Sistem de autentificare cu parolă pe placa Nucleo-64

Disciplina: Sisteme Încorporate

Student 1: Irimia Dorin-Alexandru

Student 2: Hăpăianu Robert

An: lll, CTI-Ro

Caracteristici

La inițializarea sistemului, acesta se află în starea „locked”**,** iar pentru deblocare, utilizatorului îi este cerută, printr-un mesaj afișat pe LCD, introducerea unei parole.

Utilizând tastatura plăcii, utilizatorul poate introduce parola pentru a debloca sistemul, având la dispoziție 3 încercări.

În cazul în care parola introdusă este cea corectă, sistemul trece din starea „locked” în „unlocked”, acțiune confirmată și de apariția unui mesaj corespunzător pe afișajul LCD. Astfel, utilizatorul se poate folosi de funcționalitățile plăcii.

În caz contrar, cele 4 leduri de culoare roșie dispuse deasupra tastaturii se vor aprinde iar utilizatorului i se va cere din nou introducerea parolei. După epuizarea celor 3 încercări, sistemul va rămâne blocat în 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

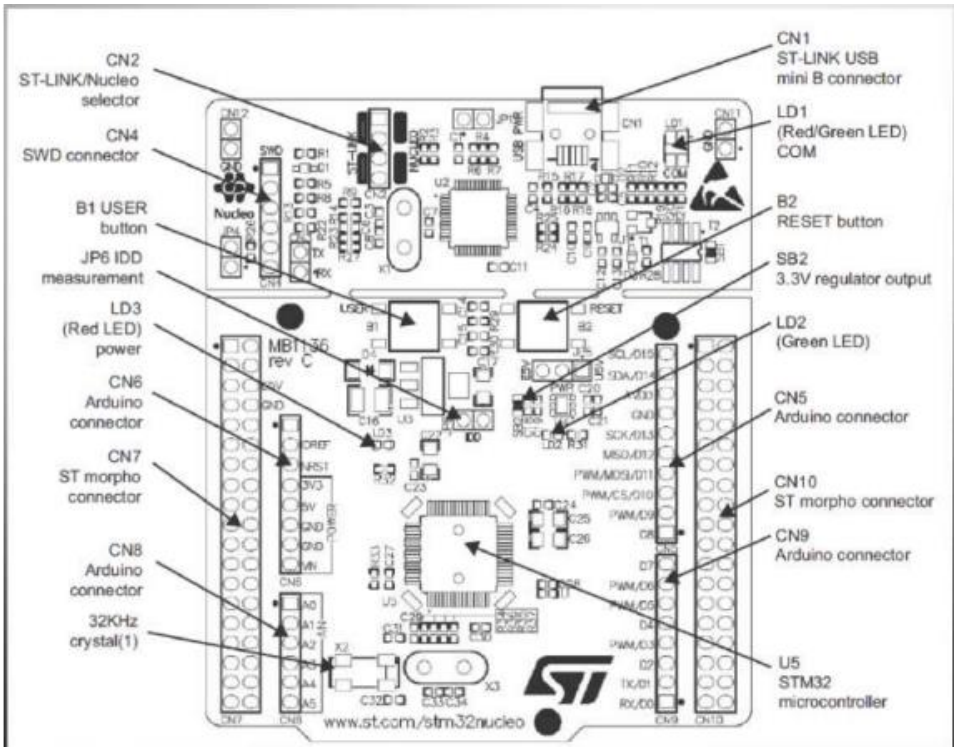
12. Senzor de lumină
13. Slot pentru card de memorie MicroSD
14. Breadboard
15. Conector pentru OLED de 0.96", 128x64
16. Conector pentru TFT QVGA de 2.2”, interfața RTC DS3231, CAN, FTDI

Avantaje:

