Karolina Majstrović
0036468807

Pomoć pri parkiranju vozila u garažu

- Svima zainteresiranima
- Komunikacija sa senzorom
- Programiranje STM32F407 i spajanje komponenti
- Upute za povezivanje podsustava

7. svibanj 2015

creative commons (cc) BY-NC-SA

Sažetak

Tehnologiju koristimo da bi nam život bio lakši. U sklopu kolegija „Sustavi za praćenje i vođenje procesa" nastao je projekt pametne kuće. Studenti timskim radom rade na razvoju jedne takve kuće kako bi napravili kuću budućnosti. Poboljšanjem funkcija već postojećih dijelova kuće boravak ukućana se dovodi na novu razinu udobnosti.

Sadržaj

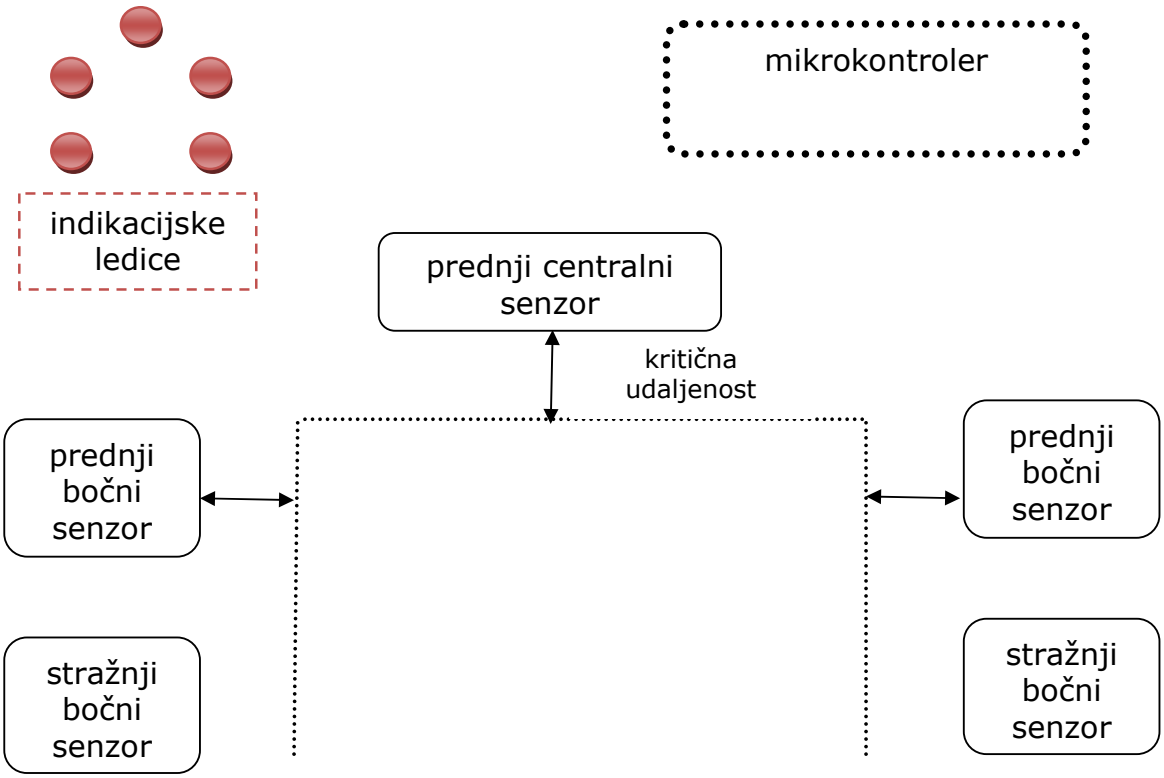
Ovaj seminarski rad je izrađen u okviru predmeta „Sustavi za praćenje i vođenje procesa" na Zavodu za elektroničke sustave i obradbu informacija, Fakulteta elektrotehnike i računarstva, Sveučilišta u Zagrebu.

Sadržaj ovog rada može se slobodno koristiti, umnožavati i distribuirati djelomično ili u cijelosti, uz uvjet da je uvijek naveden izvor dokumenta i autor, te da se time ne ostvaruje materijalna korist, a rezultirajuće djelo daje na korištenje pod istim ili sličnim ovakvim uvjetima.

