

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

PARKOVACÍ INFORMAČNÝ SYSTÉM

Programátorský manuál k semestrálnemu projektu z predmetu VRS

Bratislava 2016

Bc. Matej Vargovčík

Bc. Matej Ondrášik

Bc. Peter Vítek

Obsah

1	Štruktúra projektu	3
1.1	Základné údaje o projekte.....	3
1.2	Štruktúra projektu.....	3
2	Popis komponentov	4
2.1	Brána.....	4
2.1.1	Servo motor.....	4
2.1.2	Ovládanie brány.....	5
2.2	Displej.....	6
2.3	Ultrazvukové senzory.....	8
2.4	Hlavná slučka programu.....	9

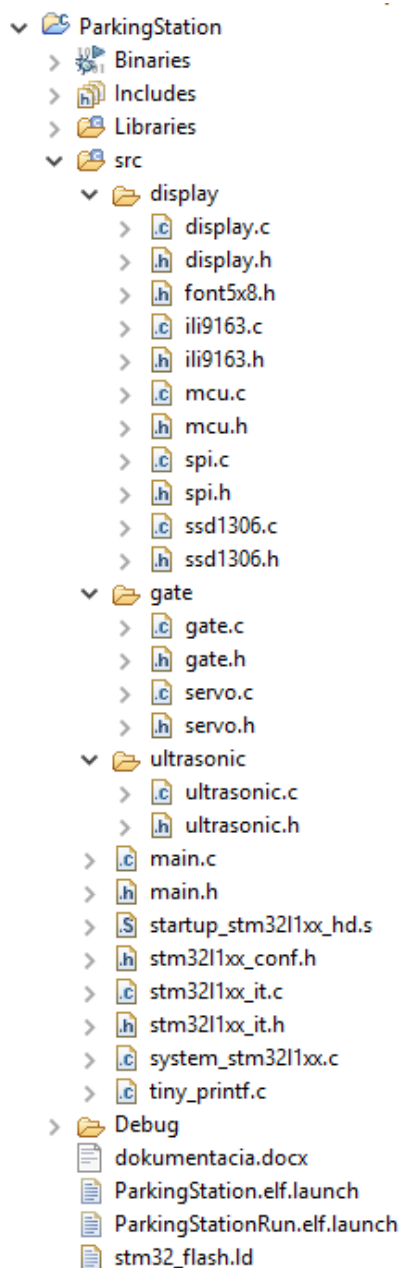
1 Štruktúra projektu

1.1 Základné údaje o projekte

Projekt je vytvorený v Atollic Studiu pre mikroprocesor STML152RE. Jazyk projektu je C.

1.2 Štruktúra projektu

Projekt pozostáva z hlavnej slučky programu obsiahnutej v súbore `main.c` a troch komponentov: brána, displej a ultrazvukové senzory pre parkovacie miesta.



2 Popis komponentov

V tejto kapitole popíšeme jednotlivé súčasti programu.

Pozn.: na meranie času sa používa globálny časovač `timer` Prejdený aktualizovaný v prerušení `SysTick_Handler`. Prejdený čas je určený rozdielom `timer - startTime`. `startTime` je hodnota časovača pri spustení merania. Ak je meranie času vypnuté, `startTime = -1`. Aby sa predchádzalo kolíziám s časovačom, v prípade, že hodnota vypnúťelného časovača je pri spustení merania -1, namiesto toho sa do `startTime` vloží hodnota -2.

2.1 Brána

Komponent brána sa nachádza v priečinku `gate` pozostáva z dvoch častí - ovládanie servo motora a ovládanie tlačidla na prechod bránou.

2.1.1 Servo motor

Servo motor je ovládaný hardvérovým PWM. Ovládanie je obsiahnuté v súbore `servo.h` a pozostáva z

