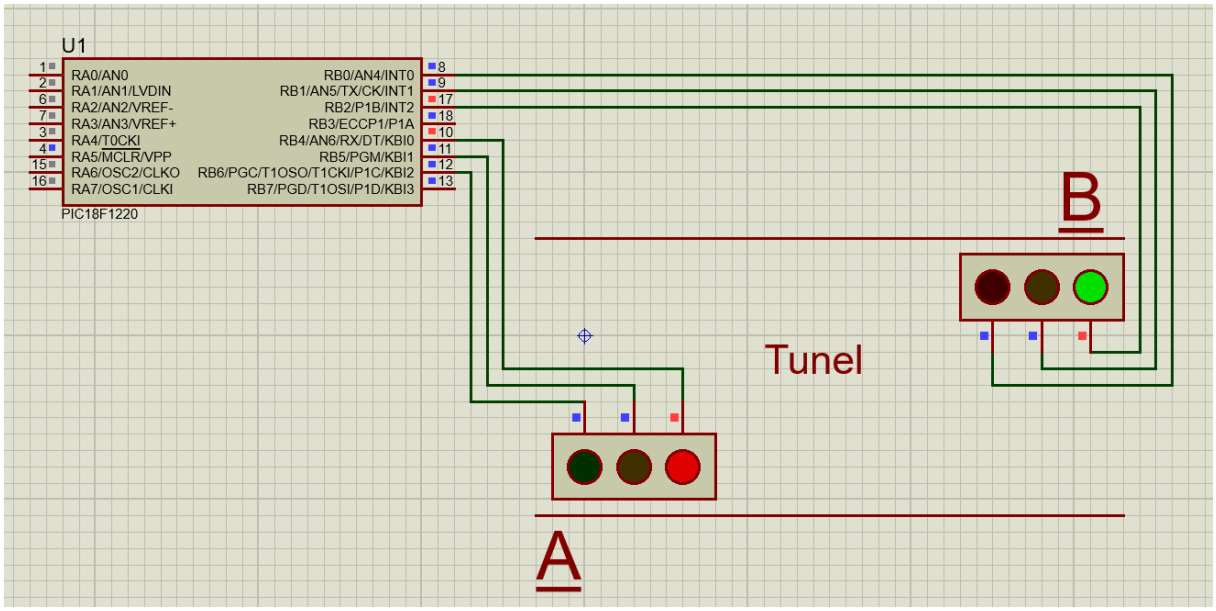
Računala i procesi

2. domaća zadaća

Ema Popović
0036506085
8. 11. 2020.

1. ZADATAK

Cilj prvog zadatka bio je sinkronizirati semafore na dva kraja tunela korištenjem programskog jezika C umjesto asemblera. Struktura napisane datoteke ostaje slična. U glavnom programu upisuju se potrebne vrijednosti u registre – sve linije su izlazne, omogućava se prekid pomoću timer0 i pokreće se timer. Semaforima se upisuje početna vrijednost. Nakon toga program vrti praznu beskonačnu petlju. Prekidni potprogram ponovno pokreće timer koji traje 2 ili 3 sekunde, ovisno o stanju. Na semaforima se pale potrebna svijetla. Računa se iduće stanje i izlazi iz potprograma. Tijekom prva dva prekida mijenja se svijetlo samo na semaforu A, što se omogućava globalnom varijablom koja se koristi kao brojač prekida. Shema je prikazana slikom 1.



