main.c:

```
/////******************** Projekt X  *****************//////////////
/*
Formål: Main er hvor funktionerne bliver kaldt til at udfører programmet.
04-08-2020
Udarbejdet af:
Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
*/


#include <avr/io.h>
#include <math.h>
#include <string.h>
#include "Uart1.h"
#include "TIMER.h"
#include "ADC.h"
#include "SPI.h"
#include <util/delay.h>
#include <avr/pgmspace.h>
#include <avr/interrupt.h>
#define F_CPU 16000000UL
#define USART_BAUDRATE 115200
#define MYBRRD F_CPU/8/USART_BAUDRATE-1
```

```
/* Variabel initialising */
volatile int record_length = 500;
volatile int comparevalue = 249;
volatile char indicator = 0;
volatile char shape = 0;
volatile char amplitude = 0;
volatile char frequency = 0;
volatile char Hexarray[11];
volatile char LENGTH = 15;
char buffer0 = 1;
char buffer1 = 0;
char rx_comp = 0;
int lab_flag = 1;
int samplrate = 500 ;
int i = 0;
int p = 0;
int l = 0;
int k = 0;
volatile char adc_s[1000];
volatile char adc_s1[1000];
volatile char adc_done = 0;
volatile char adc_done1 = 0;
```

```
ISR(TIMER1_COMPB_vect){}
//interrupt service rutine for ADC
ISR (ADC_vect){
    if (buffer0 == 1){                          //Læser ind på den første buffer når Buffer0 flag er højt
        adc_s[l] = ADCH;
        l++;
    }
    if (buffer1 == 1){                          // Læser ind på den anden buffer når Buffer1 flag er sat højt
        adc_s1[p] = ADCH;
        p++;
    }
    if (l == record_length){                    // Når den første buffer er fuld, så bliver Buffer0 flaget sat lavt og
        l = 0;                                  // buffer1 flaget bliver sat højt så de hele siden skifter.
        adc_done = 1;
        buffer0 = 0;
        buffer1 = 1;
    }
    if (p == record_length){
        p = 0;
        adc_done1 = 1;
        buffer0 = 1;
        buffer1 = 0;
    }
}
```

```
//interrupt service rutine for uart
ISR(USART1_RX_vect){                              //pakken fra UDR1 bliver fyldt på et array
  Hexarray[i] = UDR1;
  i++;                                 // Tæller i op til brug i main funktion
}
void Hex_generator(void){
    if (k == 0 && Hexarray[5] == 0x00 ){
      shape = Hexarray[6];
    }
     if (k == 1 && Hexarray[5] == 0x00 ){
      amplitude = Hexarray[6];
    }
    if (k == 2 && Hexarray[5] == 0x00 ){
      frequency = Hexarray[6];
    }
    putchUSART1(0x55);
    putchUSART1(0xAA);
    putchUSART1(0x00);
    putchUSART1(0x0B);
    putchUSART1(0x01);
    putchUSART1(indicator);
    putchUSART1(shape);
    putchUSART1(amplitude);
    putchUSART1(frequency);
    putchUSART1(0x00);
    putchUSART1(0x00);
```

```
}
void check_type(void){
    char type = Hexarray[4];
    switch(type){
         // if type == 0x01
        case 0x01:
       //BTN0
        if (Hexarray[5] == 0x00){
            Hex_generator();
            //fpga send new info
            putcSPI_master(0x55);
            putcSPI_master(shape);
            putcSPI_master(frequency);
