Univerza v Ljubljani
Fakulteta za elektrotehniko

SEMINARSKA NALOGA

Baterijski modul za industrijski števec

pri predmetu

ELEKTRONSKA VEZJA

Predavatelj: prof. dr. Marko Topič

Avtor:
Anton Kljun
FE-UNI elektronika

V Kranju, februar 2007

KAZALO VSEBINE

KAZALO SLIK

UVOD

Kot štipendist v podjetju Iskraemeco sem dobil nalogo načrtati modul, ki bo omogočal baterijsko napajanje industrijskega števca družine MT830, ko števec ni zunanje (omrežno) napajan.

Veliko naročnikov želi imeti vključeno to podporo, saj so zahteve po funkcionalnosti, kakovosti in varnosti izdelkov vedno večje. S tem modulom bo omogočena komunikacija s števcem, ko pride do izpada omrežja oz. ko števec nima zunanjega napajanja. Poleg primarne naloge zagotavljanja delovanja števca, pa je omogočen tudi nadzor nad nedovoljenim mehanskim posegom v sam števec, ter detekcija močnega magnetnega polja. Števec te dogodke registrira, tako da so vse morebitne zlorabe zabeležene.

ZAHTEVE

Pri načrtovanju modula je potrebno ugoditi zahtevam naročnika, da izdelek brezhibno funkcionira za predpisano dobo (10 let). To pomeni skrbno načrtovanje s čim manjšo porabo energije. Na trgu so prav v ta namen (embedded system) posebej razviti mikrokontrolerji, ki omogočajo majhno porabo energije z pravilno uporabo tehnik programiranja (low power mode). Eden izmed teh mikorkontrolerjev je tudi družina MSP430 proizvajalca Texas Instruments.

Zahteva je, da naj se števec zažene, ko pritisnemo na menijsko tipko, prav tako naj števec deluje, ko tipko tiščimo ali zadržimo, saj se s tem premikamo po menijih števca (tako je definiran program v števcu). Kadar je prisotna komunikacija, mora modul ravno tako zagotoviti delovanje števca. Števec naj se ugasne po 4 sekundah, ko ni prisotne komunikacije, oz. ko ni nobena tipka sklenjena, razsklenjena ali zadržana. Vse to seveda le v primeru, ko števec ni zunanje napajan.

Ker gre za izdelek, ki bo izdelan v večdeset tisoč proizvodih, naj bo tudi kar se da preprost, poceni, funkcionalen, nadgradljiv in prilagodljiv za prenos na drug tip števca.

IZVEDBA VEZJA

Sama izvedba naj bo kot že rečeno čimbolj preprosta in hkrati miniaturna, saj prostora v števcu ni prav veliko. Mikrokontroler naj poskrbi za detekcijo spremembe signalov iz dveh menijskih tipk (TIP_PA, TIP_PO), dveh detekcijski sponk (VDOR_PO, VDOR_PR) in optične komunikacije (TXDO), le kadar ni zunanje napajan oz. kadar je signal (NMI), ki nam ga posreduje števec, aktiven (5V). Ko pride do prekinitve (interrupt), mikrokontroler spremeni izhodni signal (SWITCH), ki je vezan na stikalo MOS-FET izveden z FDC604P, na 0V in odpre stikalo. Ta vključi delovanje step-up pretvornika izvedenega z LT3460, ki pretvori baterijsko napetost za napajanje števca je iz 3,6V na 5,0V (80-90% izkoristek).

Tu je pomemben izbor baterije, katera naj zagotavlja delovanje v industrijskem okolju. Za primerno se izkaže LS14250 lithium-thionyl chloride (Li-SOCL2) baterija velikosti 1/2AA in kapacitivnosti 1000 mAh, z zagotavljanjem območja delovanja od -60°C do +85°C in življensko dobo 20 let. Odlikuje jo predvsem majhen efekt samopraznjenja. Plošča se lahko opremi z največ štirimi takimi baterijami. Edini problem, ki ga ima ta baterija je odzivni čas, kadar je dalj časa v stanju mirovanja, ko ni porabe. Ta problem je rešen z dodatnim kondenzatorjem C2, kateri poskrbi za energijo v tem primeru. Preko kondenzatorja C2 pa se modul tudi napaja ves čas, ko ima vezje zunanje napajanje in tako ne troši energije baterij. Za stabilno napajalno napetost mikrokontrolerja pa je dodan kondenzator C1.

