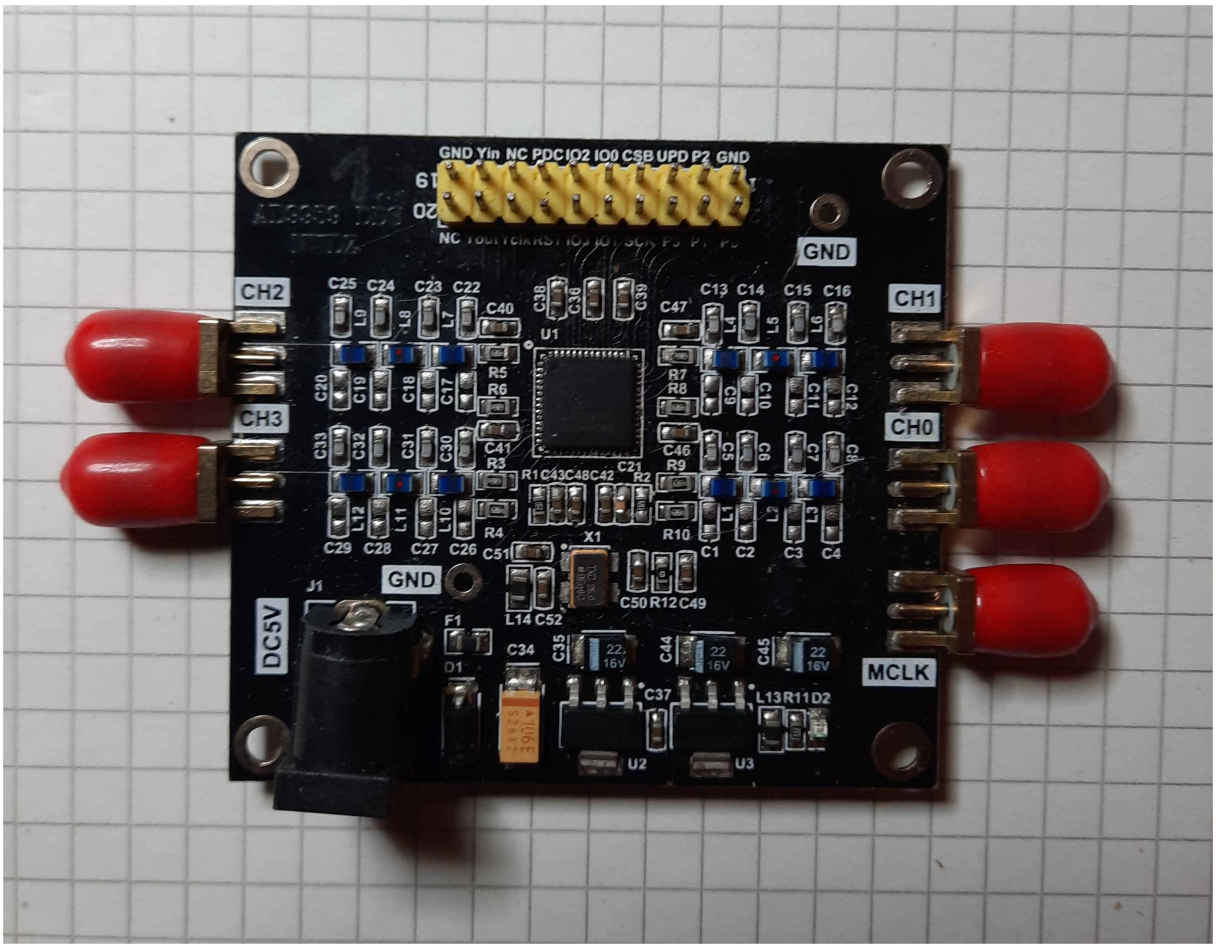
A multicannel analog output extension for Red Pitaya

For a scientific application our development department needed a sinudial signal in the range of 1 volt RMS, it was planned to use 4 to 16 channels or even more. Frequency, amplitude and phase should be set individual.

Because of my experience with Red Pitaya Devices and my good knowledge of the programming language C, I preferred this platform. Unfortunately, here we have only 2 fast analog outputs with 14 bit resolution and furthermore 4 analog outputs with reduced resolution.

The solution for this problem was the choice of a cheap DDS- Module. Those are offered by a wellknown trading place and known as „RF Signal Source AD9959 Signal Generator 4 Channel DDS Module os12“ or so. The price today is 75,02 €. Another brand I bought in 2021, known as „Nobsound“ only costs 66,99 €. In 2021 there were some supply shortages, so I ordered 2 different brands of modules, but in the electrical data they are similar.