Slika 1. Shema tunela u Proteusu

Programski kod

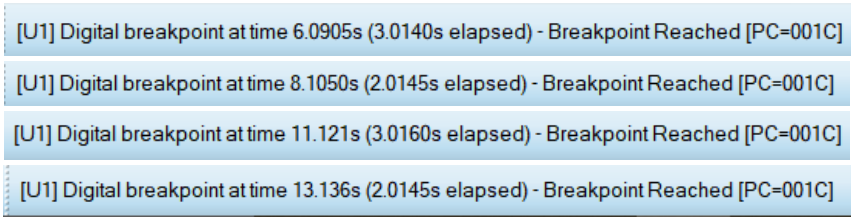
```
#include <p18f1220.h>
#pragma config WDT = OFF
#pragma config PWRT= ON
#pragma config MCLRE=OFF // zadano

unsigned int state = 1; // globalna varijabla stanje, pocetno - crveno
unsigned int stojiSemaforB = 2; // brojac semafora B - prva dva prekida se ne mice s crvenog
void high_isr(void); // prototip prekida

#pragma code high_vector=0x08 // prekidni vektor

void interrupt_at_high_vector(void){
    _asm GOTO high_isr _endasm // prekidna rutina
}
#pragma code // gotov prekidni vektor
```

```
#pragma interrupt high_isr
// prekidna rutina visokog prioriteta
void high_isr (void){
  INTCON = 0xA0; // dozvoli globalne prekide i prekid timera 0
  T0CON = 0x88; //  pokreni timer0, 16-bitni, interni oscilator, ne koristi prescaler

  if ((state == 0) || (state ==2)){    // crveno/zeleno --> 3 sekunde
    TMR0H = 0xE8; // oko 65536 – 6000 = 59536, gornji bitovi
    TMR0L = 0xAE; //  donji bitovi
  }
  else {        // prijelazna stanja --> 2 sekunde
    TMR0H = 0xF0; // oko 65536 – 4000 = 61536, gornji bitovi
    TMR0L = 0x7B; //  donji bitovi
  }

  if (stojiSemaforB) {        // B stoji u crvenom, A se mice, upali tocna svijetla
    switch (state){
        case 0:
          PORTB = 0x11;
          break;
        case 1:
          PORTB = 0x31;
          break;
        case 2:
          PORTB = 0x41;
          break;
        case 3:
          PORTB = 0x21;
          break;
    }
    --stojiSemaforB; // umanji brojac
  }
  else {        // oba semafora se micu, upali tocna svijetla
    switch (state){
        case 0:
          PORTB = 0x14;
          break;
        case 1:
          PORTB = 0x32;
          break;
        case 2:
          PORTB = 0x41;
          break;
        case 3:
          PORTB = 0x23;
          break;
    }
  }

  state = (state + 1) % 4; // promijeni stanje
}
```

```
#pragma code // gotov prekidni vektor


void main(void){
  PORTB = 0x11; // semafori pocinju u crvenom
  TRISB = 0; //sve linije izlazne

  INTCON = 0xA0; // dozvoli globalne prekide i prekid timera 0
  T0CON = 0x88; //  pokreni timer0, 16-bitni, interni oscilator, ne koristi prescaler
  TMR0H = 0xE9; // postavljamo trajanje od 3 sekundi, oko 65536 – 6000 = 65536, gornji bitovi
  TMR0L = 0x29; //  donji bitovi

  while(1){} // beskonacna petlja
}
```

Analiza

Frekvencija takta podešena je u Proteusu na 8 kHz. Frekvencija rada timera koristi unutarnji oscilator i računa se kao F_{OSC} / 4 i iznosi $\frac{8\ kHz}{4} = 2\ kHz$. Prijelazna stanja semafora traju 2 sekunde, a zeleno/crveno traje $5\ \%\ 5 + 3 = 3$ sekunde. Kako se u jednoj sekundi izvede 2000 taktova, za prijelazna stanja potrebno je 4000, a za zeleno/crveno 6000 taktova. Potrebno je 16-bitno brojilo koje se inkrementira svakim novim signalom. Konstante koje označavaju potreban broj signala su 65536 – 4000 = 61536 ($F060_{16}$) za prijelazna stanja i 59536 ($E890_{16}$) za zeleno/crveno.
Pomoću postavljanja breakpointa vidljivo je da postoji kašnjenje. Kod ciklusa od 2s dobiva se vrijeme od 2.0145s. Kod ciklusa od 3s dobivaju se naizmjenice vrijednosti 3.014s i 3.016. Slike 2 – 5 prikazuju vremenske vrijednosti za jedan prolazak kroz sva stanja. Pozivanje prekidnog potprograma, upisivanje konstanti u timer i njegovo ponovno pokretanje troše vrijeme, a nisu uračunati u programskom kodu.