Pazljivo je potrebno tudi razporediti elemente na vezju, saj s prevelikimi razdaljami in izpostavljenostjo industrijskemu okolju, kaj kmalu zaznamo neprijetne motnje, kar bo mikrokontroler zaznal kot spremembo signala in bo povzročil spremembo signala, ter tako nepotrebno trošil energijo. Pazimo predvsem pri postavitvi aktivnih elementov (tuljava, kondenzator), da so povezave čim krajše. Hardwaresko so na vhodih vgrajeni RC filtri za preprečevanje teh motenj, prav tako se lahko dodajo tudi softwareski filtri v sam program mikrokontrolerja po potrebi (če bodo rezultati iz EMC laboratorija to narekovali).

Sama narava problema zahteva tudi dobro poznavanje delovanja mikrokontrolerja, saj naj bi večinoma časa (ob pristotnosti zunanjega napajanja - omrežje) preživel v LPM (low power mode), kjer je poraba toka v rangu ~1uA. Mikrokontroler ima tudi nalogo spremljanja porabe energije, kar sporoči preko izhodnega signala BAT_ST števcu.

PROBLEMI IN REŠITVE

Seveda pa brez problemov pri načrtovanju ne gre. Z testiranjem na SMD protobord ploščici nisem imel večjih problemov, kar pa ni veljalo za prototip števca. Tu je bila sprememba izhoda pri pritistku na tipko le ~380mV namesto 3,6V, ko je tipka sklenjena oz. razsklenjena. Tako majhne spremembe mikrokontroler ne zazna kot interrupt. Po analizi problema se je izkazalo, da so na izbranem izhod v števcu, ki je namenjen temu signalu, zaščitne diode, ki so to napetost toliko oslabile.

Pojavi se vprašanje, ali zamenjati izhode na procesorju v števcu, ki nimajo te omejitve, ali pa ta signal ojačati. Ker je bil programski del modula modula prakatično že končan, smo se odločili, da ta problem rešimo z ojačanjem tega signala, a žal na račun porabe energije. Sprememba izhodov na že testiranem števcu, bi namreč spet prinesla veliko zamude s ponovnim preverjanjem in testiranjem, ter vsemi možnimi nadaljnimi zapleti pri tako kompleksnem vezju. Uporabil sem operacijski ojačevalec TLV2444 in določil upore v povratni zanki z ojačanjem A>10 da preide v nasičenje, kar zopet omogoči mikrokontrolerju detekcijo spremembe signala (interrupt).

IZBOLJŠAVE, NADGRADNJE

Ker gre za prvo verzijo funkcijskega prototipa, je bilo pričakovati še razne napake, a se je izkazalo, da manjka le en upor, potreben za krmiljenje signala pri debugiranju oz. programiranju preko JTAG vmesnika.

Veliko je še možnosti za nadgradnjo modula (dodaten flash pomnilnik za števec, RF komunikacija, razni senzorji, itd). Lahko se uporabijo še porti, ki jih zasede JTAG, vendar debugiranje na njih ne bo možno. Ker je bilo kar nekaj težav s števci na terenu, ko so jih z močnim magnetom poskušali onesposobiti, bo naslednja nadgradnja v modulu magnetni rele senzor, ki bo posredoval števcu ta dogodek, tako da bo ta poseg zabeležen. Prav tako je izkoristek zaradi padca napetosti na diodah slabši, zato bi bilo vredno razmisliti o bateriji z večjo kapacitivnostjo, ali pa da omogoča večji konstantni tok in bi jih lahko vezal zaporedno. Pri večji napetosti bi bil tudi izkoristek step-up pretvornika še malenkost boljši.