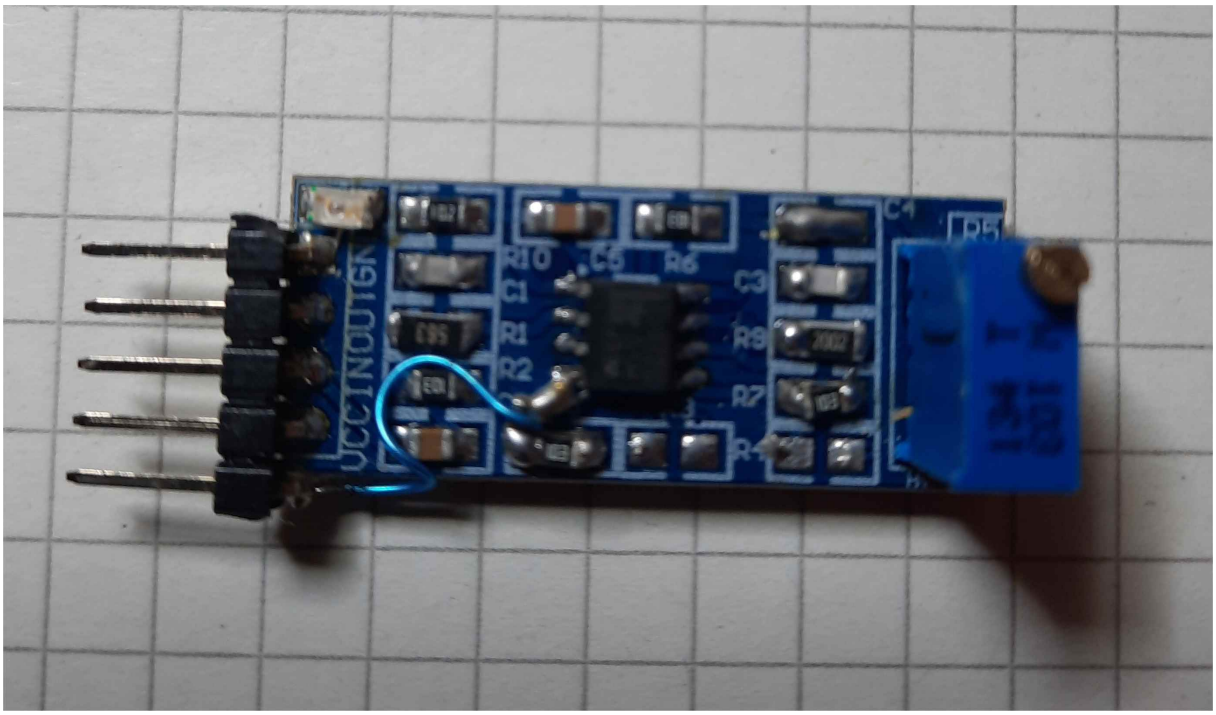
Here is a picture of one of these modules:



Important ist the yellow header on top, it offers an SPI-Interface and can be easily connected to a Red Pitaya or even to a Raspberry Pi or other Microcontroller. The knowledge therefore you can get from datasheet from Analog Devices, it is an AD 9959. This device has an SPI compatible interface and can be cascaded. To keep it synchronized, it is possible to spread out the clock of one master-modul or even therefore use an external clock. The output signal

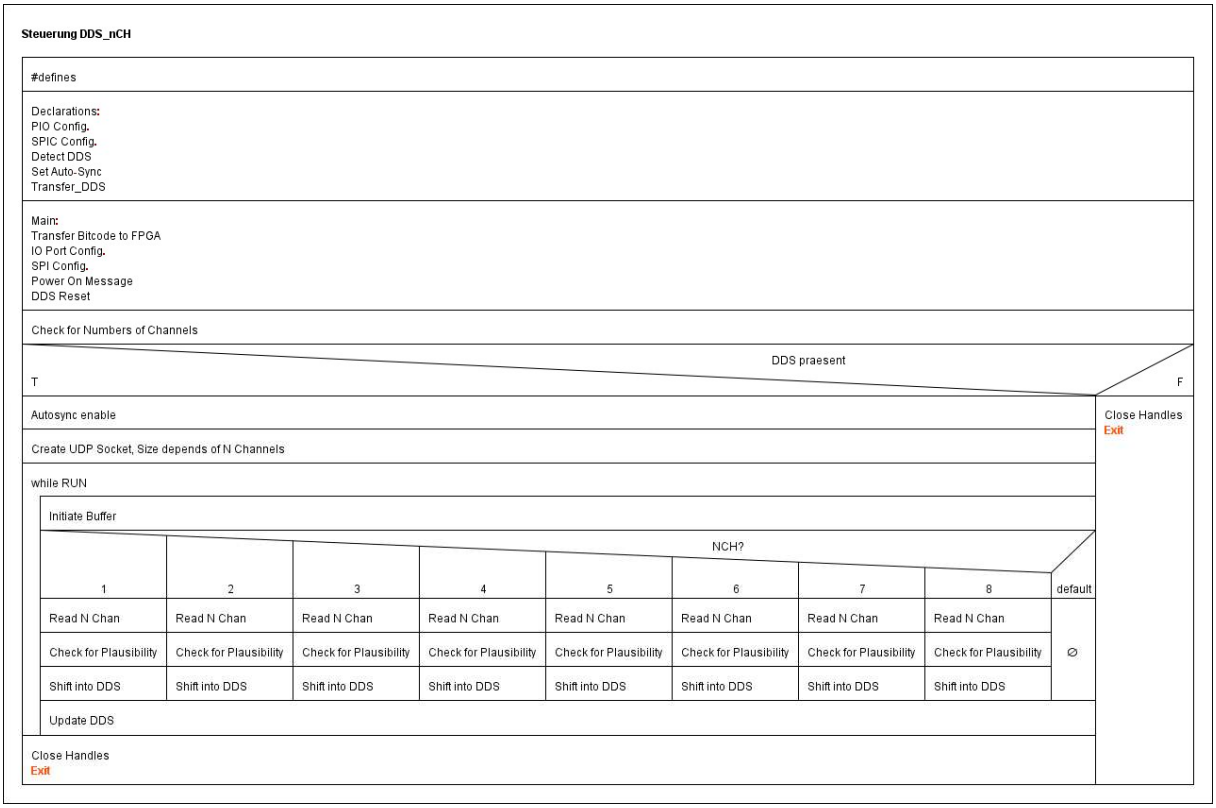
is about some 100 mV and comprises of large DC-offset. For further application it is necessary to use an additional pre-amplifier, which is AC-coupled tot he output oft he AD 9959-module. Here I modified an amplifier-modul form Segor Company in Berlin, it is called „Signalverstärker-Modul“ and offered for 2,80 €. For better performance, I changed LM358 to TL072, perhaps there will be better solutions therefore. Furthermore I use symmetrical power supply.