Slike 2-5. Početni intervali za svako stanje

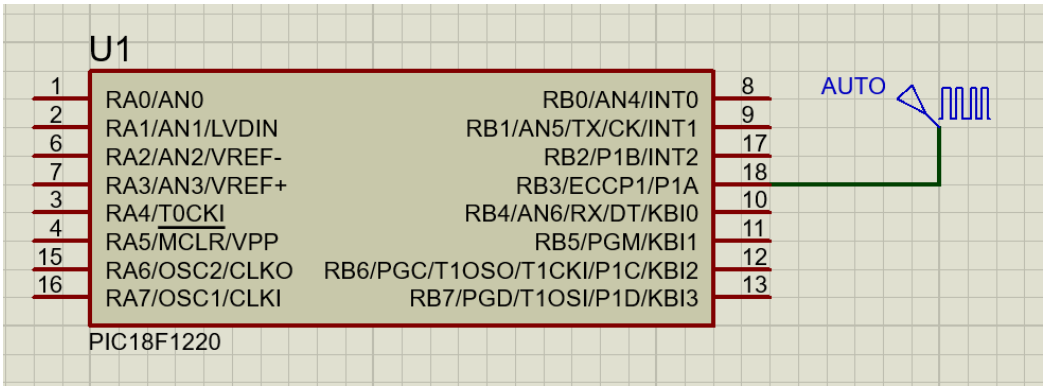
Modifikacijom konstanti (točnije povećavanjem) se može dobiti točniji interval. Ova razlika u vremenu se koristi za pozivanje prekida i upis u timer. Prvi upis konstanti događa se u main programu, nakon čega se izvodi beskonačna petlja. U tom upisu konstante 3.000 sekunde se postižu sa vrijednosti $E929_{16}$. Kako je to prvi upis u timer, konstanta je različita od budućih gdje se zbog potprograma odvijaju dodatne neuračunate radnje. U prekidnom potprogramu točne vrijednosti se postižu konstante $F07D_{16}$ (za 2s) i $E8B0_{16}$ (za 3s). Postignut je točan interval za 2s, dok duži interval izmjenjuje vrijednost između 3.000s i 2.998s. Slike 7 – 10 prikazuju vremenske vrijednosti prolaska kroz stanja nakon izmjene konstanti.

[U1] Digital breakpoint at time 5.9980s (2.9980s elapsed) - Breakpoint Reached [PC=001C]
[U1] Digital breakpoint at time 7.9980s (2.0000s elapsed) - Breakpoint Reached [PC=001C]
[U1] Digital breakpoint at time 10.998s (3.0000s elapsed) - Breakpoint Reached [PC=001C]
[U1] Digital breakpoint at time 12.998s (2.0000s elapsed) - Breakpoint Reached [PC=001C]

Slike 7-10. Konačni intervali za svako stanje

2. ZADATAK

Cilj drugog zadatka je simulirati mjerač brzine. Kretanje auta predstavlja generator impulsa. U glavnom programu pokreće se timer1 koji broji vrijeme između dva osjetila udaljena 2 cm. Aktivira se CCP jedinica u capture načinu rada i stvara prekide kada se pojavi rastući brid. Dopuštaju se prekidi od CCP-a i preljeva. Kada auto prođe prvo osjetilo prekidni potprogram počinje mjerenje brzine timerom. Ako se u timeru dogodi preljev, prekidni potprogram ga resetira i zabilježi preljev. Kada auto prođe drugo osjetilo računa se brzina i pomoćne varijable se pripreme za iduće mjerenje.