Glavni problem modula je trenutno poraba operacijskega ojačevalnika, ker troši tok iz baterije okoli 1mA! Ko bi bil števec enkrat vgrajen na svoje mesto, bi bilo delovanje modula še kar sprejemljivo, saj se baterije ne trošijo ob prisotni omrežni napetosti, vendar pa je to zelo nepredvidljiva varianta, česar si ne smemo privoščiti. Z dodatno logiko za reševanje te težave, pa bi se cena in kompleksnost modula že močno povečala.

Ker gre za večji projekt, se tudi same zahteve iz strani kupca še spreminjajo in dopolnjujejo. S tem smo pridobili nekaj na času razvoja. Vse izkušnje in dosedanje rešitve problemov nam pri načrtovanju ponujajo mnogo boljši rezultat, če se odločimo za spremembo osnovne plošče števca, kar bo vplivalo na precejšno poenostavitev in pocenitev celotnega števca.

OPIS DELOVANJA PROGRAMA

V modulu je uporabljena cenejša različica mikrokontrolerja MSP430F1122A. Ta omogoča detekcijo spremembe signala na input vhodih (P1.0-P1.3 in P2.0-P2.2) ob pogoju, da je signal NMI_CAL=0V (števec ni zunanje napajan). Izhodni signal SWITCH spremeni napetost na logično 0 in tako odpre FET stikalo. Ta zažene step-up pretvornik, ki napaja števec. Po določenem času, ki jo določimo s konstanto "repeatdelay", se SWITCH vrne na logično 1 ter ugasne števec. Ob interruptu, se program za 500ms aktivira in ne dovoli nadaljnih interruptov, dokler se ne vrne nazaj v "while (1)" zakno. Nato zažene ACLK v LPM3 in omogoči timer interrupt na vsakih 100ms, kar je definirano s frekvenco oscilatorja ~32 kHz in offsetom v timer zanki CCR2+=3277. Timer je aktiven toliko časa, dokler konstanta "repeatdelay" v timer interrupt zanki ne doseže vrednosti 0 in se nato ugasne.

Odzivni čas mikrokontrolerja na vhodni interrupt je ~83us. MSP430 bo večino časa deoval v LPM3, kjer je aktiven samo ACLK. V tem načinu delovanja še vedno omogoča interrupte, kar predstavlja zelo majhno porabo baterije (~1uA). To je velika prednost glede na »pooling« tehniko preverjanja spremembe signala, kjer bi moral biti števec v aktivnem stanju večinoma časa.

Porabo števca oz. stanje baterije merimo časovno s konstanto "batterytimer", kjer ena enota predstavlja časovno vrednost 100ms aktivnega delovanja števca. Potrebe po večji ločljivosti ni, saj dejansko stanje oz. zmogljivost baterije veliko bolj določajo drugi dejavniki kot so

temperatura, obremenitev baterije, frekvenca delovanja, itd. Program trenutno še nima napisanih SW filtrov za motnje na vhodni signal, kar lahko po potrebi dodam, če se bo tako izkazalo po opravljenih testnih v EMC laboratoriju. Prav tako na vezju še ni releja za detekcijo magnetnega polja.