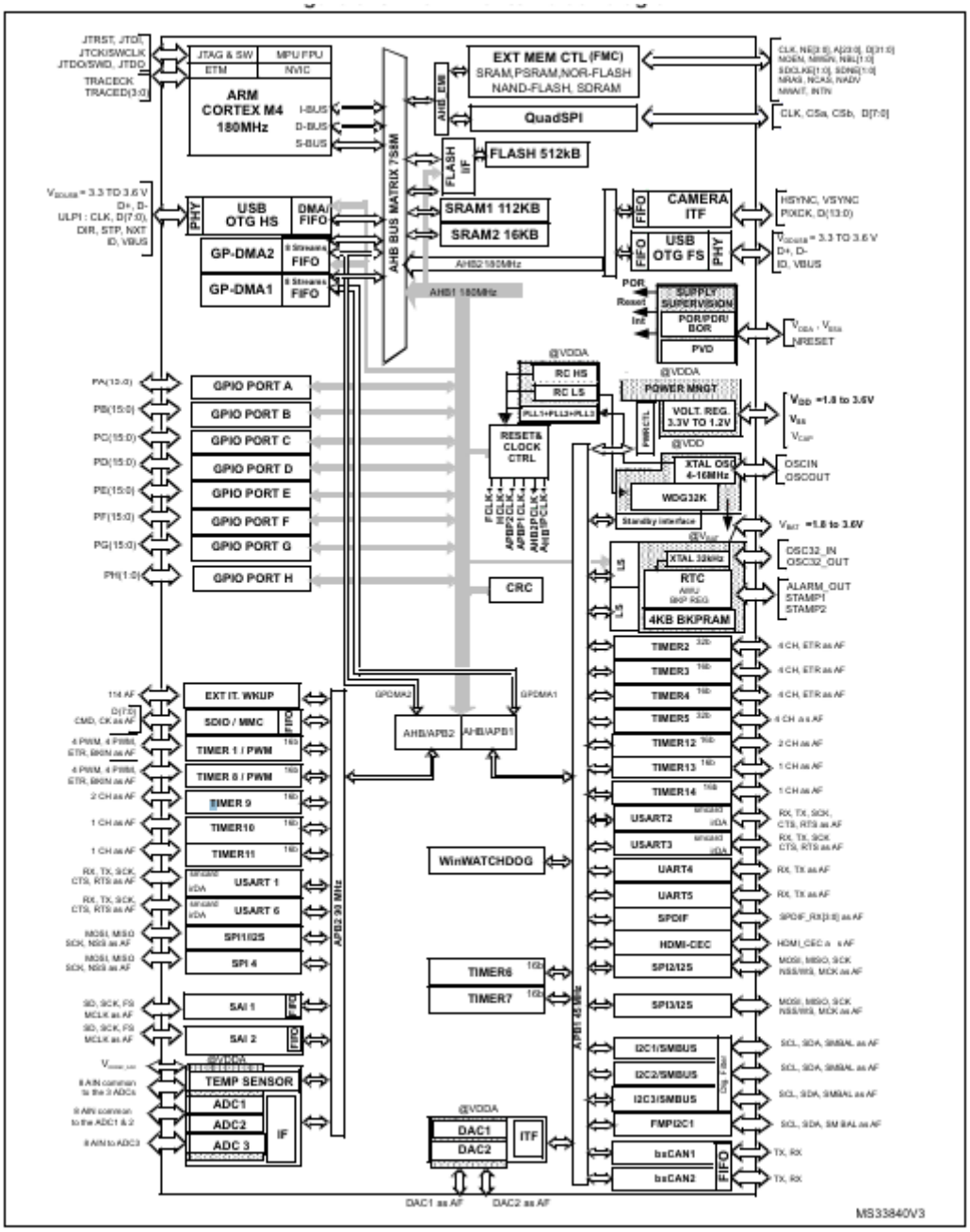
- **Performanță ridicată**: Microcontroller-ul STM32F446RE oferă o frecvență de până la 180 MHz și are capacități avansate de procesare.
- **Versatilitate**: Suport pentru numeroase interfețe de comunicare și periferice, ceea ce o face potrivită pentru o gamă largă de aplicații.
- **Compatibilitate Arduino:** 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.



