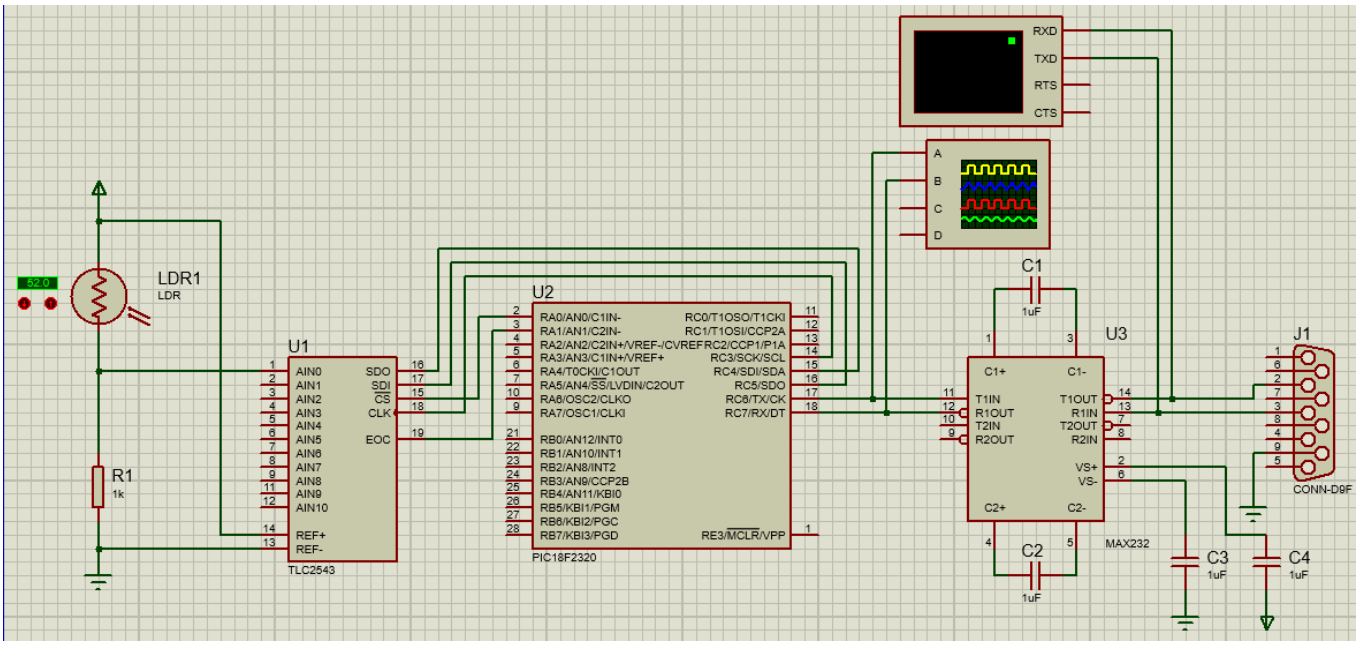
# Računala i procesi

## 4. domaća zadaća

Ema Popović
0036506085
4. 1. 2020.

U zadatku je potrebno ostvariti modul za praćenje razine osvjetljenja u tunelu. U glavnom programu definiraju se ulazi i izlazi, omogućuju se prekidi, inicijalizira se SSP jedinica (vanjski A/D pretvornik) i konfigurira sučelje RS-232. U beskonačnoj petlji se dopušta komunikacija s pretvornikom. Slijedi čitanje izmjerenog podatka uz pomoć buffer varijable, prvo viši, pa niži bajt. Izvodi se u modificiranoj funkciji u kojoj se podaci mogu istovremeno čitati i pisati, `WriteSPI_mine`. Nakon čitanja podatka komunikacija se zabranjuje i A/D pretvornik prelazi u stanje spavanja. Aktivira se i spavanje mikrokontrolera pomoću watchdog timera i traje 16384ms. Upisom znaka ? u virtualni terminal može se zatražiti razina osvjetljenja. Program u prekidnoj proceduri pronalazi potrebnu vrijednost u lookup tablici i ispisuje ju u terminal.