PROGRAMSKA KODA

```
// ###########################################################################
// ###########################################################################
// ******** 694 bytes of CODE  memory
// ******** 96 bytes of DATA  memory (+ 23 absolute )
// ******** 24 bytes of CONST memory
// ******** Errors: none
// ******** Warnings: none
// ###########################################################################
// ###########################################################################


#include <msp430x11x2.h>

/* P1.0 - P1.3 input ports */
#define TIP_PA          (0x01)    // P1.0
#define TIP_PO          (0x02)    // P1.1
#define VDOR_PO         (0x04)    // P1.2
#define VDOR_PR         (0x08)    // P1.3

/* P2.0 - P2.2 input ports */
#define NMI_CAL         (0x01)    // P2.0
#define TXDO            (0x02)    // P2.1
#define DET_MAG         (0x04)    // P2.2

/* P2.3 - P2.5 output ports */
#define BAT_ST          (0x08)    // P2.3
#define RELE_ST         (0x10)    // P2.4
#define SWITCH          (0x20)    // P2.5

void delay_500ms(void);           // 500ms loop delay function
unsigned int repeatdelay;         // delay_timer*100ms
unsigned int flag;                // interrupt flag
unsigned long int batterytimer;   // SWITCH activity -> 1 sec
unsigned long int batterytemp;    // LPM3 timer
unsigned int batteryflag;         // Low batter indicator
unsigned int magnetflag;          // High magnetic field detected
unsigned int magnettimer;         // Delay after magnetflag detected

void main(void)
{
  WDTCTL = WDT_ADLY_1000;                          // WDT as timer - 1sec
  IE1 |= WDTIE;                                    // Enable WDT interrupt
  P1OUT |= 0x00;                                   // P1.x outputs set LOW
  P1DIR=0xff & ~(TIP_PA+TIP_PO+VDOR_PO+VDOR_PR);   // P1 inputs
  P2OUT = SWITCH;                                  // P2.5 high (števec OFF)
                                                   // other are low output
  P2DIR = 0x3f & ~(NMI_CAL+TXDO+DET_MAG);          // P2 inputs

  batterytimer=0;                                  // INITIALIZE timer/flag
  batterytemp=0;
  batteryflag=0;
  magnetflag=0;
  magnettimer=0;

//###########################################################################
//     MAIN PROGRAM LOOP
//###########################################################################

 while (1)
 {
 P1IE |= (TIP_PA+TIP_PO+VDOR_PO+VDOR_PR);   // P1.0-P1.3  interrupts
 P1IES |= (TIP_PA+TIP_PO);                  // P1.0 & P1.1 HIGH->LOW triger,
 CCTL2 = CCIE;                              // timer interrupt enabled
 CCR2 = 16384;                              // 500ms delay (~32kHz)
 P2IE |= TXDO+DET_MAG;                      // P2.1 & P2.2 interrupt enable
 P2IES &= ~(TXDO+DET_MAG);                  // P2.1 & P2.2 LOW->HIGH edge
 flag = 0;                                  // Initialize Port1 flag
 _EINT();                                   // Enable interrupts
 _BIS_SR(LPM3_bits);                        // Go to LPM3 sleep

 while (flag==0);                           // Wait here in LPM3
```

```
   if(flag==1)
     {
      if (!(P2IN & NMI_CAL))   // external power OFF
      {
      // ŠTEVEC ON (P2.5 => 0) always
      // Battery and Magnet flag possibility
      if (batteryflag==0 & magnetflag==0) P2OUT &= ~SWITCH;       // števec ON
      if (batteryflag==0 & magnetflag==1) P2OUT = RELE_ST;        // števec ON
      if (batteryflag==1 & magnetflag==0) P2OUT = BAT_ST;         // števec ON
      if (batteryflag==1 & magnetflag==1) P2OUT = RELE_ST+BAT_ST; // števec ON
      }
      // ŠTEVEC OFF (P2.5 => 1) always
      if (P2IN & NMI_CAL)     // external power ON
      {
      if (batteryflag==0 & magnetflag==0) P2OUT |= SWITCH;                // OFF
      if (batteryflag==0 & magnetflag==1) P2OUT |= SWITCH+RELE_ST;        // OFF
      if (batteryflag==1 & magnetflag==0) P2OUT |= SWITCH+BAT_ST;         // OFF
      if (batteryflag==1 & magnetflag==1) P2OUT |= SWITCH+RELE_ST+BAT_ST; // OFF
      }

     loop:
      delay_500ms();          // loop delay 500ms

      // Formula: TIMER_DELAY_SET = (repeatdelay-1)*500ms + 1sec = ~4sec
      repeatdelay=7;

      if (P2IN & NMI_CAL)     // If NMI_CAL=1, turn OFF števec
       {
         P2OUT = SWITCH;      // P2.5 => 1 (šetevec OFF)
         break;
       }
      else if(!(P1IN & TIP_PA)) goto loop;   // P1.0 interrupted
      else if(!(P1IN & TIP_PO)) goto loop;   // P1.1 interrupted
      else TACTL = TASSEL_1 + MC_2 + TAIE;   // ACLK, Contmode, interrupt enable
   }
 }
}


//#############################################################################
//     LOOP DELAY FUNCTION 500ms
//#############################################################################
void delay_500ms(void)
{
 volatile unsigned int i;
 for (i=37760;i>0;i--);        // 500ms "loop cycle" delay
}


//#############################################################################
//     TIMER_A1 INTERRUPT
//#############################################################################

#pragma vector=TIMERA1_VECTOR
__interrupt void Timer_A1(void)
{
  switch( TAIV )
  {
    case  2:  break;                          // CCR1 not used
    case  4:  if (repeatdelay > 1)            // 1st count is always -> 500ms
              {
                CCR2 += 16384;                // add 500ms offset to CCR2
                repeatdelay--;
               }
              else
              {
                TACTL = MC_0 + TACLR;         // stop timer, clear timer
                P2OUT |= SWITCH;              // P2.5 => 1 (šetevec OFF)
              }
              break;
    case 10:  break;                          // overflow not used
  }
}
```