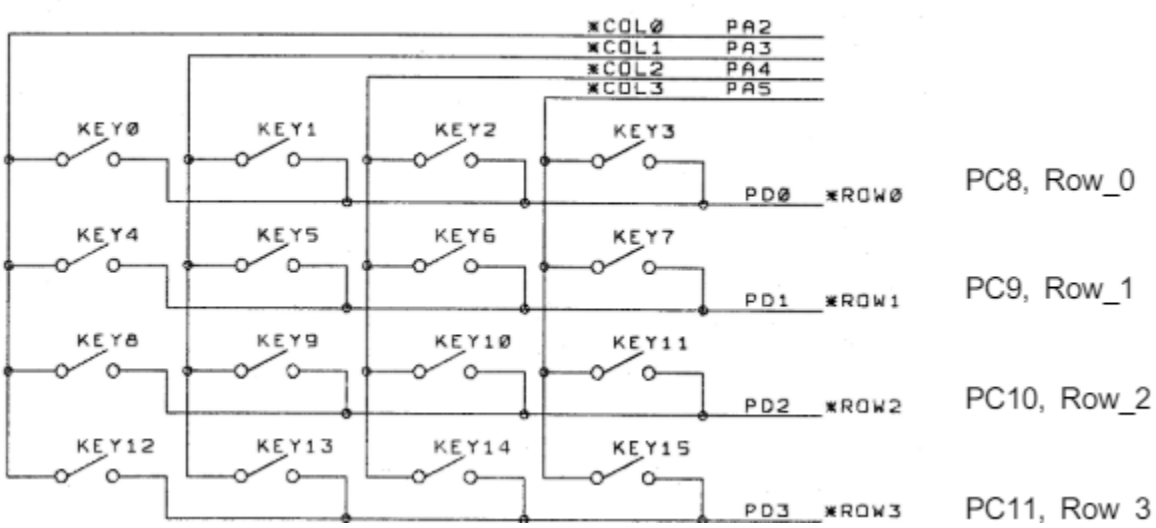
STM32F446 Block Diagram



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.

PB12 Col_0 PB13 Col_1 PB14 Col_2 PB15 Col_3



- **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.

Codul aplicației

main.c

```
#include "stm32f4xx.h"

#include <stdio.h>

#include <string.h>


#include "Types.h"

#include "Drivers.h"


//---------------------------------------

#define RS 1    /* BIT0 mask for reg select */

#define EN 2    /* BIT1 mask for E */

#define CORRECT_PASSWORD 1234
```

```
//-----------------------------------------

int corect[4]={2,2,2,2};
        int tries=3;
        int key=0;
        int count=0;
        int k=0;



void delay(void);
void delayMs(int);
void LCD_nibble_write(char data, unsigned char control);
void LCD_command(unsigned char command);
void LCD_data(char data);
void LCD_init(void);
void SPI1_write(unsigned char data);
int readRows(void);
int getPasswordFromKeypad(void);
void outputEnableCols(char n);
void writeCols(char n);
void writeLEDs(char n);
void keypad_init(void);
char keypad_getkey(void);
void writeStringLCD(char *line);
void newLine(unsigned int size);
```

```
void stopProgram();
int len =4;
U8 TimerCountDown_U16 = SECONDS_TO_COUNT_U8;
U8 PrintInCycleMode_U8 = 0;

void PeriphInit(void)
{
	__disable_irq();
	// Configure PB[7..4] as output
	RCC->AHB1ENR |= 0;                          /* Enable GPIOB clock */
	GPIOB->MODER &= 0; /* Reset GPIOB PB[7..4]  */
	GPIOB->MODER |= 0;          /* Set GPIOB PB[7..4]  as ouput */

	// Configure PC[11..8] as input
	RCC->AHB1ENR |= 0;                          /* Enable GPIOC clock */
	GPIOC->MODER &= 0; /* Reset GPIOC PC[11..8]  for input mode */

	// Configure PB[15..12] port as input and enables pull-ups
	GPIOB->MODER &= 0; /* Reset GPIOB PB[15..12]  */
	GPIOB->PUPDR |= 0;          /* Enable pull-ups on GPIOB PB[15..12]  */
	USART2_init();
	__enable_irq();
}

void citire()
{
 while(k<4)
   {
```

```
    while((key = keypad_getkey()) == 0);
        if(key != corect[k]){}
        else
        {
            ++count;
        }
        ++k;
        while(keypad_getkey() != 0);
    }
}

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(500);
if(tries>0)
{
        citire();
        if(count==4)
        {
                /* clear LCD display */
                LCD_command(1);
                delayMs(500);

                //writeLEDs(0x0);
                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;
                count=0;
                break;
                }
                else
```

```
        {
                /* clear LCD display */
                LCD_command(1);
                delayMs(500);

                --tries;
                LCD_data('W');
                LCD_data('R');
                LCD_data('O');
                LCD_data('N');
                LCD_data('G');
                writeLEDs(0xF);
                k=0;
                count=0;
                delayMs(500);
                LCD_command(1);
                }
        }
else
{
        while(1){
        LCD_command(1);
        delayMs(500);
        LCD_data('L');
        LCD_data('O');
        LCD_data('C');
        LCD_data('K');
        LCD_data('E');
```

```
                        LCD_data('D');
                        LCD_data('!');
                        delayMs(5000);
                        LCD_command(1);
                        }
                        }
        }

}
//-------------------------------------------------------------------------
/* configure SPI1 and the associated GPIO pins */
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);
}

/* This function enables slave select, writes one byte to SPI1, */
/* wait for transmit complete and deassert slave select. */
void SPI1_write(unsigned char data) {
    while (!(SPI1->SR & 2)) {}      /* wait until Transfer buffer Empty */
    GPIOA->BSRR = 0x10000000;       /* assert slave select */
    SPI1->DR = data;                /* write data */
    while (SPI1->SR & 0x80) {}      /* wait for transmission done */
    GPIOA->BSRR = 0x00001000;       /* deassert slave select */
}
/* 16 MHz SYSCLK */
void delayMs(int n) {
    int i;
```

```
    for (; n > 0; n--)
        for (i = 0; i < 3195; i++) ;
}
void writeStringLCD(char *line) {
    for(unsigned int i=0; i < strlen(line); i++)
    {
        LCD_data(line[i]);
    }
}
void newLine(unsigned int size){
    for(unsigned int i=0; i < 40-size; i++)
    {
        LCD_data(' ');
    }
}
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 */
}
```

```
/* enable columns according to bit 3-0 of the parameter n */
void outputEnableCols(char n) {
   GPIOB->MODER &= ~0xFF000000;    /* clear pin mode */

   /* make the pin output according to n */
        if (n & 1)
                GPIOB->MODER |=  0x01000000;
   if (n & 2)
                GPIOB->MODER |=  0x04000000;
        if (n & 4)
                GPIOB->MODER |=  0x10000000;
        if (n & 1 << 3)
                GPIOB->MODER |=  0x40000000;
}

/* write columns high or low according to bit 3-0 of the parameter n */
void writeCols(char n) {
   GPIOB->BSRR = 0xF0000000;   // turn off all column pins
   GPIOB->BSRR = n << 12;      // turn on column pins
}
/* read rows and return them in bit 3-0 */
int readRows(void) {
        return (GPIOC->IDR & 0x0F00) >> 8;
}
void writeLEDs(char n) {
   GPIOB->BSRR = 0x00F00000;   // turn off all LEDs
   GPIOB->BSRR = n << 4;       // turn on LEDs
}
```

```
/* system clock at 16 MHz delay about 100 us */
void delay(void) {
        int j;
        for (j = 0; j < 300; j++);      /* do nothing */
}
/* This function intializes the pins connected to the keypad. */
void keypad_init(void) {
        /* make rows input first */
   RCC->AHB1ENR |=  4;            /* enable GPIOC clock */
   GPIOC->MODER &= ~0x00FF0000;    /* clear pin mode */

        /* make columns input */
   RCC->AHB1ENR |=  2;            /* enable GPIOB clock */
   GPIOB->MODER &= ~0xFF000000;    /* clear pin mode */
}
//---------------------------------------------------------------------------
/* ISRs */
void TIM2_IRQHandler(void)
{
        TIM2->SR = 0; /* clear UIF */
        TimerCountDown_U16--;
        PrintInCycleMode_U8 = 1; /* time to print */
}
void SysTick_Handler(void)
{
        GPIOB->ODR ^= 0x00000020;
}
```