Slika 1. Proteus - shema projekta

# Programski kod:

```
#include <p18f2320.h>
#include <timers.h>
#include <delays.h>
#include <reset.h>
#include <spi.h>
#include <usart.h>
#include <stdlib.h>

#pragma config WDT = ON
#pragma config WDTPS = 4096

#pragma config PWRT= ON
#pragma config MCLRE=OFF

#define SPI_CS LATAbits.LATA0

float lookupTable[13] = {0.0012, 0.0033, 0.0091, 0.0254, 0.0711, 0.1993, 0.5623, 1.6053, 4.6983, 14.5071, 50.8071, 258.0626,1000};
unsigned int procitano;
char ispisano[];
unsigned int index;
int svijetlo;
unsigned char terminal;

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){
   terminal = ReadUSART();
   if ( terminal== '?' ) { // ako je dosao upit
      index = 15;
      while ( (index >=0) && ((procitano >> index) != 1) ) {       //trazi poziciju prve jedinice, kad je pronadena, to je indeks za vrijednost iz lookup tablice
         index--;
      }
      svijetlo = (int)lookupTable[index] ;
      itoa( svijetlo, ispisano); // pretvori lookup vrijednost u char
      putsUSART(ispisano);  // upisi rezultat
   }
   INTCON |= 0xC0;    // omogucen prekid od vanjske jedinice
}
#pragma code // gotov prekidni vektor

unsigned char WriteSPI_mine( unsigned char data_out, unsigned int * procitano ) {
   unsigned char TempVar;
   TempVar = SSPBUF; // Clears BF
   PIR1bits.SSPIF = 0; // Clear interrupt flag

   SSPBUF = data_out; // write byte to SSPBUF register
   if ( SSPCON1 & 0x80 ) // test if write collision occurred
      return ( -1 ); // if WCOL bit is set return negative #
   else
      while( !PIR1bits.SSPIF ); // wait until bus cycle complete

   *procitano = SSPBUF * 256;    // upisi najvisih 8b
   PIR1bits.SSPIF = 0;       // omoguci novi prijenos podataka
   SSPBUF = 0x00;    // novi ciklus
   while( !PIR1bits.SSPIF ); // wait until bus cycle complete
   *procitano += SSPBUF;  // ucitaj i donjih 8b
   return ( 0 );
}

void main(void) {
   TRISA &= 0xFE;    // CS bit izlazni
   INTCON |= 0xC0;    // omogucen prekid od vanjske jedinice
   OpenSPI(SPI_FOSC_16, MODE_00, SMPEND);       // inicijaliziraj SSP jedinicu
   OpenUSART( USART_TX_INT_OFF &
              USART_RX_INT_ON &
              USART_ASYNCH_MODE &
              USART_NINE_BIT &
              USART_CONT_RX &
              USART_BRGH_HIGH,
              15 ); // konfiguriraj USART - off transmit interrupt, on receive interrupt, async, 9b (8b i stop),
                    // kontinuirani receive, brzi nacin, spbrg = 4915200 / (16 * 19200) - 1 = 15
   while(1) {
      SPI_CS = 0;       // aktiviraj ~CS
      Delay1TCY();Delay1TCY();Delay1TCY();Delay1TCY();Delay1TCY();Delay1TCY();Delay1TCY();Delay1TCY();       // cekaj da se sklopovlje pripremi 0.2us * 8 = 1.6us
      WriteSPI_mine(0x0C, &procitano);        // pocni s pisanjem/citanjem, AIN0, 12b, MSB
      SPI_CS = 1;       // deaktiviraj ~CS, gotov prijenos

      WriteSPI(0xE0);  // A/D software power down
      Sleep();       // spavaj 16384ms
   }
}
```

