SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

inteligentné Parkovisko

Programátorský manuál k semestrálnemu projektu z predmetu VRS

**Bratislava 2016 Bc. Matej Vargovčík**

**Bc. Matej Ondrášik**

**Bc. Peter Vítek**

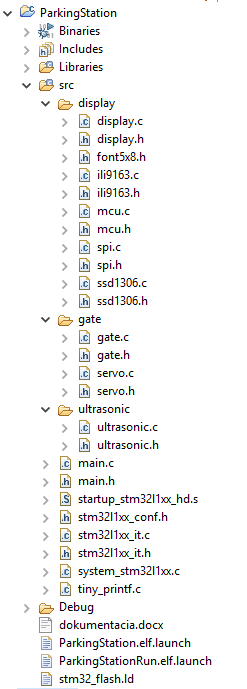
# Štruktúra projektu

## Základné údaje o projekte

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

## Štruktúra projektu

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



# 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 kolíziam s časovačom, v prípade, že hodnota vypnúteľného časovača je pri spustení merania -1, namiesto toho sa do startTime vloží hodnota -2.

## Brána

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

### 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;

}

### 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 závory (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;

}

}

}

## 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 obdlžníka. 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));

}

## 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 gpioInitStruct;

gpioInitStruct.GPIO\_OType = GPIO\_OType\_PP;

gpioInitStruct.GPIO\_Speed = GPIO\_Speed\_400KHz;

gpioInitStruct.GPIO\_Mode = GPIO\_Mode\_OUT;

gpioInitStruct.GPIO\_Pin = ultrasonicSensors[i].pinOut;

GPIO\_Init(ultrasonicSensors[i].gpioOut, &gpioInitStruct);

gpioInitStruct.GPIO\_Mode = GPIO\_Mode\_IN;

gpioInitStruct.GPIO\_Pin = ultrasonicSensors[i].pinIn;

GPIO\_Init(ultrasonicSensors[i].gpioIn, &gpioInitStruct);

gpioInitStruct.GPIO\_Mode = GPIO\_Mode\_OUT;

gpioInitStruct.GPIO\_Pin = ultrasonicSensors[i].pinLed;

GPIO\_Init(ultrasonicSensors[i].gpioLed, &gpioInitStruct);//

}

}

* 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) {

if (!ultrasonicSensors[sensor].placeOccupied) {

if (ultrasonicSensors[sensor].lastProximity == 0)

ultrasonicSensors[sensor].placeOccupyingStart = timer;

if (timer - ultrasonicSensors[sensor].placeOccupyingStart > 200) {

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;

}

## 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 placeIsFree[kUltrasonicSensorsCount] = {1, 1, 1, 1};

while (1)

{

for (int i = 0; i < kUltrasonicSensorsCount; i++) {

int isFree = !measureProximity(i);

if (isFree != placeIsFree[i]) {

placeIsFree[i] = isFree;

displayPlaceFree(i, isFree);

}

}

handleGate();

}

return 0;

}