Drivers.c

```
#include "Drivers.h"
#include "stm32f4xx.h"

/* DRIVER C*/
#define BOARD_SIZE 3
/* TIM 2 as a general timer for cyclic messages */
void InitPeriodicTimer(U16 timeInMs)
{
  /* setup TIM2 */
  RCC->APB1ENR |= 1;        /* enable TIM2 clock */
  TIM2->PSC = 16000 - 1;    /* divided by 16000  to generate an up timer at 1kHz or 1ms*/
  TIM2->ARR = timeInMs - 1; /* divided by TimeInMs to generate the diseired timeout */
  TIM2->CR1 = 1;            /* enable counter */
  TIM2->DIER |= 1;          /* enable UIE */
  NVIC_EnableIRQ(TIM2_IRQn); /* enable interrupt in NVIC */
  /* setup main board LED to togle each 500 ms for periodic messages */
  RCC->AHB1ENR |= 1; /* enable GPIOA clock */

  GPIOA->MODER &= ~0x00000C00; /* clear the led */
  GPIOA->MODER |= 0x00000400;  /* set the mod as output */
}
void StopPeriodicTimer(void)
{
  RCC->APB1ENR &= ~1;         /* disable TIM2 clock */
  TIM2->CR1 = 0;              /* disable counter */
  TIM2->DIER &= ~1;           /* disable UIE */
  NVIC_DisableIRQ(TIM2_IRQn); /* disable interrupt in NVIC */}
```

```
/* configure SPI1 and the associated GPIO pins */
/* This function enables slave select, writes one byte to SPI1, */
/* wait for transmit complete and deassert slave select. */

void USART2_init (void) {
   RCC->AHB1ENR |= 1;          /* Enable GPIOA clock */
   RCC->APB1ENR |= 0x20000;    /* Enable USART2 clock */
   /* Configure PA2 and PA3 for USART2_TX and USART2_RX */
   GPIOA->AFR[0] &= ~0xFF00;
   GPIOA->AFR[0] |=  0x7700;   /* alt7 for USART2 */
   GPIOA->MODER  &= ~0x00F0;
   GPIOA->MODER  |=  0x00A0;   /* enable alternate function for PA2 and PA3 */
   USART2->BRR = 0x008B;       /* 115200 baud @ 16 MHz */
   USART2->CR1 = 0x000C;       /* enable Tx/Rx, 8-bit data */
   USART2->CR2 = 0x0000;       /* 1 stop bit */
   USART2->CR3 = 0x0000;       /* no flow control */
   USART2->CR1 |= 0x2000;      /* enable USART2*/
}
/* Write a character to USART2 */
int USART2_write (int ch) {
   while (!(USART2->SR & 0x0080)) {}   // wait until Tx buffer empty
   USART2->DR = (ch & 0xFF);
                  return ch;
}
/* Read a character from USART2 */
int USART2_read(void) {
   while (!(USART2->SR & 0x0020)) {}   // wait until char arrives
```

```
    return USART2->DR;
}
/* The code below is the interface to the C standard I/O library.
 * All the I/O are directed to the console, which is UART2.
 */
//struct __FILE { int handle; };
FILE __stdin  = {0};
FILE __stdout = {1};
FILE __stderr = {2};

/* Called by C library console/file input
 * This function echoes the character received.
 * If the character is '\r', it is substituted by '\n'.
 */
int fgetc(FILE *f) {
    int c;
    c = USART2_read();      /* read the character from console */
    if (c == '\r') {        /* if '\r', after it is echoed, a '\n' is appended*/
        USART2_write(c);    /* echo */
        //c = '\n';
    }
    USART2_write(c);        /* echo */
    return c;
}
/* Called by C library console/file output */
int fputc(int c, FILE *f) {
    return USART2_write(c);  /* write the character to console */
}
```