            putcSPI_master(amplitude);
            putcSPI_master(0xFF);
            putcSPI_master(0x00);
        }
        //BTN1
        if (Hexarray[5] == 0x01){
            k++;
            indicator = k;
            if (k == 3){
                k = 0;
            }
            Hex_generator();
        }
```

```
//BTN3, code and labview reset
    if (Hexarray[5] == 0x03){
      k = 0;
      indicator = 0;
      shape = 0;
      amplitude = 0;
      frequency = 0;
      putchUSART1(0x55);
      putchUSART1(0xAA);
      putchUSART1(0x00);
      putchUSART1(0x0B);
      putchUSART1(0x01);
      putchUSART1(0x00);
      putchUSART1(0x00);
      putchUSART1(0x00);
      putchUSART1(0x00);
      putchUSART1(0x00);
      putchUSART1(0x00);
      //fpga reset
      putcSPI_master(0x55);
      putcSPI_master(0x00);
      putcSPI_master(0x00);
      putcSPI_master(0x00);
      putcSPI_master(0x00);
      putcSPI_master(0x00);
    }
```

```
        break
        //Gemmer samplerate, udregner comparevalue og gemmer Record length
        case 0x02:
            samplrate = (Hexarray[5]<<8)|(Hexarray[6] & 0xFF);
            record_length = (Hexarray[7]<<8)|(Hexarray[8] & 0xFF);
            comparevalue = (250000UL/(samplrate))-1;
            Timer1(comparevalue);
            break;
    }
}
int main (void){
    //call of functions
    sei();            //enable globalt interrupt
    enableReceice_Itr1();
    uart1_Init(MYBRRD);
    ADCinit(1);
    Timer1(249);
    SPI_master_init ();

    while(1){
        if (rx_comp == 0){
            if(Hexarray[0]!= 0x55){                //check synch byte
                    i = 0;
            }
            if (Hexarray[0] == 0x55 && i > 2){        //save length of hexadecimal package
            LENGTH = Hexarray[3];
```

```
                    }

        if (i > LENGTH-2){                        //Done reading and storing information
        i = 0;
        rx_comp = 1;
        }
    }

    if (rx_comp == 1){
        _delay_ms(10);
        check_type();
    }

    // Data pakke sendes til LabView
    else{
        if (adc_done == 1 && rx_comp == 0){
            putchUSART1(0x55);
            putchUSART1(0xAA);
            putchUSART1((record_length+7)>>8);
            putchUSART1((char)(record_length+7));
            putchUSART1(0x02);
            for (int p = 0; p < record_length; p++){
                putchUSART1(adc_s[p]);
            }
            putchUSART1(0x00);
            putchUSART1(0x00);
            adc_done = 0;
        }

        if (adc_done1 == 1 && rx_comp == 0){
            putchUSART1(0x55);
            putchUSART1(0xAA);
            putchUSART1(((record_length+7)>>8));
            putchUSART1((char)(record_length+7));
            putchUSART1(0x02);
            for (int p = 0; p < record_length; p++){
            putchUSART1(adc_s1[p]);
            }
            putchUSART1(0x00);
            putchUSART1(0x00);
            adc_done1 = 0;
        }
    }
    rx_comp = 0;
  }
}
```