1. Uvod

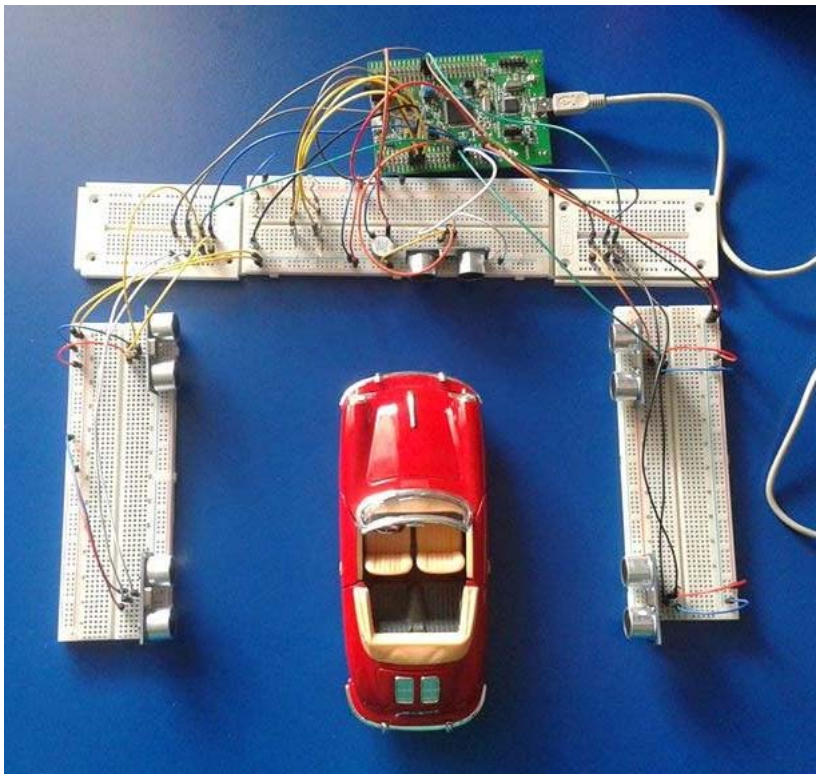
Mnogim vozačima je veliki problem parkirati vozilo u manju garažu. Tu im stvaraju problem različite prepreke i nevidljivost svih rubova automobila i prostorije. Postojeći sustavi koji olakšavaju parkiranje namijenjeni su direktnoj ugradnji u automobil, što ima izrazito visoku cijenu i time onemogućava dostupnost laganog parkiranja svima. Da bi se taj problem riješio, osmišljen je kvalitetan sustav izrazito niske cijene i jednostavne ugradnje. Cilj ovog sustava je na jednostavan način obavještavati vozača gdje je udaljenost automobila i prepreke (zida garaže) premala. Obavještavanje se vrši zujalicom i indikacijskom LED.

Svaka LED odgovara pojedinom senzoru. Tj. ako prednji centralni senzor detektira udaljenost manju od predefinirane vrijednosti LED koja označava mjesto gdje se nalazi taj senzor se uključuje, te zujalica počinje zujati kako bi vozača zvučno obavijestila.



Slika 1 Blok shema sustava

Na maketi sustav ovako izgleda:



Slika 2 Maketa sustava

1.1. Senzori

U projektu je korišten senzor udaljenosti HC-SR04. Mikrokontroleru podatak o udaljenosti šalje ne u obliku vrijednosti izražene u metrima, već je udaljenost poslana kao vrijeme trajanja impulsa. Udaljenost izražena u metrima se dobije iz formule:

Udaljenost = vrijeme visoke razine impulsa · 340 / 2

1.2. Zvučna i vizualna indikacija

Da bi korisnik uočio da se udaljenost između automobila i prepreke smanjila sustav mu to dojavljuje zvučno (zujalica) i vizualno (LED).

Zujalica zuji frekvencijom od 10 MHz, a određena LED se pali kada se smanji udaljenosti između odgovarajućeg senzora i automobila.

1.3. STM32F407

Mikrokontroler ARM arhitekture korišten je za izradu ovog projekta. On dio sustava koji odlučuje o svemu na temelju dobivenih podataka. Dok god je udaljenost veća od kritične mikrokontroler neće upaliti LED i zujalicu, no kada udaljenost između nekog senzora i automobila bude manja od kritične, tada naređuje paljenje odgovarajuće LED i zujalice. Nakon što je udaljenost ponovno veća od kritične LED i zujalica se gase.

