ZAVRSNI PROJEKAT

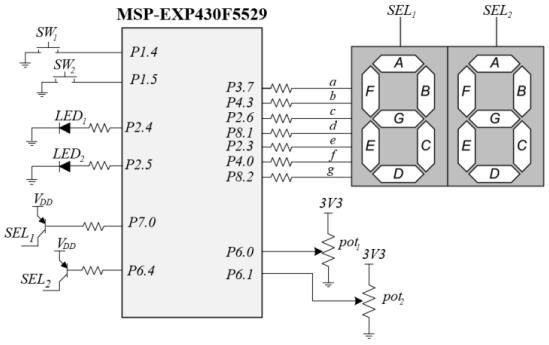
ELEKTROTEHNIČKI FAKULTET U BEOGRADU, ODSEK ZA SIGNALE
I SISTEME NAMENSKI RAČUNARSKI SISTEMI

Dragana Ninkovic 2019/0052

24. januar 2023.

1 Tekst zadatka 23.

Napisati program kojim se implementira funkcionalnost elektricne brave za bicikle. Otkljucavanje brave se vrsi unosenjem odgovarajuse dvocifrene sifre. Sifra se unosi preko dva kanala AD konvertora,gde jedan kanal sluzi za podesavanje jedne cifre. Prikaz trenutne kombinacije je na LED displeju. Ukoliko je uneta ispravna sifra, brava je otkljucana i svetli jedna LED dioda. Za vreme dok je brava otkljucana moguce je zapoceti unos nove sifre pritiskom na jedan taster. Kada se udje u rezim unosa nove sifre, potrebno je ukljuciti drugu diodu i pomeranjem potenciometara moguce je zadati novu sifru. Ponovnim pritiskom istog tastera,nova sifra se pamti.



Slika 1 Uproscena sema razvojnog sistema

2 Inicijalizacija promenljivih

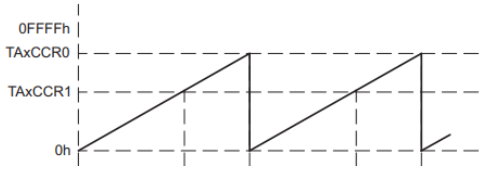
Oznaka volatile se stavlja da bi se zabranilo kompajleru da vrsi optimizacije, da se ne bi desilo da se u nekom od prekida menja promenljiva bez da kompajler to primeti i u main-u nastane problem prilikom optimizacije.
Opisi promenljivih dati su u komentarima.

```
#include <msp430.h>
#include "ETF_5529_HAL/hal_7seg.h"

/**
 * @file   main.c
 * @brief  bicycle lock
 *
 * In this example ADC ISR is used to read conversion results from two
     channels A0 i A1.
 * Every channel is converted to one cipher and displayed on digital
     display
 * If cipher is correct diode P2.4 is on, otherwise it is off. If cipher
     is correct it is possible to
 * change cipher by pressing the button, moving potentiometers. To save
     new cipher, button is pressed again.
 * During cipher change, diode P2.5 is on.
 *
 * @date   2023
 * @author Dragana Ninkovic 2019/0052
 *
 */

/**
 * @brief display refresh period
 * display refresh period
 * our eye cannot see above 50Hz -> 2ms
 * approximately 32768 * 1.95/ 1000
 */
#define DISPLAY_REFRESH_PERIOD (63) /* ~2ms (1.95ms) */
/**
 * @brief period for button debouncing
 * approximately 32768 * 10/1000
 */
#define BUTTON_WAIT_PERIOD     (327) /* ~10ms */

/*
 * @brief cipher which should be guessed
 * low - lse
 * high - mse
 */
volatile uint8_t real_cipher_low = 3;
```

```
volatile uint8_t real_cipher_high = 0;
/*
 * @brief current cipher
 * low - lse
 * high - mse
 */
volatile uint8_t curr_cipher_low = 0;
volatile uint8_t curr_cipher_high = 0;

/*
 * @brief when to perform check of cipher
 * we dont have to check it all the time, only when cipher is changed
 *
 */
volatile uint8_t check = 1;
/*
 * @brief whether lock is unlocked
 *
 */
volatile uint8_t unlocked = 0;
/*
 * @brief whether owner is currently changing cipher
 *
 */
volatile uint8_t waiting_new_cipher = 0;
/*
 * @brief whether button press is detected
 *
 */
volatile uint8_t button_pressed = 0;
/*
 * @brief two displays
 *
 */
typedef enum{
    DISP1,
    DISP2
}active_display_t;
/*
 * @brief currently active display
 * changes displays with display refresh period
 */
volatile active_display_t activeDisplay;
```