- inicializácie (GPIO portu PB_6, časovača TIM4, PWM kanálu PWM1):

```
void initializeServoGPIO()
{
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);

    GPIO_InitTypeDef gpioStructure;
    gpioStructure.GPIO_Pin = GPIO_Pin_6;
    gpioStructure.GPIO_Mode = GPIO_Mode_AF;
    gpioStructure.GPIO_Speed = GPIO_Speed_40MHz;
    GPIO_Init(GPIOB, &gpioStructure);
}

void initializeTimer(int prescaler, int period)
{
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

    TIM_TimeBaseInitTypeDef timerInitStructure;
    timerInitStructure.TIM_Prescaler = prescaler;
    timerInitStructure.TIM_CounterMode = TIM_CounterMode_Up;
    timerInitStructure.TIM_Period = period;
    timerInitStructure.TIM_ClockDivision = TIM_CKD_DIV1;
    TIM_TimeBaseInit(TIM4, &timerInitStructure);
    TIM_Cmd(TIM4, ENABLE);
}

void initializePWMChannel()
{
    TIM_OCInitTypeDef outputChannelInit = {0,};
    outputChannelInit.TIM_OCMode = TIM_OCMode_PWM1;
    outputChannelInit.TIM_Pulse = 1500;
}
```

```

outputChannelInit.TIM_OutputState = TIM_OutputState_Enable;
outputChannelInit.TIM_OCPolarity = TIM_OCPolarity_High;

TIM_OC1Init(TIM4, &outputChannelInit);
TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_TIM4);
}

void initializeServo() {
    RCC_ClocksTypeDef RCC_Clocks;
    RCC_GetClocksFreq(&RCC_Clocks);
    initializeServoGPIO();
    initializeTimer(RCC_Clocks.HCLK_Frequency/1000000, 20000);
    initializePWMChannel();
}

```

- nastavenia žiadaného uhla otočenia motora

```

void setServoSignalLength(int us) {
    TIM4->CCR1 = us;
}

```

2.1.2 Ovládanie brány

Ovládanie brány je obsiahnuté v súbore `gate.c` a skladá sa z:

- inicializácie GPIO pinu tlačidla (PC_6)

```

void initializeGateButton() {
    GPIO_InitTypeDef gpioInitStruct;
    gpioInitStruct.GPIO_OType = GPIO_OType_PP;
    gpioInitStruct.GPIO_Speed = GPIO_Speed_400KHz;
    gpioInitStruct.GPIO_Mode = GPIO_Mode_IN;
    gpioInitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpioInitStruct.GPIO_Pin = GPIO_Pin_6;
    GPIO_Init(GPIOC, &gpioInitStruct);
}

```

- úkonov na ovládanie závery (otvorenie, zatvorenie)

```

void openGate() {
    setGateAngle(70);
}

```

```

void closeGate() {
    setGateAngle(0);
}

```

```

void setGateAngle(int degrees) {
    setServoSignalLength(degrees*1000/180 + 400);
}

```

- ovládania samotnej logiky brány:
 - jednoduché stlačenie tlačidla (nenasledované ďalším stlačením v priebehu ďalších 500ms) značí príchod auta do parkoviska. Ak je počet voľných parkovacích miest väčší ako 0, brána sa otvorí a zmenší sa počet parkovacích miest o 1

- dvojité stlačenie tlačidla značí odchod auta z parkoviska. Ak v parkovisku reálne auto je (počet voľných miest je menší ako kapacita parkoviska), brána sa otvorí a zvýši sa počet parkovacích miest o 1
- otvorenie brány trvá 1,5s a počas neho sú ignorované akékoľvek stlačenia tlačidla

```
void handleGate() {
    if (gateOpenTime == -1) {
        int pressed = (GPIOC->IDR & (uint16_t)(0b01<<6)) != 0;
        int clicked = pressed == 0 && gateButtonLastPressed == 1;
        gateButtonLastPressed = pressed;
        if (clicked) {
            if (gateButtonLastClickTime == -1) {
                gateButtonLastClickTime = timer;
                if (gateButtonLastClickTime == -1)
                    gateButtonLastClickTime = -2;
            }
            else {
                if (freePlacesCount < kUltrasonicSensorsCount) {
                    openGate();
                    gateOpenTime = timer;
                    if (gateOpenTime == -1)
                        gateOpenTime = -2;
                    freePlacesCount++;
                    displayPlacesCount(freePlacesCount);
                }
                gateButtonLastClickTime = -1;
            }
        }
        else if (gateButtonLastClickTime != -1 && timer - gateButtonLastClickTime > 500) {
            if (freePlacesCount > 0) {
                openGate();
                gateOpenTime = timer;
                if (gateOpenTime == -1)
                    gateOpenTime = -2;
                freePlacesCount--;
                displayPlacesCount(freePlacesCount);
            }
            gateButtonLastClickTime = -1;
        }
    }
    else {
        if (gateOpenTime != -1 && timer - gateOpenTime > 1500) {
            closeGate();
            gateOpenTime = -1;
        }
    }
}
```

2.2 Displej

SPI komunikácia a zobrazovanie na displeji boli prevzaté z príkladového projektu `spi_lcd`, preto budú popísané len veľmi stručne.

SPI prebieha na pinoch PA_5 (SCK), PA_6 (MISO, nie je pripojený) a PA_7 (MOSI), PA_8 (CD), PB_10 (CS - nepripojený, slave je iba jeden, preto je pripojený na GND), PA_9 (RES).