1.4. UART

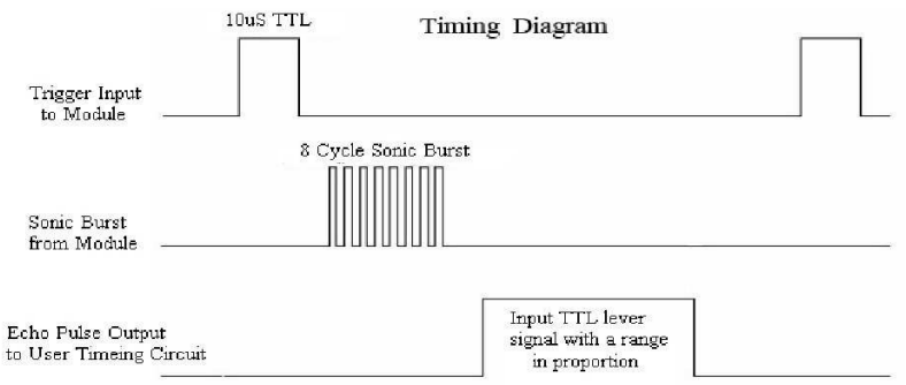
Serijska komunikacija s računalom je bila korištena pri otklanjanju grešaka u razvoju sustava.

2. Pomoć pri parkiranju vozila u garažu

U ovom poglavlju će biti riječi o implementaciji sustava za pomoć pri parkiranju vozila u garažu. Ovaj sustav čine komponente navedene na slici 1., a već objašnjene u prethodnom poglavlju.

2.1. Senzor

HC-SR04 zbog svoje specifičnosti komunikacije s mikrokontrolerom bilo je teže implementirati. To je zahtijevalo upotrebu brojača i prekidne rutine u mikrokontroleru. Kada mikrokontroler želi izmjeriti udaljenost između automobila i senzora, senzoru šalje impuls trajanja od barem 10 µs. Nakon primitka impulsa senzor šalje 8 perioda pravokutnog signala frekvencije 40 KHz i na izvod „*Echo*" senzor postavlja impuls duljine trajanja koja je proporcionalna udaljenosti između automobila i njega. Duljinu tog impulsa mjeri mikrokontroler i na temelju nje odlučuje je li potrebno obavijestiti korisnika da se automobil nalazi preblizu prepreke.



Slika 3 Komunikacija senzora i mikrokontrolera

2.2. Zvučna i svjetleća indikacija

Nakon što mikrokontroler primi podatak o udaljenosti odlučuje o obavještavanju vozača. LED su spojene na GPIO (engl. *General Purpose Input Output*) izvode koji su deklarirani kao izlazni. Zujalica je spojena na

PWM (engl. *Pulse Wide Modulation*) izvode. Pomoću PWM-a je moguće generirati pravokutni izlazni signal frekvencije od 10 kHz. Da bi zujalica zujala na njen izvod je potrebno dovesti promjenjivi signal (PWM) i GND. Da bi LED svijetlile na njihove izvode se dovodi signal s GPIO pinova i GND.

2.3. Kodovi

U ovom poglavlju je pregled koda koji je napisan za potrebe ovog projekta.