## Rezultati


Slike 2. i 3. Zahtjev i odziv, vrijednost 2.0


Slike 4. i 5. Zahtjev i odziv, vrijednost 5.0

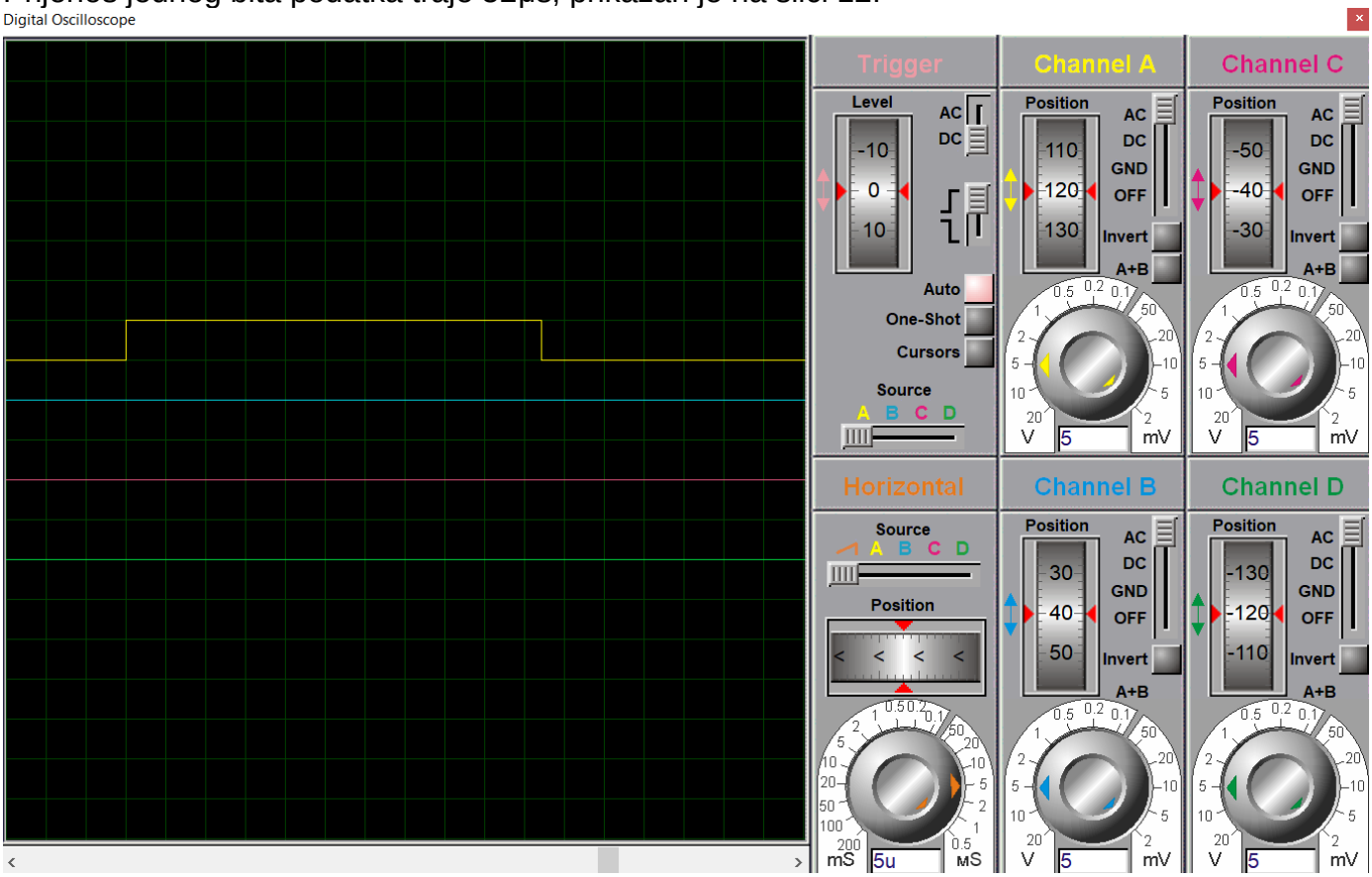

Slike 6. i 7. Zahtjev i odziv, vrijednost 25.0