Drivers.h

```
#ifndef DRIVERS_H
#define DRIVERS_H

#include "Types.h"
#include "stm32f4xx.h"
#define PERIODIC_TIMER_VALUE_500US_U16 500

void InitPeriodicTimer(U16 timeInMs);
void StopPeriodicTimer(void);
void SPI1_init(void);
void SPI1_write(U8 data);

void USART2_init(void);
int USART2_write(int c);
int USART2_read(void);

#endif /* DRIVERS_H */
```

Types.h

```
#ifndef TYPES_H
#define TYPES_H

    #include "stm32f4xx.h"
    #include <stdio.h>
    #include <string.h>
```

```
	typedef uint8_t U8;
	typedef uint16_t U16;
	typedef uint32_t U32;

	typedef int16_t I16;

#define MAX_PACKET_STR_SIZE ((U8)(15u))
#define MAX_NR_OF_MSG_DATA_BYTES ((U8)(8u))
#define SECONDS_TO_COUNT_U8 ((U8)(10u))
#define MAX_NR_OF_CHANNELS ((U8)(18u))
#define PERIODIC_TIMER_VALUE_500US_U16 ((U16)(500u))
	struct UART_MSG_STRUCT
	{
		U8 len;
		U8 id;
		U8 data[MAX_NR_OF_MSG_DATA_BYTES - 1];
	};
	typedef struct UART_MSG_STRUCT UART_MSG_T;
	enum IDs_ENUM {
		SW_VERSION = 0,
		PORT_INPUT = 1,
		PORT_OUTPUT = 2,
		PORT_HEX = 3,
		TIMERS = 4,
		ADC_DAC = 5,
		IRQ = 6,
		SPI = 7,
		I2C = 8
```

```
	};
	typedef enum IDs_ENUM ID_TYPE;
	enum TIMER_IDs_ENUM {
		GENERAL_TIM3 = 0,
		TIM3_CH1_COMPARE = 1,
		TIM8_CH3_CAPTURE = 2,
		TIM8_CH1_PWM = 3,
		SYS_TICK_TIMER = 4
	};
	typedef enum TIMER_IDs_ENUM TIMER_ID_TYPE;
#endif /* TYPES_H */
```

Bibliografie

1. https://web.archive.org/web/20220122054809/http://www.microdigitaled.com/ARM/STM_ARM_books.html
2. Laborator Sisteme Încorporate