ADC.c:

```
//Formål: Dette modul initialisere ADC'en.
//Created: 04-08-2020
//Udarbejdet af:
//Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
```

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include "ADC.h"
#include "TIMER.h"
#include "Uart1.h"

void ADCinit(int channel){
    ADCSRA|=(1<<ADEN);      // Enabler ADC'en
    ADCSRA|=(1<<ADATE)|(1<<ADIE);       //enabler Auto trigger mode og ADC Interrupt Enable
    ADCSRB|=(1<<ADTS2)|(1<<ADTS0); // timer compare match B
    ADMUX = channel;
    ADMUX|=(1<<ADLAR);
    DIDR0 = (1<<channel);
    DIDR0 = 0b11111101;
    DIDR1 = 0xff;
}
```

ADC.h:

```
// Created: 04-08-2020
// Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
#ifndef ADC_H_
#define ADC_H_
```

```
extern char call_adc_done();
extern void ADCinit(int channel);
#endif
```

SPI.c:

```
  // Created: 04-08-2020
  // Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
#include <avr/io.h>
void SPI_master_init (){
    DDRB|=(1<<DDB2)|(1<<DDB1)|(1<<DDB0); //opens needed ports
    SPCR|=(1<<SPE)|(1<<MSTR)|(1<<CPOL)|(1<<CPHA); // sets clock rate start, sample on rising, setup on falling.
    SPCR|=(1<<SPR1)|(1<<SPI2X); // 500k baud
    PORTB|=(1<<PB0); //pin b0 = 1
}
void putcSPI_master(unsigned char DATA){
    PORTB &=~(1<<PB0);
    SPDR=DATA;// transmits data
    while(!(SPSR&(1<<SPIF))); // waits for data to complete
    PORTB|=(1<<PB0);
}
```

SPI.h:

```
  // Created: 04-08-2020
  // Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811 Author: victo
#ifndef SPI_H_
#define SPI_H_
extern void SPI_master_init ();
extern void SPI_slave_init();
extern void putcSPI_master(unsigned char DATA);
unsigned char getcSPI_master(void);
#endif /* INCFILE1_H_ */
```

TIMER.c:

```
  // Created: 04-08-2020
  // Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
#include <avr/io.h>
#include "ADC.h"
#include "TIMER.h"
#include "Uart1.h"
int Comparevalue;
void Timer1 (unsigned int Comparevalue){
   TCCR1B |=(1<<CS11)|(1<<CS10);      // Sætter prescaler factor til 64 så vi får den ønskede compare match value
   TCCR1B |=(1<<WGM02);               // Indstiller CTC mode
   OCR1B = Comparevalue;              // Compare match value sat til at ændre sig i forhold til samplrate modtaget fra LabView
   OCR1A = Comparevalue;
   TIMSK1|=(1<<OCIE1B);               // enabler timer interrupt
}
```

TIMER.h:

```
     // Created: 04-08-2020
    // Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
#ifndef TIMER_H_
#define TIMER_H_
volatile char state;
```

```
//int samplrate;
extern void Timer1 (unsigned int Comparevalue);
extern void TimskEnable ();
extern void TimskClear ();
extern int Compvalcalc(char Hexarray[]);
#endif
```

UART1.c:

```
            // created: 04-08-2020
            // Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811
#include <avr/io.h>
#include <avr/interrupt.h>
#include "uart1.h"
#define USART_BAUDRATE 115200
#define baud F_CPU/8/USART_BAUDRATE-1
void uart1_Init(unsigned int ubrr){
   UCSR1B|=(1<<RXEN1)|(1<<TXEN1)|(1<<RXCIE1); // enable receive and transmit and receive complete interrupt
   UCSR1C|=(1<<UCSZ10)|(1<<UCSZ11);   // frame: 1 start bit, 8 data bit, no parity:
   UBRR1H = (unsigned char)(ubrr>>8); //baud rate values up to 16 bit therefore to registers
   UBRR1L = (unsigned char)ubrr;
   UCSR1A=(1<<U2X1);    //full duplex
}
char getchUSART1(void){                              //modtager et bit oretunerer det
   while (!(UCSR1A &(1<<RXC1)));                     // venter til karakter er modtaget
   return UDR1;
}
void putchUSART1(char tx){                           //transmiterer et byte
   while (!(UCSR1A&(1<<UDRE1)));
   UDR1 = tx;
}

void enableReceice_Itr1(){
   UCSR1B|=(1<<RXCIE1);
}
```

UART1.h:

```
        // created: 04-08-2020
        // Rolf J. Godfrey s190813, Ulrik Hansen  s195091, Holger Bregnhøi Weise s195118, Victor Strauss s190811

#ifndef UART_H_
#define UART_H_
#define BAUD 115200
#define MYUBRRF F_CPU/8/BAUD-1   //full duplex
#define MYUBRRH F_CPU/16/BAUD-1   //half duplex
#define max 20 // number of data in the receive array
extern void putchUSART1 (char tx);
extern char getchUSART1(void);
extern void uart1_Init(unsigned int ubrr);
extern void enableReceice_Itr1();
#endif /* UART_H_ */
```