Whatchdog timer konstantno nadgleda magistralu i proverava da li se nas sistem negde zaglavio i ako se neko vreme ne desava nista smatra da je doslo do zaglavljenja i resetuje sistem. S obzirom da mi razvijamo softver a ne instaliramo ga on moze da nam smeta i zato ga odmah na pocetku iskljucujemo.
Za rad sa displejom koristimo Harisovu biblioteku hal7seg unutar koje se uklju-

cuju odgovarajuci segmenti pomocu ulaza a,..., f oznacenih na semi.
Na pocetku se aktivira displej 1, a iskljucuje displej 2.
Koristimo tri tajmera. Tajmer 0 koristi se za kontinualno citanje ADC sa periodom 0.5s. Biramo mod set reset kao najintuitivniji. TA0CCR0 stavljamo na 0.5s jer se sa tom periodom desavati set, a reset stavljamo na 0.25s jer se u tom delu periode desava reset kao sto je prikazano na slikama 2 i 3. Takodje, ovaj tajmer se ukljucuje odmah ovde i broji sve vreme. Tajmere 1 i 2 koristimo za osvezavanje ekrana displeja odnosno za debaunsiranje dugmeta. Razlika je samo u tome sto ove tajmere ne ukljucujemo odmah, vec dozvoljavamo prekid i brojanje pocinje kada se taj prekid desi.



Slika 2 Podesavanje registara CCR0 i CCR1

Slika 3 Grafik za set reset mod

```
/**
 * main.c
 */
int main(void)
{
    WDTCTL = WDTPW | WDTHOLD; // stop watchdog timer

    // initialize 7seg
    HAL_7Seg_Init();
    HAL_7SEG_DISPLAY_1_ON;
    HAL_7SEG_DISPLAY_2_OFF;
    activeDisplay = DISP1;

    /*Timer is clocked by ACLK (32768Hz).
     * If we need period of X ms then nr of cycles that is written to
         CCR0 register
     * is X*32768/1000.*/

    // timer used for reading ADC
    // initialize timer A0
    TA0CCR0 = 16382;                    // period is 0.5 -> 32768*0.5
```

```
TA0CCTL1 = OUTMOD_3;                    // use set/reset mode
TA0CCR1 = 8191;                         // CCR1 = 1/2 CCR0 (half of
    period)
TA0CTL = TASSEL__ACLK | MC__UP;         // ACLK source, UP mode

// timer used for display multiplexing
 /* initialize Timer A1 */
 TA1CCR0    = DISPLAY_REFRESH_PERIOD;
 TA1CCTL0   = CCIE;
 TA1CTL     = TASSEL__ACLK | MC__UP;


 // timer for debouncing button
 /* initialize Timer A2 */
 TA2CCR0    = BUTTON_WAIT_PERIOD;
 TA2CCTL0   = CCIE;          // enable CCR0 interrupt
 TA2CTL     = TASSEL__ACLK;
```

ADC konverzija nam je potrebna zato sto imamo dva potenciometra koja predstavljaju analogne ulaze. S obzirom da kompjuter ne moze da radi sa analognim ulazima potrebno ih je konvertovati u digitalne pomocu AD konvertora. Prilikom inicijalizacije podesavaju se kanali za koje se vrsi konverzija - 0 i 1, omogucavaju se prekidi i stavljamo da se za sample time koristi perioda tajmera 0. Mode koji koristimo je repeat-sequence-of-channels mode jer radimo uzastopno vise ADC konverzija(nad svakih Ts sekundi) nad vise kanala(2).
Inicijalizacija dioda se radi tako sto u registru P2DIR oznacavamo da je u pitanju izlazni pin (stavljanjem odgovarajucih bita na 1), a zatim postavljamo pocetne vrednosti izlaza na 0.
Slicno tome inicijalizuje se i dugme samo sto moramo da postavimo i pullup otpornik da bi u otvorenom stanju vrednost bila definisana i da ne bismo imali preveliku struju. Ponovo omogucujemo prekid na silaznu ivicu.

```
// initialize ADC
P6SEL |= BIT0;            // set P6.0 for ADC
P6SEL |= BIT1;            // set P6.1 for ADC
ADC12CTL0 = ADC12ON;      // turn on ADC
ADC12CTL1 = ADC12SHS_1 | ADC12CONSEQ_3; // set SHS = 1 (TA0.0 used
    for SAMPCON) and repeat-sequence-of-channels mode
ADC12MCTL0 = ADC12INCH_1;           // select channel 1
ADC12MCTL1 = ADC12INCH_0 | ADC12EOS; // select channel 0 and
ADC12CTL0 |= ADC12ENC;              // enable conversion
ADC12IE |= ADC12IE0 | ADC12IE1;    // enable interrupt for MEM0
    and MEM1

// initialize diodes
P2DIR |= (BIT4 | BIT5);
P2OUT &= ~(BIT4|BIT5);// set P2.4 and P2.5 as out
```

```
    // initialize button
    P1DIR &= ~BIT4;          // set P1.4 as in
    P1REN |= BIT4;           // This is important because there is no
        PullUp Resistor on the board
    P1OUT |= BIT4;           // This is important because there is no
        PullUp Resistor on the board

    P1IES |= BIT4;           // set P1.4 irq as h->l transition
    P1IFG &= ~BIT4;          // clear P1.4 IFG
    P1IE  |= BIT4;           // enable P1.4 irq


  __enable_interrupt();
```