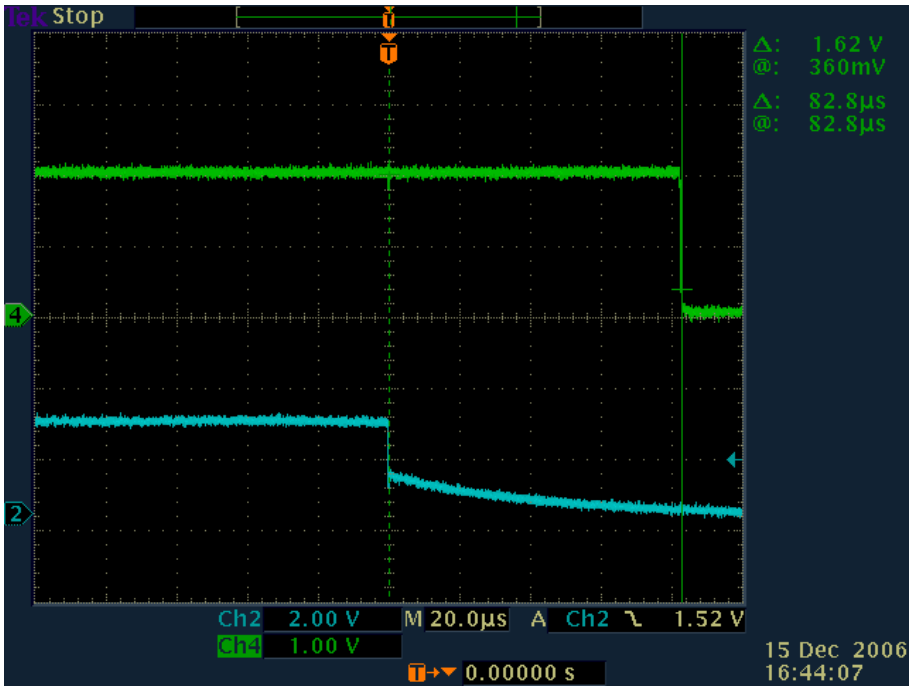
```
//#############################################################################
//     PORT_1 INTERRUPT FUNCTION SUBRUTINE
//#############################################################################

#pragma vector = PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
  _DINT();
  if((P1IFG & TIP_PA)||(P1IFG & TIP_PO)||(P1IFG & VDOR_PO)||(P1IFG & VDOR_PR))
 {
  flag = 1;                                    // P1.0-P1.3 caused interrupt
  P1IFG &= ~(TIP_PA+TIP_PO+VDOR_PO+VDOR_PR);   // P1.0-P1.3 IFG cleared
  TACTL = MC_0 + TACLR;                        // stop timer, clear timer
  _BIC_SR_IRQ(LPM3_bits);                      // Wake up system from LPM3 sleep
 }
}


//#############################################################################
//     PORT_2 INTERRUPT FUNCTION SUBRUTINE
//#############################################################################

#pragma vector = PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{
  _DINT();
  if(P2IFG & TXDO)                    // TXDO caused interrupt
  {
   P2IFG &= ~TXDO;                    // P2.1 IFG cleared
  }
  if(P2IFG & DET_MAG)                 // DET_MAG caused interrupt
  {
   P2IFG &= ~DET_MAG;                 // P2.2 IFG cleared
   magnetflag=1;
  }
 flag = 1;
 TACTL = MC_0 + TACLR;                // ustavi timer, pobriši timer
 _BIC_SR_IRQ(LPM3_bits);              // Wake up system from LPM3 sleep
}


//#############################################################################
//     WATCHDOG TIMER INTERRUPT FUNCTION SUBRUTINE
//#############################################################################

#pragma vector=WDT_VECTOR
__interrupt void watchdog_timer(void)
{
    if (P2OUT & RELE_ST)
  {
   if (magnettimer<3) magnettimer++;        // magnettimer = 3 sec
   else
   {
    magnettimer=0;
    magnetflag=0;
   }
  }

  if (!(P2OUT & SWITCH)) batterytimer++;    // ŠTEVEC ON, CPU ON (~40mA/sec)
  else
  {
   if (batterytemp<40000) batterytemp++;    // ŠTEVEC OFF, LPM3 (~1uA/sec)
   else if (batterytimer>342000) batteryflag=1;
   else
   {
     batterytemp=0;
     batterytimer++;
   }
  }
}
```