2.3.1. main.c

```
#include "stm32f4_discovery.h"
#include "spvp.h"

int main()
{
	SystemInit();
	TimInit();

  RCC_AHB1PeriphClockCmd(LED_RCC, ENABLE);
	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

	// kontrolne ledice
	InitOutput(LED_PORT, LED_CENTAR_SPRIJEDA |
			     LED_LIJEVA_SPRIJEDA |
			     LED_LIJEVA_STRAGA   |

LED_DESNA_SPRIJEDA  |
			     LED_DESNA_STRAGA    );

	// upravljacki signali za senzor --> output iz mikrokontrolera
	RCC_AHB1PeriphClockCmd(SENZOR_RCC, ENABLE);
	InitOutput(SENZOR_PORT, SENZOR_IZLAZ_CENTAR_SPRIJEDA |
			        SENZOR_IZLAZ_LIJEVO_SPRIJEDA |
			        SENZOR_IZLAZ_LIJEVO_STRAGA   |

SENZOR_IZLAZ_DESNO_SPRIJEDA  |
			        SENZOR_IZLAZ_DESNO_STRAGA    );
	Off(SENZOR_PORT, SENZOR_IZLAZ_CENTAR_SPRIJEDA);
	Off(SENZOR_PORT, SENZOR_IZLAZ_LIJEVO_SPRIJEDA);
	Off(SENZOR_PORT, SENZOR_IZLAZ_LIJEVO_STRAGA);
	Off(SENZOR_PORT, SENZOR_IZLAZ_DESNO_SPRIJEDA);
	Off(SENZOR_PORT, SENZOR_IZLAZ_DESNO_STRAGA);
	DelayMs(10);

	// signali za citanje vremena impulsa --> input u mikrokontroler
	RCC_AHB1PeriphClockCmd(SENZOR_RCC, ENABLE);
	InitInputEnable(SENZOR_PORT, SENZOR_ULAZ_CENTAR_SPRIJEDA);
	InitInputEnable(SENZOR_PORT, SENZOR_ULAZ_LIJEVO_SPRIJEDA);
	InitInputEnable(SENZOR_PORT, SENZOR_ULAZ_LIJEVO_STRAGA);
  InitInputEnable(SENZOR_PORT, SENZOR_ULAZ_DESNO_SPRIJEDA);
	InitInputEnable(SENZOR_PORT, SENZOR_ULAZ_DESNO_STRAGA);

	BuzzerInit();
```

```
      BuzzerOff();

  while (1)
  {
              CheckSensor(SENZOR_PORT, SENZOR_IZLAZ_CENTAR_SPRIJEDA, LED_PORT,
LED_CENTAR_SPRIJEDA,  SENZOR_PORT, SENZOR_ULAZ_CENTAR_SPRIJEDA);
              DelayMs(10);
              CheckSensor(SENZOR_PORT, SENZOR_IZLAZ_LIJEVO_SPRIJEDA, LED_PORT,
LED_LIJEVA_SPRIJEDA, SENZOR_PORT, SENZOR_ULAZ_LIJEVO_SPRIJEDA);
              DelayMs(10);
              CheckSensor(SENZOR_PORT, SENZOR_IZLAZ_LIJEVO_STRAGA,   LED_PORT,
LED_LIJEVA_STRAGA,   SENZOR_PORT, SENZOR_ULAZ_LIJEVO_STRAGA);
              DelayMs(10);
              CheckSensor(SENZOR_PORT, SENZOR_IZLAZ_DESNO_SPRIJEDA,  LED_PORT,
LED_DESNA_SPRIJEDA,  SENZOR_PORT, SENZOR_ULAZ_DESNO_SPRIJEDA);
              DelayMs(10);
              CheckSensor(SENZOR_PORT, SENZOR_IZLAZ_DESNO_STRAGA,    LED_PORT,
LED_DESNA_STRAGA,    SENZOR_PORT, SENZOR_ULAZ_DESNO_STRAGA);
              DelayMs(10);
  }
}

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line
number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line)
*/

  /* Infinite loop */
  while (1)
  {
  }
}
#endif

/****************** (C) COPYRIGHT 2011 STMicroelectronics *****END OF
FILE****/
```

2.3.2. spvp.h

```
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_tim.h"
#include "stm32f4xx_exti.h"

/*
 * Definicije kontrolnih LED koje ce pokazivati na mjesto gdje se auto
priblizio rubu garaze.
 */
#define LED_PORT            GPIOD
#define LED_RCC             RCC_AHB1Periph_GPIOD
```

```
#define LED_CENTAR_SPRIJEDA GPIO_Pin_0
#define LED_LIJEVA_SPRIJEDA GPIO_Pin_1
#define LED_LIJEVA_STRAGA   GPIO_Pin_2
#define LED_DESNA_SPRIJEDA  GPIO_Pin_3
#define LED_DESNA_STRAGA    GPIO_Pin_4

/*
 * Definicije pinova mikrokontrolera koji su ulazni (senzor na taj pin salje
 * impuls proporcinalan vremenu putovanja vala).
 */
#define SENZOR_PORT                                    GPIOE
#define SENZOR_RCC                 RCC_AHB1Periph_GPIOE
#define SENZOR_EXTI
      EXTI_PortSourceGPIOE

#define SENZOR_ULAZ_CENTAR_SPRIJEDA GPIO_Pin_0
#define SENZOR_ULAZ_LIJEVO_SPRIJEDA GPIO_Pin_1
#define SENZOR_ULAZ_LIJEVO_STRAGA   GPIO_Pin_2
#define SENZOR_ULAZ_DESNO_SPRIJEDA  GPIO_Pin_3
#define SENZOR_ULAZ_DESNO_STRAGA    GPIO_Pin_4

/*
 * Definicije pinova mikrokontrolera koji su izlazni (senzor na taj pin
dobiva impuls).
 */

#define SENZOR_IZLAZ_CENTAR_SPRIJEDA GPIO_Pin_5
#define SENZOR_IZLAZ_LIJEVO_SPRIJEDA GPIO_Pin_6
#define SENZOR_IZLAZ_LIJEVO_STRAGA   GPIO_Pin_7
#define SENZOR_IZLAZ_DESNO_SPRIJEDA  GPIO_Pin_8
#define SENZOR_IZLAZ_DESNO_STRAGA    GPIO_Pin_9

/*
 * timer/delay_s
 */
#define TIMER_RCC         RCC_APB1Periph_TIM3
#define TIMER_IRQn        TIM3_IRQn
#define TIMER_PRIVATE     TIM3

/*
 *  PWM timer
 */
#define PWM_RCC                              RCC_APB1Periph_TIM4
#define PWM_RCC_GPIO                   RCC_AHB1Periph_GPIOD
#define PWM_TIMER         TIM4
#define PWM_PORT                             GPIOD
#define PWM_SOURCE                     GPIO_PinSource12
#define PWM_AF                                    GPIO_AF_TIM4
#define PWM_PIN                                   GPIO_Pin_12

/*
*  Senzorske definicije
*/
#define FALSE                                     0
#define TRUE                                      1
#define CRITICAL_DISTANCE 10     // 10cm
#define ERROR_DISTANCE    50     // 50cm = 0.5m ako je udaljenost veca od 50m
#define ERROR_FACTOR      0      // 0cm
/*
 * Opcenite funkcije namjenjene za ovaj projekt.
 */
void InitOutput(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
void On(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
```

```
void Off(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
void OneToggle(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, int delay);
void InitInputEnable(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
int  ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin);
void TimInit();
void DelayMs(uint32_t ms);
void DelayUs(uint32_t us);
void CountingInit();
void CountingEnable();
void CountingDisable();
void CheckSensor(GPIO_TypeDef *GPIOx_from_uC, uint16_t GPIO_Pin_from_uC,
GPIO_TypeDef *GPIOx_led, uint16_t GPIO_Pin_led,GPIO_TypeDef *GPIOx_in_uC,
uint16_t GPIO_Pin_in_uC);
void BuzzerInit();
void BuzzerOn();
void BuzzerOff();
```