V zobrazovaní (`ili9163.c`) bolo oproti príkladovému projektu opravená funkcia `lcdFilledRectangle`, v ktorej bol zmenšený pravý a spodný okraj okna vyplňovania o 1

pixel, čím sa dosiahlo správne dokončenie obdĺžníka. Takisto bola pridaná funkcia `lcdDrawPictogram`, ktorá podobným spôsobom ako `lcdPutCh` vykreslí piktogram podľa sekvencie bitov, ktorá však spolu s výškou a šírkou prichádza ako vstupný parameter do funkcie (tieto parametre nie sú pevne dané ako font).

Vykresľovanie objektov pre aplikáciu je obsiahnuté v súbore `display.c`. Pozostáva z:

- inicializácie displeja a vykreslenia úvodnej grafiky (cesta, text, voľné parkovacie miesta)

```
void initializeDisplay() {
    initSPI2();
    initCD_Pin();
    initCS_Pin();
    initRES_Pin();

    lcdInitialise(LCD_ORIENTATION2);
    lcdClearDisplay(decodeRgbValue(0, 0, 0));

    lcdPutS("Free places count: ", lcdTextX(0), lcdTextY(0),
            decodeRgbValue(31, 31, 31), decodeRgbValue(0, 0, 0));

    lcdFilledRectangle(10, 70, 110, 80, decodeRgbValue(16, 16, 16));
    lcdFilledRectangle(10, 55, 20, 70, decodeRgbValue(16, 16, 16));
    lcdFilledRectangle(40, 55, 50, 70, decodeRgbValue(16, 16, 16));
    lcdFilledRectangle(70, 55, 80, 70, decodeRgbValue(16, 16, 16));
    lcdFilledRectangle(100, 55, 110, 70, decodeRgbValue(16, 16, 16));
    lcdFilledRectangle(10, 80, 20, 90, decodeRgbValue(16, 16, 16));

    for (int i = 0; i < placesCount; i++) {
        displayPlaceFree(i, 1);
    }
    displayPlacesCount(placesCount);
}
```

- metódy na vykreslenie obsadeného / voľného parkovacieho miesta

```
void displayPlaceFree(int place, int free) {
    if (free) {
        lcdFilledRectangle(placesX[place], placesY[place],
                           placesX[place]+placeWidth, placesY[place]+placeHeight,
                           decodeRgbValue(0, 31, 0));
    }
    else {
        lcdDrawPictogram(placesOrientation[place] ? car : carUpsideDown,
                           placesX[place], placesY[place], placeWidth, placeHeight,
                           decodeRgbValue(31, 0, 0), decodeRgbValue(0, 0, 0));
    }
}
```

- metódy na vykreslenie počtu voľných miest

```
void displayPlacesCount(int count) {
    char s[5];
    lcdPutS(itoa(count, s, 10), lcdTextX(19), lcdTextY(0),
            decodeRgbValue(31, 31, 31), decodeRgbValue(0, 0, 0));
}
```

2.3 Ultrazvukové senzory

Ultrazvukové senzory sú spracovávané v súbore `ultrasonic.h`. Každý senzor je definovaný GPIO pinmi, stavom obsadenosti a vlastnými meračmi času kvôli odstráneniu šumu.

```
typedef struct UltrasonicSensor {
    GPIO_TypeDef *gpioOut;
    GPIO_TypeDef *gpioIn;
    GPIO_TypeDef *gpioLed;
    uint16_t pinOut;
    uint16_t pinIn;
    uint16_t pinLed;
    int placeOccupied;
    int senseCount;
    int lastProximity;
    int placeFreeingStart;
} UltrasonicSensor;
```

Súbor pozostáva z:

- inicializácie jednotlivých senzorov a ich GPIO pinov (trigger, echo, indikačná LED)

```
void initializeUltrasonicSensors() {
    ultrasonicSensors[0] = ultrasonicSensor(GPIOA, GPIO_Pin_10, GPIOB, GPIO_Pin_3, GPIOB,
    GPIO_Pin_5);
    ultrasonicSensors[1] = ultrasonicSensor(GPIOB, GPIO_Pin_13, GPIOB, GPIO_Pin_14, GPIOB,
    GPIO_Pin_15);
    ultrasonicSensors[2] = ultrasonicSensor(GPIOB, GPIO_Pin_1, GPIOB, GPIO_Pin_2, GPIOB,
    GPIO_Pin_11);
    ultrasonicSensors[3] = ultrasonicSensor(GPIOA, GPIO_Pin_4, GPIOA, GPIO_Pin_1, GPIOA,
    GPIO_Pin_0);
```