Slika 11. Shema mjerenja brzine u Proteusu

Programski kod

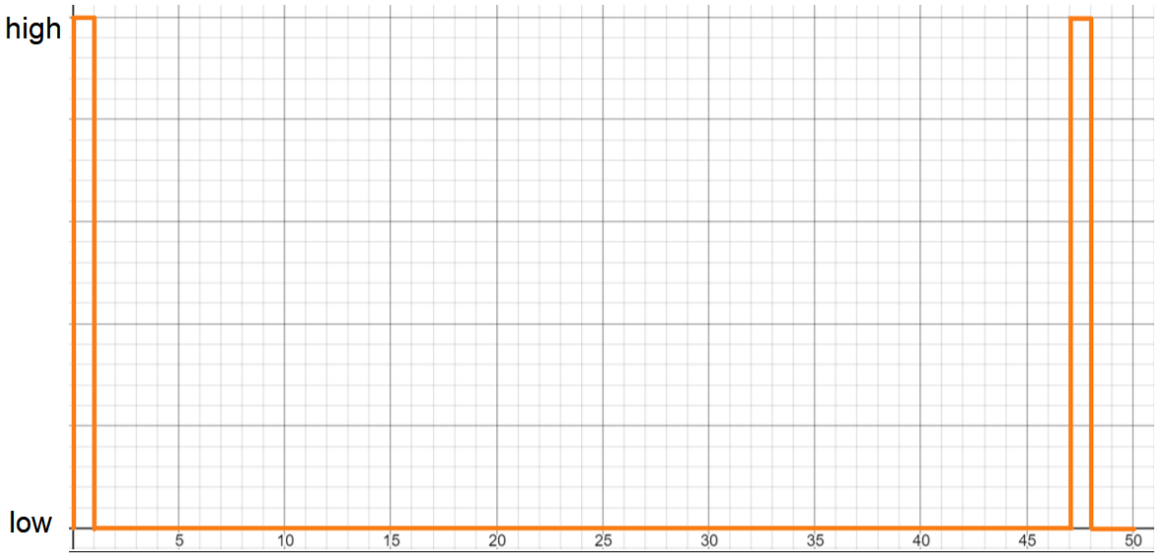
```
#include <p18f1220.h>
#pragma config WDT = OFF
#pragma config PWRT= ON
#pragma config MCLRE=OFF // zadano

unsigned int brzina = 0; // globalna varijabla brzine
unsigned int preljev = 0; // koliko puta se dogodio preljev
unsigned mjerenjeBrzine = 0; // je li u tijeku mjerenje brzine

void high_isr(void); // prototip prekida

#pragma code high_vector=0x08 // prekidni vektor

void interrupt_at_high_vector(void){
    _asm GOTO high_isr _endasm // prekidna rutina
}
#pragma code // gotov prekidni vektor

#pragma interrupt high_isr
// prekidna rutina visokog prioriteta
void high_isr (void){

  if (PIR1bits.TMR1IF) { // ako se dogodio preljev
    ++preljev; // povecaj brojac preljeva
```

```
      PIR1bits.TMR1IF = 0; // postavi indikator prekida nazad na 0

      T1CON = 0xC9; // pokreni timer1 otpocetka
      TMR1H = 0; // broji do 0 (65536 taktova)
      TMR1L = 0;
    }

    else if (PIR1bits.CCP1IF) { // ako je detektiran pokret
      PIR1bits.CCP1IF = 0; // postavi indikator prekida nazad na 0

      if ( !mjerenjeBrzine ) { // ako je auto sada prosao kroz prvo osjetilo
          mjerenjeBrzine = 1; // pocni mjeriti vrijeme

          T1CON = 0xC9; // pokreni timer1, 16-bitni, interni oscilator, ne koristi prescaler
          TMR1H = 0; // broji do 0 (65536 taktova)
          TMR1L = 0;
      }

      else { // ako je auto prosao kroz oba osjetila
          mjerenjeBrzine = 0; // vise se ne mjeri brzina
          brzina = 0.02 / ( (float) ( 65536 * preljev + 256 * CCPR1H + CCPR1L  ) / 10000000
); // izracunaj brzinu -> 2cm / (taktovi / (Fosc / 4 ) )
          preljev = 0; // ocisti brojac preljeva
      }
    }

}

#pragma code // gotov prekidni vektor


void main(void){
  PORTB = 0; // ocisti PORTB
  TRISB = 0x08; // cetvrti bit je ulaz mjerila

  CCP1CON = 0x05; //  capture mode, na rastuci brid
  T3CON = 0; // timer1 je izvor takta za ccp

  PIE1 = 0x05; // dozvoli CCP prekid i preljev prekid
  IPR1 = 0x05; // CCP i preljev prekid imaju visoki prioritet
  INTCON = 0xC0; // dozvoli globalne prekide + prekid zbog preljeva

  while(1){} // beskonacna petlja
}
```