2.3.3. spvp.c

```
#include "spvp.h"
//extern volatile int measurement_complete;
//extern volatile int sensor_status;
volatile int sensor_status        = FALSE;
volatile int measurement_complete = TRUE;
/*
 * @brief  Inicijalizacija jednog ili vise GPIO pinova kao izlazni. Potrebno
je pozvati
          funkciju RCC_AHB1PeriphClock(...) prije inicijalizacije.
 * @param  GPIOx: x moze biti (A..I) za odabit GPIO porta
 * @param  GPIO_Pin: Definira koji redni broj pina/ova ce biti deklariran kao
izlazni
 * @retval None
 */
void InitOutput(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin){

      GPIO_InitTypeDef  GPIO_InitStructure;

      GPIO_InitStructure.GPIO_Pin   = GPIO_Pin;
      GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOx, &GPIO_InitStructure);

}

/*
 * @brief  Postavljanje izlaznog napona na 3.3V
 * @param  GPIOx: x moze biti (A..I) za odabit GPIO porta
 * @param  GPIO_Pin: Definira koji redni broj pina/ova ce biti ukljucen
 * @retval None
 */
void On(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin){

  GPIO_SetBits(GPIOx, GPIO_Pin);

}

/*
 * @brief  Postavljanje izlaznog napona na 0V
```

```
 * @param  GPIOx: x moze biti (A..I) za odabit GPIO porta
 * @param  GPIO_Pin: Definira koji redni broj pina/ova ce biti ukljucen
 * @retval None
 */
void Off(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin){
  GPIO_ResetBits(GPIOx, GPIO_Pin);
}

/*
 * @brief  Prvo se upali i nakon toga ugasi LED
 * @param  GPIOx: x moze biti (A..I) za odabit GPIO porta
 * @param  GPIO_Pin: Definira koji redni broj pina/ova ce biti ukljucen
 * @param  delay: vrijeme koje mora proteci izmedu paljenja i gasenja
 * @retval None
 */
void OneToggle(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin, int delay){
      On(GPIOx, GPIO_Pin);
      DelayMs(delay);
      Off(GPIOx, GPIO_Pin);
}

/*
 * @brief  Inicijalizacija jednog ili vise GPIO pinova kao ulazni
 * @param  GPIOx: x moze biti (A..I) za odabit GPIO porta
 * @param  GPIO_Pin: Definira koji redni broj pina/ova ce biti deklariran kao
izlazni
 * @retval None
 */
void InitInputEnable(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin){
  GPIO_InitTypeDef  GPIO_InitStructure;
      NVIC_InitTypeDef   NVIC_InitStructure;
      EXTI_InitTypeDef   EXTI_InitStructure;

      GPIO_InitStructure.GPIO_Pin   = GPIO_Pin;
      GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_PuPd  = GPIO_PuPd_UP;
      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_Init(GPIOx, &GPIO_InitStructure);

      switch(GPIO_Pin){
            case SENZOR_ULAZ_CENTAR_SPRIJEDA:
SYSCFG_EXTILineConfig(SENZOR_EXTI, EXTI_PinSource0);


      EXTI_InitStructure.EXTI_Line = EXTI_Line0;


      NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;

                                                      break;
            case SENZOR_ULAZ_LIJEVO_SPRIJEDA:
SYSCFG_EXTILineConfig(SENZOR_EXTI, EXTI_PinSource1);


      EXTI_InitStructure.EXTI_Line = EXTI_Line1;


      NVIC_InitStructure.NVIC_IRQChannel = EXTI1_IRQn;

                                                      break;
            case SENZOR_ULAZ_LIJEVO_STRAGA:
SYSCFG_EXTILineConfig(SENZOR_EXTI, EXTI_PinSource2);
```

```
	EXTI_InitStructure.EXTI_Line = EXTI_Line2;


	NVIC_InitStructure.NVIC_IRQChannel = EXTI2_IRQn;

                                                        break;
		case SENZOR_ULAZ_DESNO_SPRIJEDA:
SYSCFG_EXTILineConfig(SENZOR_EXTI, EXTI_PinSource3);


	EXTI_InitStructure.EXTI_Line = EXTI_Line3;


	NVIC_InitStructure.NVIC_IRQChannel = EXTI3_IRQn;

                                                        break;
		case SENZOR_ULAZ_DESNO_STRAGA:
SYSCFG_EXTILineConfig(SENZOR_EXTI, EXTI_PinSource4);


	EXTI_InitStructure.EXTI_Line = EXTI_Line4;


	NVIC_InitStructure.NVIC_IRQChannel = EXTI4_IRQn;

                                                        break;
	}
	EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
	EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising_Falling;