ZAKLJUČEK

Naloga je na prvi pogled dokaj preprosta, a glede na vse zahteve in standrade, ki jih je potrebno upoštevati, postane stvar precej zahtevna. Z nasveti in pomočjo sodelavcev sem za delovanje prototipa potreboval en mesec, kar gre predvsem na račun predhodnega nepoznavanja mikrokontrolerja in uvajanja v delo v podjetju. Za odpravljanje drugih manjših težav, optimiziranja programske kode in dodatnih sprememb na modulu, pa pa mi je vzelo še dober mesec dni dela.

Proti koncu projekta se je vseeno izkazalo, da bo potrebno s prenovo osnovne plošče števca. Z pridobljenimi znanji, izkušnjami in novimi idejami, bo tudi baterijski modul precej poenostavljen in cenejši. Izveden bo preprosto z logičnim vezjem, softwareska naloga pa bo v samem števcu. Vse to pa gre seveda na račun časa razvoja.

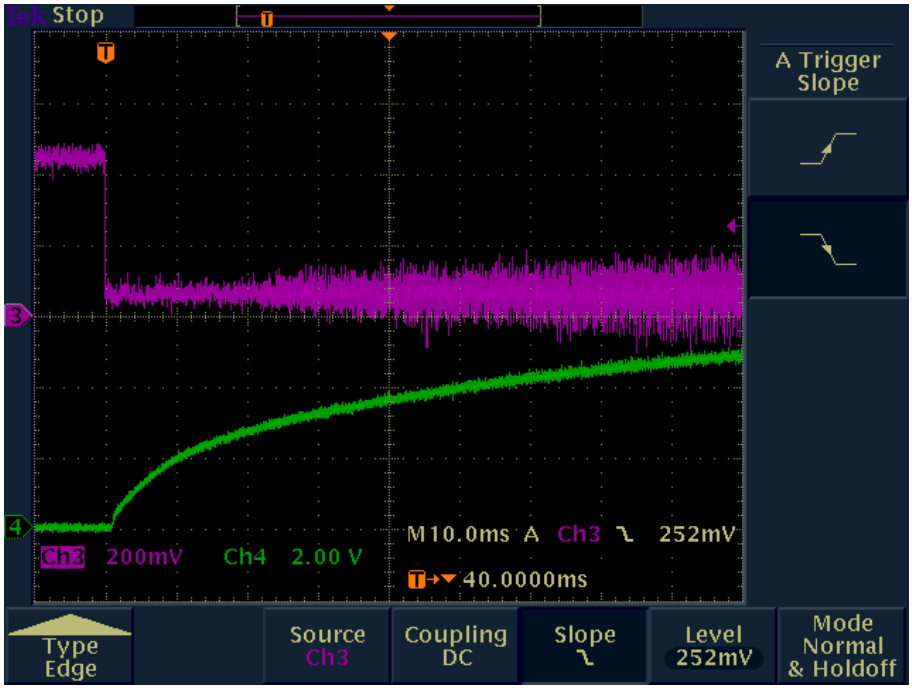
PRILOGA: karakteristike vezja, načrt



SLIKA 1 - ODZIV MIKROKONTROLERJA

CH4 – signal SWITCH

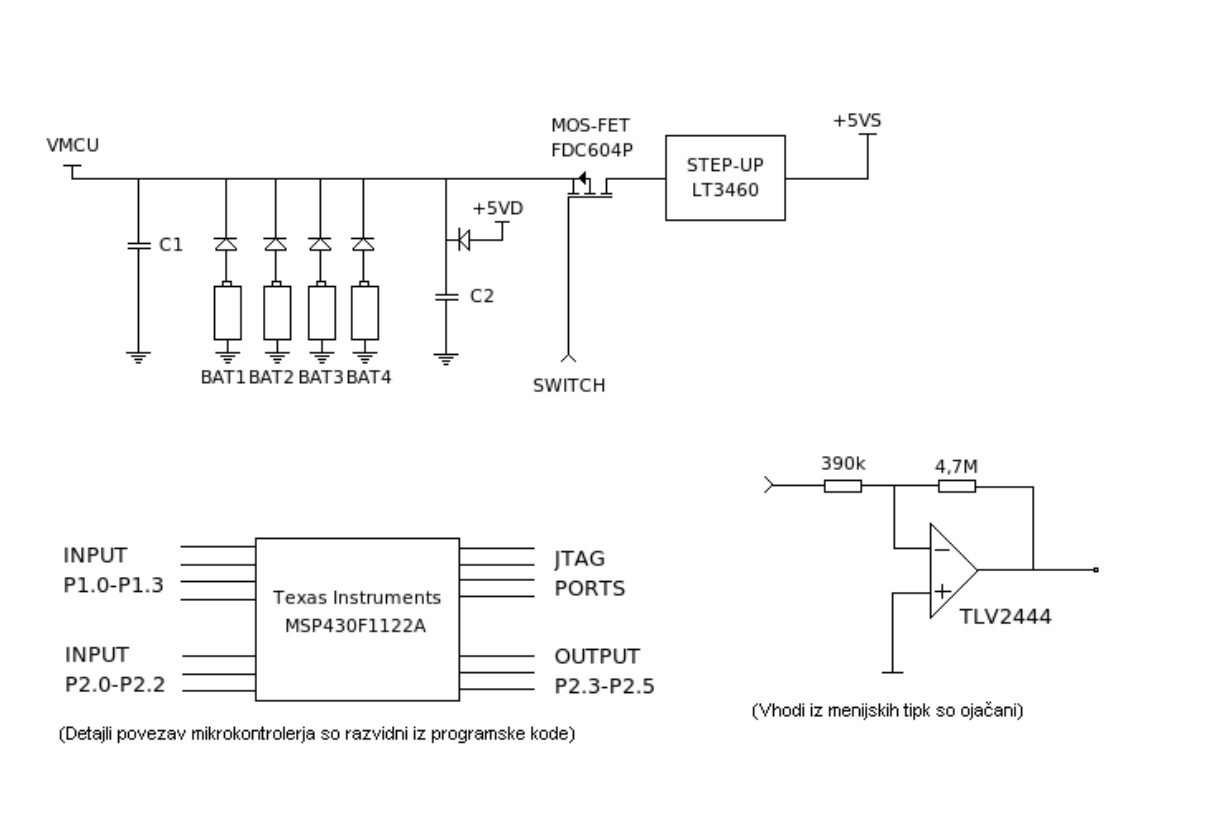
CH2 – signal TIP_PA



SLIKA 2 - POTEK IZHODNE NAPETOSTI +5VS

CH3 – Tipka PA (brez ojačanja)

CH4 – Izhod step-up pretvornika +5VS



SLIKA 3 - PREPROSTA BLOK SHEMA VEZJA