3 Glavni deo programa

Unutar glavnog programa nakon inicijalizacije radimo glavno procesiranje svih podataka. S obzirom da ne zelimo da sve vreme proveravamo sifru, promenljiva check nam sluzi da aktivira proveru sifre nakon sto je promenjen trenutni pokusaj sifre pomocu potenciometra a da nismo u stanju gde je korisnik vec pogodio sifru i odlucio da je promeni.
Ako je trenutna sifra pogodjena promenljiva unlocked koja sluzi kao flag za to da li je brava otkljucana ili ne se stavlja na 1, i dioda 4 se ukljucuje.
U suprotnom, dioda se iskljucuje a unlocked postaje 0. Bitno je vratiti check na 0 nakon sto smo izvrsili proveru ili ustanovili da ne treba izvrsiti proveru.
Druga provera uslova vezana je za promenu sifre. Ako je dugme pritisnuto i brava je otkljucana omogucava se promena sifre. Ako je u pitanju prvi pritisak dugmeta waiting for cipher ce biti 0 i tada ukljucujemo diodu i postavljamo ovu promenljivu na vrednost 1.
Ako ova promenljiva ima vrednost 1 to znaci da je u pitanju drugi stisak dugmeta, treba iskljuciti diodu, kao novu sifru upisati trenutnu sifru koja se ocitava sa potenciometara i vratiti ovu promenljivu u stanje 0.
Nakon svih provera bitno je oznaciti da je obradjen pritisak dugmeta.

```
while (1){
    // if cipher is changed and owner isn't changing cipher
    if((check == 1) && (waiting_new_cipher == 0)){
        if(real_cipher_low == curr_cipher_low && real_cipher_high ==
             curr_cipher_high ){// if cipher is correct
            unlocked = 1;// unlock
            P2OUT |= BIT4;// turn on diode
        }
        else{// if cipher is not correct
            unlocked = 0;// lock
            P2OUT &= ~BIT4;// turn off diode
        }
        check = 0;// check is done
    }
    else{
        check = 0;
    }
   // if button is pressed and lock is unlocked
   if((button_pressed == 1) && (unlocked == 1)){
       if(waiting_new_cipher == 0){// first press
            P2OUT |= BIT5;// turn on second diode
            waiting_new_cipher = 1;// wait for next button
         }
       else{// second press
            P2OUT &= ~BIT5;// turn off diode
            real_cipher_high = curr_cipher_high;// change cipher
            real_cipher_low = curr_cipher_low;
```

```
                waiting_new_cipher = 0;// detect end
               }
            button_pressed = 0;// press is processed
        }
        else{
            button_pressed = 0;
        }
    }

    return 0;
}
```