Slike 8. i 9. Zahtjev i odziv, vrijednost 100.0


Slike 10. i 11. Zahtjev i odziv, vrijednost 400.0

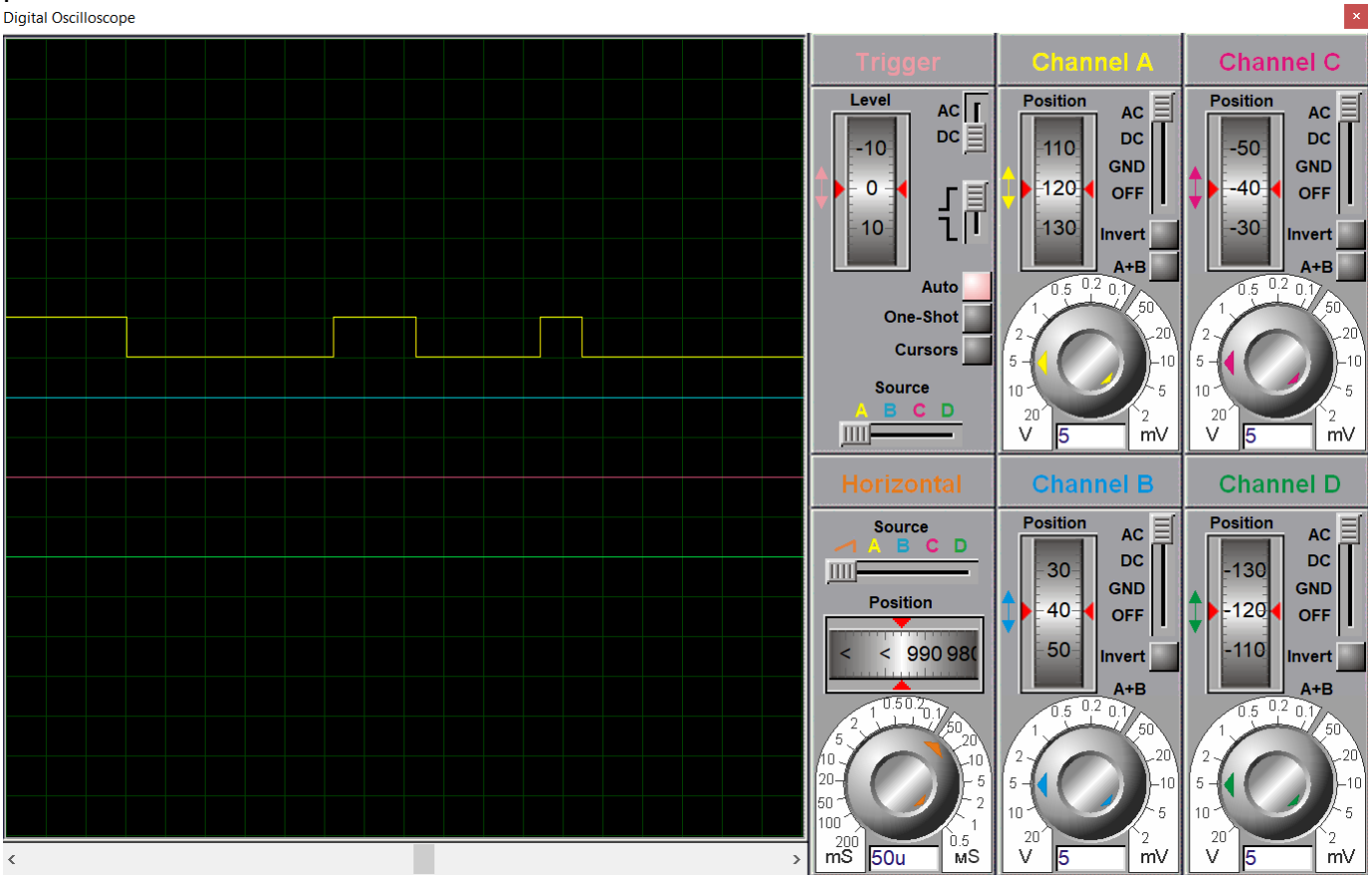
# OSCILOSKOP

Prijenos jednog bita podatka traje 52μs, prikazan je na slici 12.