	EXTI_InitStructure.EXTI_LineCmd = ENABLE;
	EXTI_Init(&EXTI_InitStructure);

	NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
	NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
	NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
	NVIC_Init(&NVIC_InitStructure);

}

/*
 * @brief  Stanje GPIO pina
 * @param  GPIOx: x moze biti (A..I) za odabit GPIO porta
 * @param  GPIO_Pin: Definira koji redni broj pina/ova ce biti deklariran kao
izlazni
 * @retval None
 */
int ReadPin(GPIO_TypeDef *GPIOx, uint16_t GPIO_Pin){

    if (GPIO_ReadInputDataBit(GPIOx, GPIO_Pin)) {
                return 1;
                switch(GPIO_Pin){
                        case SENZOR_ULAZ_CENTAR_SPRIJEDA: On(LED_PORT,
LED_CENTAR_SPRIJEDA);

                                                        break;
                        case SENZOR_ULAZ_LIJEVO_SPRIJEDA: On(LED_PORT,
LED_LIJEVA_SPRIJEDA);

                                                        break;
                        case SENZOR_ULAZ_LIJEVO_STRAGA:   On(LED_PORT,
LED_LIJEVA_STRAGA);
```

```
                                                        break;
                        case SENZOR_ULAZ_DESNO_SPRIJEDA:  On(LED_PORT,
LED_DESNA_SPRIJEDA);

                                                        break;
                        case SENZOR_ULAZ_DESNO_STRAGA:    On(LED_PORT,
LED_DESNA_STRAGA);

                                                        break;
                }
    } else {
                return 0;
                switch(GPIO_Pin){
                        case SENZOR_ULAZ_CENTAR_SPRIJEDA: Off(LED_PORT,
LED_CENTAR_SPRIJEDA);

                                                              break;
                        case SENZOR_ULAZ_LIJEVO_SPRIJEDA: Off(LED_PORT,
LED_LIJEVA_SPRIJEDA);

                                                              break;
                        case SENZOR_ULAZ_LIJEVO_STRAGA:   Off(LED_PORT,
LED_LIJEVA_STRAGA);

                                                              break;
                        case SENZOR_ULAZ_DESNO_SPRIJEDA:  Off(LED_PORT,
LED_DESNA_SPRIJEDA);

                                                              break;
                        case SENZOR_ULAZ_DESNO_STRAGA:    Off(LED_PORT,
LED_DESNA_STRAGA);

                                                              break;
                }
    }
}

/*
 *
 */
void TimInit(){
      NVIC_InitTypeDef NVIC_InitStructure;
      TIM_TimeBaseInitTypeDef TIM_TimeBaseStruct;

      RCC_APB1PeriphClockCmd(TIMER_RCC, ENABLE);

  NVIC_InitStructure.NVIC_IRQChannel = TIMER_IRQn;
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

  TIM_TimeBaseStruct.TIM_Period = 65535;
  TIM_TimeBaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
  TIM_TimeBaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_TimeBaseStruct.TIM_Prescaler = 0;
  TIM_TimeBaseInit(TIMER_PRIVATE, &TIM_TimeBaseStruct);
}

/*
 *
 */
```

```
void DelayMs(uint32_t ms){

	TIM_OCInitTypeDef TIM_OCInitStruct;
	uint32_t PrescalerVal;

	TIM_SetCounter(TIMER_PRIVATE, 0);

	PrescalerVal = (uint16_t)((SystemCoreClock / 2) / 2000) - 1;
	TIM_PrescalerConfig(TIMER_PRIVATE, PrescalerVal,
TIM_PSCReloadMode_Immediate);

	TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_Timing;
	TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
	TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
	TIM_OCInitStruct.TIM_Pulse = ms * 2;
	TIM_OC1Init(TIMER_PRIVATE, &TIM_OCInitStruct);
	TIM_OC1PreloadConfig(TIMER_PRIVATE, TIM_OCPreload_Disable);

	TIM_ITConfig(TIMER_PRIVATE, TIM_IT_CC1, ENABLE);
	TIM_Cmd(TIMER_PRIVATE, ENABLE);

	while(!(TIMER_PRIVATE->SR & TIM_IT_CC1));

	TIM_Cmd(TIMER_PRIVATE, DISABLE);
}

/*
 *
 */
void DelayUs(uint32_t us){
  TIM_TimeBaseInitTypeDef TIM_TimeBaseStruct;
	uint32_t PrescalerVal;
	TIM_OCInitTypeDef TIM_OCInitStruct;

	TIM_SetCounter(TIMER_PRIVATE, 0);
  PrescalerVal = (uint16_t) ((SystemCoreClock / 2) / 2000000) - 1;
