set pin alternate mode \*/

GPIOA->AFR[0] &= ~0xF0F00000; /\* clear alt mode \*/

GPIOA->AFR[0] |= 0x50500000; /\* set alt mode SPI1 \*/

/\* PA12 as GPIO output for SPI slave select \*/

GPIOA->MODER &= ~0x03000000; /\* clear pin mode \*/

GPIOA->MODER |= 0x01000000; /\* set pin output mode \*/

/\* initialize SPI1 module \*/

SPI1->CR1 = 0x31F;

SPI1->CR2 = 0;

SPI1->CR1 |= 0x40; /\* enable SPI1 module \*/

/\* LCD controller reset sequence \*/

delayMs(20);

LCD\_nibble\_write(0x30, 0);

delayMs(5);

LCD\_nibble\_write(0x30, 0);

delayMs(1);

LCD\_nibble\_write(0x30, 0);

delayMs(1);

LCD\_nibble\_write(0x20, 0); /\* use 4-bit data mode \*/

delayMs(1);

LCD\_command(0x28); /\* set 4-bit data, 2-line, 5x7 font \*/

LCD\_command(0x06); /\* move cursor right \*/

LCD\_command(0x01); /\* clear screen, move cursor to home \*/

LCD\_command(0x0F); /\* turn on display, cursor blinking \*/

}

void LCD\_nibble\_write(char data, unsigned char control)

{

data &= 0xF0; /\* clear lower nibble for control \*/

control &= 0x0F; /\* clear upper nibble for data \*/

SPI1\_write (data | control); /\* RS = 0, R/W = 0 \*/

SPI1\_write (data | control | EN); /\* pulse E \*/

delayMs(0);

SPI1\_write (data);

}

void LCD\_command(unsigned char command)

{

LCD\_nibble\_write(command & 0xF0, 0); /\* upper nibble first \*/

LCD\_nibble\_write(command << 4, 0); /\* then lower nibble \*/

if (command < 4)

delayMs(2); /\* command 1 and 2 needs up to 1.64ms \*/

else

delayMs(1); /\* all others 40 us \*/

}

void LCD\_data(char data)

{

LCD\_nibble\_write(data & 0xF0, RS); /\* upper nibble first \*/

LCD\_nibble\_write(data << 4, RS); /\* then lower nibble \*/

delayMs(1);

}

void delayMs(int n)

{

int i;

for (; n > 0; n--)

for (i = 0; i < 3195; i++) ;

}

void citire()

{

while(k<4)

{

while((key = keypad\_getkey()) == 0);

if(key != corect[k])

{

}

else

{

++count;

}

++k;

while(keypad\_getkey() != 0);

}

}

char keypad\_getkey(void)

{

int row, col;

/\* check to see any key is pressed first \*/

outputEnableCols(0xF); /\* enable all columns \*/

writeCols(0xF); /\* and drive them high \*/

delay(); /\* wait for signal to settle \*/

row = readRows(); /\* read all rows \*/

writeCols(0x0); /\* discharge all columns \*/

outputEnableCols(0x0); /\* disable all columns \*/

if (row == 0) return 0; /\* if no key pressed, return a zero \*/

/\* If a key is pressed, it gets here to find out which key.

\* It activates one column at a time and read the rows to see

\* which is active.

\*/

for (col = 0; col < 4; col++) {

outputEnableCols(1 << col); /\* enable one column \*/

writeCols(1 << col); /\* turn the active row high \*/

delay(); /\* wait for signal to settle \*/

row = readRows(); /\* read all rows \*/

writeCols(0x0); /\* discharge all columns \*/

if (row != 0) break; /\* if one of the row is low, some key is pressed. \*/

}

outputEnableCols(0x0); /\* disable all columns \*/

if (col == 4)

return 0; /\* if we get here, no key is pressed \*/

/\* gets here when one of the rows has key pressed.

\* generate a unique key code and return it.

\*/

if (row == 0x01) {return 0 + col;} // key in row 0

if (row == 0x02) {return 4 + col; } // key in row 1

if (row == 0x04) {return 8 + col; } // key in row 2

if (row == 0x08) {return 12 + col; }// key in row 3

return 0; /\* just to be safe \*/

}