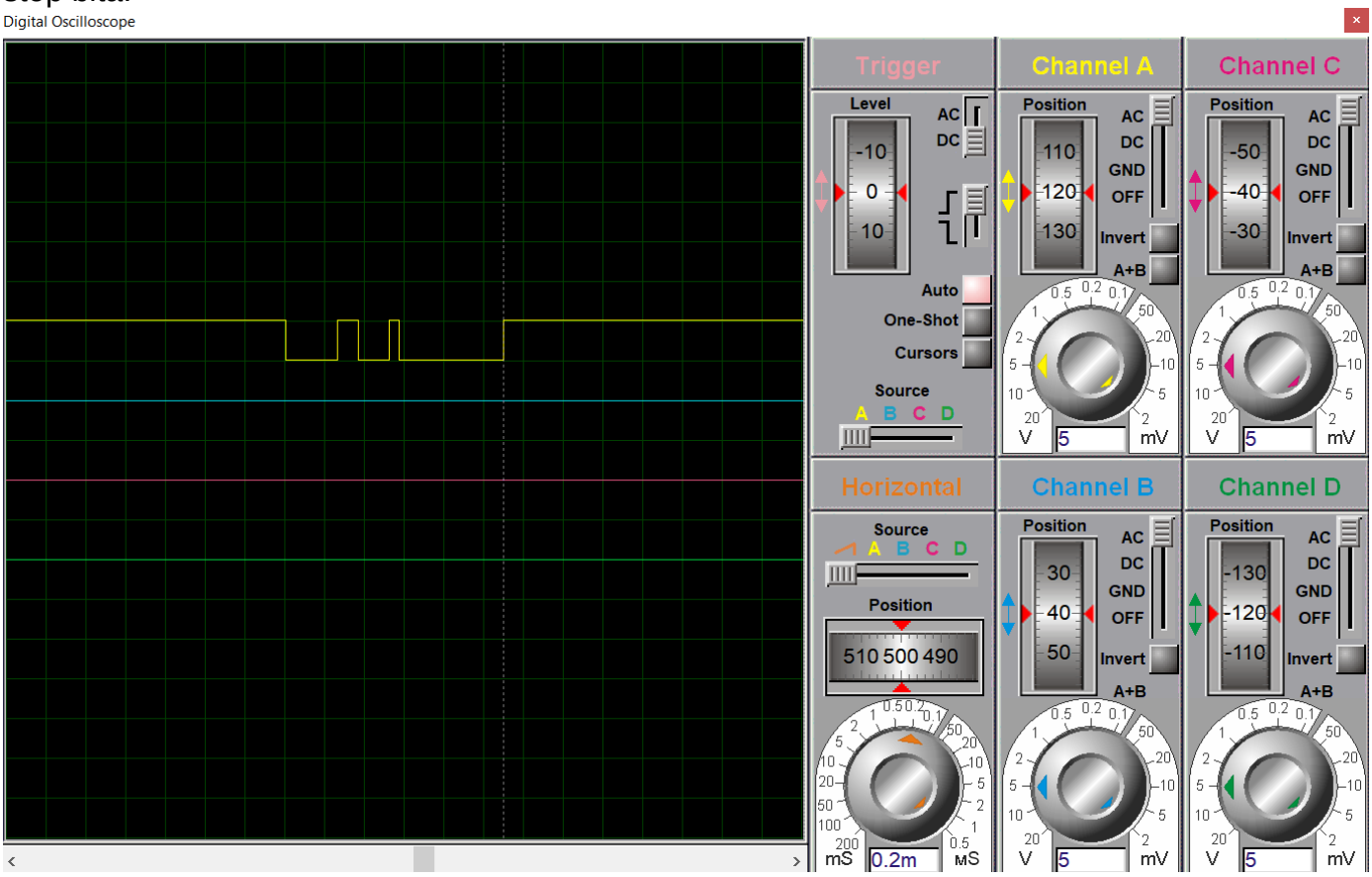

Slika 12. Osciloskop – prikaz prijenosa jednog bita

Trajanje prijenosa jednog okteta poruke traje 520μs i prikazan je na slici 13. Razlog duljeg prijenosa je što se prije okteta šalje start bit, a nakon okteta stop bit – za oktet je zapravo potrebno 10 bitova.




Slika 13. Osciloskop – prikaz prijenosa okteta

Za slanje cijele poruke potrebna je 1.1ms i prijenos je prikazan slikom 14. Sadržaj cijele poruke iznosi više od 8 bitova, pa su potrebna dva okteta za sadržaj. Uz njih se šalju ukupno dva start i stop bita.




Slika 14. Osciloskop – prikaz prijenosa poruke

## TOČNOST

Izračuni dobiveni pomoću lookup tablice veoma su neprecizni. Pri dobivanju rezultata ne koriste se operacije potenciranja, one bi usporile rad i važnije, program bi postao veći od maksimalnih 2kB u Proteusu. Osim uvođenja računanja vrijednosti za točniju preciznost mogla bi se koristiti i lookup tablica sa više vrijednosti. U tom slučaju u obzir bi se trebalo uzeti više od samo prvog bita vrijednosti dobivene iz pretvornika kako bi se dobilo više manjih koraka. I u ovom slučaju bi program bio prevelik za Proteus s maksimalnih 2kB izvršnog koda.