  TIM_PrescalerConfig(TIMER_PRIVATE, PrescalerVal,
TIM_PSCReloadMode_Immediate);

  TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_Timing;
  TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStruct.TIM_Pulse = us * 2;
  TIM_OC1Init(TIMER_PRIVATE, &TIM_OCInitStruct);

  TIM_ITConfig(TIMER_PRIVATE, TIM_IT_CC1, ENABLE);
  TIM_Cmd(TIMER_PRIVATE, ENABLE);

  while(!(TIMER_PRIVATE->SR & TIM_IT_CC1));

  TIM_Cmd(TIMER_PRIVATE, DISABLE);
}

/*
 *
 */
void TIM3_IRQHandler(void){
  TIM_ClearITPendingBit(TIMER_PRIVATE, TIM_IT_CC1);
}


void CountingInit(){
  TIM_OCInitTypeDef TIM_OCInitStruct;
```

```
	uint32_t PrescalerVal;

	TIM_TimeBaseInitTypeDef TIM_TimeBaseStruct;

	TIM_SetCounter(TIMER_PRIVATE, 0);
  PrescalerVal = (uint16_t) ((SystemCoreClock / 2) / 2000000) - 1; // 41
  TIM_PrescalerConfig(TIMER_PRIVATE, PrescalerVal,
TIM_PSCReloadMode_Immediate);

  TIM_OCInitStruct.TIM_OCMode = TIM_OCMode_Timing;
  TIM_OCInitStruct.TIM_OCPolarity = TIM_OCPolarity_High;
  TIM_OCInitStruct.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStruct.TIM_Pulse = 65535; // max = 65536
  TIM_OC1Init(TIMER_PRIVATE, &TIM_OCInitStruct);

  TIM_ITConfig(TIMER_PRIVATE, TIM_IT_CC1, ENABLE);
}
void CountingEnable(){
	TIM_Cmd(TIMER_PRIVATE, ENABLE);
}
void CountingDisable(){
  TIM_Cmd(TIMER_PRIVATE, DISABLE);
}
/*
 * Funkcija koja u najgorem slucaju broji do maksimalne vrijednosti ()
 */
void CheckSensor(GPIO_TypeDef *GPIOx_from_uC, uint16_t GPIO_Pin_from_uC,
GPIO_TypeDef *GPIOx_led, uint16_t GPIO_Pin_led,GPIO_TypeDef *GPIOx_in_uC,
uint16_t GPIO_Pin_in_uC){
	unsigned int time = 0;
	float distance;
	sensor_status        = FALSE;
	measurement_complete = FALSE;

	// slanje signala senzoru
	On(GPIOx_from_uC, GPIO_Pin_from_uC);
	DelayUs(100);
	Off(GPIOx_from_uC, GPIO_Pin_from_uC);

	// mjerenje signala
	CountingInit();
	CountingEnable();
	while(measurement_complete == FALSE){
		if(TIM_GetCounter(TIMER_PRIVATE) * 17 > ERROR_DISTANCE * 4000){
			break;
		}
	}
  CountingDisable();
	time = TIM_GetCounter(TIMER_PRIVATE);   // distance = time/2 * 340 / 2
	distance = CRITICAL_DISTANCE * 4;
	distance = distance + 10;
  if(time * 17 < CRITICAL_DISTANCE * 4000){ // distance * 4 = time * 340
		On(GPIOx_led, GPIO_Pin_led);
		BuzzerInit();
	//	DelayMs(10);
	}
	else{
		Off(GPIOx_led, GPIO_Pin_led);
		BuzzerOff();
//		DelayMs(10);
	}

	return;
```

```
}


void BuzzerInit(){
      TIM_TimeBaseInitTypeDef TIM_BaseStruct;
      TIM_OCInitTypeDef TIM_OCStruct;
      GPIO_InitTypeDef GPIO_InitStruct;

  RCC_APB1PeriphClockCmd(PWM_RCC, ENABLE);
      RCC_AHB1PeriphClockCmd(PWM_RCC_GPIO, ENABLE);

      TIM_BaseStruct.TIM_Prescaler = 0;
      TIM_BaseStruct.TIM_CounterMode = TIM_CounterMode_Up;
      TIM_BaseStruct.TIM_Period = 8399; /* 10kHz PWM */
  TIM_BaseStruct.TIM_ClockDivision = TIM_CKD_DIV1;
  TIM_BaseStruct.TIM_RepetitionCounter = 0;
  TIM_TimeBaseInit(PWM_TIMER, &TIM_BaseStruct);
  //TIM_Cmd(TIM4, ENABLE);

      TIM_OCStruct.TIM_OCMode = TIM_OCMode_PWM2;
  TIM_OCStruct.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCStruct.TIM_OCPolarity = TIM_OCPolarity_Low;
      TIM_OCStruct.TIM_Pulse =  4199; /* 25% duty cycle */
      TIM_OCStruct.TIM_OCIdleState = TIM_OCIdleState_Reset;
  TIM_OC1Init(TIM4, &TIM_OCStruct);
  TIM_OC1PreloadConfig(PWM_TIMER, TIM_OCPreload_Enable);

      GPIO_PinAFConfig(PWM_PORT, PWM_SOURCE, PWM_AF);

    /* Set pins */
  GPIO_InitStruct.GPIO_Pin = PWM_PIN;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(PWM_PORT, &GPIO_InitStruct);

}

void BuzzerOn(){
      GPIO_InitTypeDef GPIO_InitStruct;

      GPIO_InitStruct.GPIO_Pin = PWM_PIN;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(PWM_PORT, &GPIO_InitStruct);
      TIM_Cmd(TIM4, ENABLE);

}

void BuzzerOff(){
      GPIO_InitTypeDef GPIO_InitStruct;
      TIM_Cmd(TIM4, DISABLE);

      GPIO_InitStruct.GPIO_Pin = PWM_PIN;
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_Init(PWM_PORT, &GPIO_InitStruct);
      Off(PWM_PORT, PWM_PIN);}
```