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);
```

```
for (int i = 0; i < kUltrasonicSensorsCount; i++) {
    //trigger_sensor
    GPIO_InitTypeDef gpiolnitStruct;
    gpiolnitStruct.GPIO_OType = GPIO_OType_PP;
    gpiolnitStruct.GPIO_Speed = GPIO_Speed_400KHz;

    gpiolnitStruct.GPIO_Mode = GPIO_Mode_OUT;
    gpiolnitStruct.GPIO_Pin = ultrasonicSensors[i].pinOut;
    GPIO_Init(ultrasonicSensors[i].gpioOut, &gpiolnitStruct);

    gpiolnitStruct.GPIO_Mode = GPIO_Mode_IN;
    gpiolnitStruct.GPIO_Pin = ultrasonicSensors[i].pinIn;
    GPIO_Init(ultrasonicSensors[i].gpioIn, &gpiolnitStruct);

    gpiolnitStruct.GPIO_Mode = GPIO_Mode_OUT;
    gpiolnitStruct.GPIO_Pin = ultrasonicSensors[i].pinLed;
    GPIO_Init(ultrasonicSensors[i].gpioLed, &gpiolnitStruct);
}
}
```

- meraním obsadenosti parkovacieho miesta pred senzorom (spolu s odstránením šumu - miesto prestáva / začína byť obsadené až keď senzor nezaznamenal auto po dobu 500ms)


```

int measureProximity(int sensor) {

    GPIO_TypeDef *gpioOut = ultrasonicSensors[sensor].gpioOut;
    uint16_t pinOut = ultrasonicSensors[sensor].pinOut;
    GPIO_TypeDef *gpioIn = ultrasonicSensors[sensor].gpioIn;
    uint16_t pinIn = ultrasonicSensors[sensor].pinIn;

    int time0 = 0;
    int time1 = 0;
    GPIO_SetBits(gpioOut, pinOut);
    for(int i=0;i<10;i++);
    GPIO_ResetBits(gpioOut, pinOut);
    //wait for pulse on echo pin
    while(GPIO_ReadInputDataBit(gpioIn, pinIn)==0 && time0 < 500)
        time0++;
    //measure pulse width
    while(GPIO_ReadInputDataBit(gpioIn, pinIn)==1) time1++;
    if (time0 == 500)
        return ultrasonicSensors[sensor].placeOccupied;

    float distance=time1/20.0;

    if (distance < 7.0) {
        ultrasonicSensors[sensor].placeOccupied = 1;
        GPIO_SetBits(ultrasonicSensors[sensor].gpioLed, ultrasonicSensors[sensor].pinLed);
    }
    else if (ultrasonicSensors[sensor].placeOccupied) {
        if (ultrasonicSensors[sensor].lastProximity == 1)
            ultrasonicSensors[sensor].placeFreeingStart = timer;

        if (timer - ultrasonicSensors[sensor].placeFreeingStart > 500) {
            ultrasonicSensors[sensor].placeOccupied = 0;
            GPIO_ResetBits(ultrasonicSensors[sensor].gpioLed, ultrasonicSensors[sensor].pinLed);
        }
    }
    ultrasonicSensors[sensor].lastProximity = distance < 7.0;

    return ultrasonicSensors[sensor].placeOccupied;
}

```

2.4 Hlavná slučka programu

Hlavná funkcia programu pozostáva z inicializácie všetkých komponentov a hlavnej slučky. V hlavnej slučke sa prechádza všetkými ultrazvukovými senzormi a aktualizuje sa obsadenosť miest na displeji. Tiež sa volá metóda na ovládanie brány (ktorá podľa stlačenia tlačidla na vstup / výstup otvára bránu a aktualizuje počet voľných miest na parkovisku).

```

int main(void)
{
    RCC_ClocksTypeDef rccClocks;
    RCC_GetClocksFreq(&rccClocks);
    SysTick_Config(rccClocks.HCLK_Frequency/1000);

    timer = 0;
    initializeDisplay();
    initializeGate();
    initializeUltrasonicSensors();

    int placelsFree[kUltrasonicSensorsCount] = {1, 1, 1, 1};

    while (1)
    {

```

```
    for (int i = 0; i < kUltrasonicSensorsCount; i++) {  
        int isFree = !measureProximity(i);  
        if (isFree != placesIsFree[i]) {  
            placesIsFree[i] = isFree;  
            displayPlaceFree(i, isFree);  
        }  
    }  
    handleGate();  
}  
return 0;  
}
```