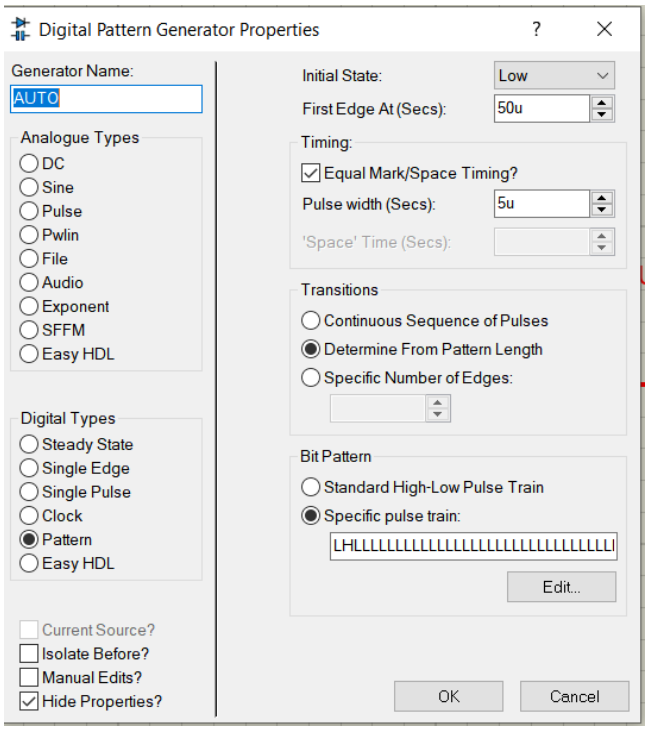
Analiza

Brzina pri kojoj se testirao rad sklopa iznosi 85 m/s. Vrijeme između prolaska kroz dva osjetila je $\frac{0.02\ m}{85\ m/s} = 235.3\ \mu s$ (zaokruženo na 235 µs). Ovo vrijeme rastavljeno je na 47 taktova od 5 µs, generator impulsa šalje visoki signal, 46 niskih, pa opet visoki. Grafički prikaz signala je na slici 12. Brzina se uz ove postavke netočno izračuna kao 91 m/s.

Postavke generatora nalaze se na slici 13. Broj impulsa treba se povećati kako bi se ukalkuliralo i vrijeme potrošeno u prekidnom potprogramu prilikom preljeva.



Slika 12. Izlazni signal generatora



Slika 13. Postavke generatora impulsa

Izračunata brzina se može mijenjati brojem niskih impulsa između visokih. Točan rezultat se dobije ako se generira 49 niskih impulsa u jednom mjerenju. Točnost ovisi i o širini impulsa. Kraćim impulsima točnije se aproksimiraju brzine. Korištenjem impulsa širine 47 µs dobivaju se ili 91 m/s (za 4 niska impulsa između visokih) ili 75 (za 5 niskih impulsa).

Za brzinu 1 m/s koristi se 40 impulsa širokih 500 µs (visoki – 39 niskih – visoki) i dobiva se točan iznos. Za manju brzinu trebao bi se povećati broj niskih signala. S 40 niskih signala brzina pada na nulu kako je razlučivost mjerenja 1 m/s i račun se obavlja s cijelim brojevima. Za brzinu 100 m/s koristi se 40 impulsa širokih 5 µ/s (visoki – 39 niskih – visoki) i dobiva se dobiva se netočnih 109 m/s. Dodavanje 3 niska impulsa rezultira sa 100 m/s. Točnost mjerenja je veća za šire impulse. Uži impulsi (i više brzine) rezultiraju većim brojem preljeva, koji svaki put rezultira prekidom. U prekidu se povećava brojač preljeva, postavlja bit registra PIR1 i pokreće timer1, što troši vrijeme i smanjuje točnost.