Zaključak

Ovaj sustav je veoma praktičan, jeftin i jednostavan za ugradnju. Olakšava parkiranje vozačima na mjestima gdje je vidljivost cijele garaže ograničena. Svojim načinima obavještavanja omogućava vozaču da se ne mora koncentrirati samo na LED koje se nalaze na zidu, već i zvučno doznaje da je udaljenost premala.

3. Literatura

[1] http://www.keil.com/download/docs/352.asp

[2] http://visualgdb.com/tutorials/arm/stm32/timers/

[3] http://www.micropik.com/PDF/HCSR04.pdf

4. Pojmovnik

Pojam	Kratko objašnjenje	Više informacija potražite na
PWM	Engl. *Pulse Wide Modulation*. Generiranje pravokutnog signala na izvodima mikrokontrolera	http://en.wikipedia.org/wiki/Pulse-width_modulation
GPIO	Engl. *General Purpose Input Output*. Izvodi mikrokontrolera koji mogu biti ulazni i izlazni.	http://en.wikipedia.org/wiki/General-purpose_input/output
STM32F407	Mikrokontroler koji je „mozak" ovog sustava.	http://en.wikipedia.org/wiki/STM32#Discovery_boards
HC-SR04	Senzor udaljenosti.	http://www.micropik.com/PDF/HCSR04.pdf
LED	Engl. *Light Emitting Diode*. Vizualna indikacija.	http://en.wikipedia.org/wiki/Light-emitting_diode
Zujalica	Zvučna indikacija.	http://en.wikipedia.org/wiki/Buzzer