4 Prekidna rutina za debaunsiranje dugmeta

Zbog mehanicke prirode prekidaca, cesto se desi da prilikom pritiska dobijemo odskakanje tokom nekog kratkog vremena koje moze dovesti do lose detekcije. Zbog toga je potrebno sacekati neko vreme(ovo vreme meri tajmer od pritiska dugmeta kada ga iskljucimo do isteka predefinisanog vremena kada ga iskljucimo i proverimo uslov) i ustanoviti da li smo idalje sigurni da je doslo do pritiska(promene high to low). Ako jesmo onda tek obavestavamo glavni program da je doslo do prekida koji treba obraditi.
Tajmer se resetuje, flag prekidne rutine je na nama da obrisemo jer smo mi ti koji obradjujemo prekid i znamo kada je obrada zavrsena.

```
void __attribute__ ((interrupt(TIMER2_A0_VECTOR))) CCR0ISR (void)
{
    if ((P1IN & BIT4) == 0) // check if button is still pressed
    {
       button_pressed = 1;// when button is pressed flag it for main
    }
    TA2CTL &= ~(MC0 | MC1); // stop and clear timer
    TA2CTL |= TACLR;
    P1IFG &= ~BIT4;       // clear P1.4 flag
    P1IE |= BIT4;         // enable P1.4 interrupt
    return;
}
/**
 * @brief PORT1 Interrupt service routine
 *
 * ISR starts timer which will check button press
 */
void __attribute__ ((interrupt(PORT1_VECTOR))) P1ISR (void)
{
    if ((P1IFG & BIT4) != 0)     // check if P1.3 flag is set
    {
        /* start timer */
        TA2CTL |= MC__UP;
        P1IFG &= ~BIT4;          // clear P1.4 flag
        P1IE &= ~BIT4;           // disable P1.4 interrupt
    }

    return;
}
```

5 Prekidna rutina za ADC konverziju

Prilikom ADC konverzije citaju se odgovarajuce lokacije. Siftujemo broj za 8 mesta ulevo da bismo dobili samo najvisa 4 bita, cime zapravo radimo NF filtriranje, jer nam trebaju samo brojevi 0 do 9, a najvisi biti se menjaju najmanjom periodom. Konvertujemo opseg 0-15 u opseg 0-9 kako bismo imali jednake intervale za svaku od cifara prilikom okretanja potenciometra.

```
void __attribute__ ((interrupt(ADC12_VECTOR))) ADC12ISR (void)
{
    switch(__even_in_range(ADC12IV,34))
    {
    case  0: break;                          // Vector 0:  No interrupt
    case  2: break;                          // Vector 2:  ADC overflow
    case  4: break;                          // Vector 4:  ADC timing
        overflow
    case  6: // Vector 6:  ADC12IFG0
          // read lower digit of cipher
          curr_cipher_low = (uint8_t)((ADC12MEM0>>8)*9.0/15.0);
          // check whether cipher is correct
          check = 1;

       break;
    case  8:                        // Vector 8:  ADC12IFG1
          // read higher digit of cipher
          curr_cipher_high = (uint8_t)((ADC12MEM1>>8)*9.0/15.0);
          // check whether cipher is correct
          check = 1;


       break;
    case 10: break;                          // Vector 10: ADC12IFG2
    case 12: break;                          // Vector 12: ADC12IFG3
    case 14: break;                          // Vector 14: ADC12IFG4
    case 16: break;                          // Vector 16: ADC12IFG5
    case 18: break;                          // Vector 18: ADC12IFG6
    case 20: break;                          // Vector 20: ADC12IFG7
    case 22: break;                          // Vector 22: ADC12IFG8
    case 24: break;                          // Vector 24: ADC12IFG9
    case 26: break;                          // Vector 26: ADC12IFG10
    case 28: break;                          // Vector 28: ADC12IFG11
    case 30: break;                          // Vector 30: ADC12IFG12
    case 32: break;                          // Vector 32: ADC12IFG13
    case 34: break;                          // Vector 34: ADC12IFG14
    default: break;
    }
}
```

6 Prekidna rutina za multipleksiranje displeja

Da ne bismo koristili 18 ulaza da povezemo dva displeja sa mikrokontrolerom, koristi se multipleksiranje displeja. Koristimo konfiguraciju sa zajednickom anodom. Ovim su sve anode povezane na visok naponski nivo a postavljanjem nula se ukljucuju razliciti segmenti na displeju. Koristimo 8 ulaza za segmente oba displeja, i 2 ulaza da izaberemo da li ce odgovarajuci displej biti uklucen. Ovime je u jednom trenutku ukljucen samo jedan displej ali prividno izgleda kao da su ukljucena oba jer ih smenjujemo sa frekvencijom 50 Hz a ljudi ne mogu da primete promenu vecu od 40 Hz.

```
// display multiplexing
void __attribute__ ((interrupt(TIMER1_A0_VECTOR))) TAIEISR (void)
{
    switch (activeDisplay)
    {
    case DISP1:
        HAL_7SEG_DISPLAY_1_ON;
        HAL_7SEG_DISPLAY_2_OFF;

        HAL_7Seg_WriteDigit(curr_cipher_high);

        activeDisplay = DISP2;
        break;
    case DISP2:
        HAL_7SEG_DISPLAY_1_OFF;
        HAL_7SEG_DISPLAY_2_ON;

        HAL_7Seg_WriteDigit(curr_cipher_low);

        activeDisplay = DISP1;
        break;
    }
}
```