Here is a picture of modified Signalverstärker-Modul:



But now here is the most interessesting part of my work, that ist the software, which can control two of DDS modules.

First a simplified structogram:



As we see, the analog channels are controlled by transfer through ethernet connection by using UDP. For each channel, frequency, phase and amplitude are sent from host and then transferred into the DDS. The decission, how many channels I want to use is made at the beginning of the program by passing it as start- argument.

Here follows the listing of the programm

```
// Universal-Version for 1 to 8 channels, only works with 2 DDS-modules
// n frequencies, n phases and n amplitude-values sent to channel 0 to n-1 for DDS , via UDP/IP as
variable integer-vektor
//  JK 20.10.20
//  startparameter: number of channels
// here depends on: size of UDP-packet; number of dds modules and their selection and recognition

#include <sys/stat.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <errno.h>
#include <linux/spi/spidev.h>

#include <string.h>
#include <stdlib.h>

#include <math.h>


static const char *device = "/dev/spidev1.0";
static unsigned char mode;
static unsigned char bits = 8;
static unsigned int speed = 5000000;
static unsigned int delay;


// switch for debug messages
#define DEBUG 0

/* port for requests */
#define PORT 1234


/* size of buffer */
#define BUFSIZE 64

#define schreiben 0x00   // write
#define lesen 0x80       // read
#define fullduplex 0x02
#define RES1 968         // reset modul1
#define RES2 969         // reset modul2
#define SEL1 970         // select modul1
#define SEL2 971         // select modul2
#define UPD 972          // update all outputs

#define IN  0
#define OUT 1

#define LOW  0
#define HIGH 1

#define VALUE_MAX 30
#define BUFFER_MAX 3

#define MAX_PATH 64
```

```
static int pin_export(int pin)
{
        char shell[MAX_PATH];
        sprintf(shell,"echo %d > /sys/class/gpio/export", pin);
        system(shell);
        return 0;
}

static int pin_unexport(int pin)
{
        char shell[MAX_PATH];
        sprintf(shell,"echo %d > /sys/class/gpio/unexport", pin);
        system(shell);

        return 0;
}

static int pin_direction(int pin, int dir){

        char shell[MAX_PATH];
        snprintf(shell, MAX_PATH, "echo %s >
/sys/class/gpio/gpio%d/direction",((dir==IN)?"in":"out"),pin);
        system(shell);

        return 0;
}

static int pin_write(int pin, int value)
{
        char path[VALUE_MAX];
        int pin_fd;

        snprintf(path, VALUE_MAX, "/sys/class/gpio/gpio%d/value", pin);
        // get pin value file descrptor
        pin_fd = open(path, O_WRONLY);
        if (-1 == pin_fd) {
                fprintf(stderr, "Unable to to open sysfs pins value file %s for writing\n",path);
                return -1;
        }
        if(value==LOW){
                //write low
                if (1 != write(pin_fd, "0", 1)) {
                        fprintf(stderr, "Unable to write value\n");
                        return -1;
                }
        }
        else if(value==HIGH){
                //write high
                if (1 != write(pin_fd, "1", 1)) {
                        fprintf(stderr, "Unable to write value\n");
                        return -1;
                }
        }else fprintf(stderr, "Nonvalid pin value requested\n");

        //close file
        close(pin_fd);
        return 0;
}

static int spi_transfer(int fd, void *out, void *in, int len)
{
/* write and read on SPI. parameter:
 * fd        Devicehandle
```

```
 * data      buffer with data to send, overwritten with received data
 * length    size of buffer
struct spi_ioc_transfer {
  __u64  tx_buf;
  __u64  rx_buf;

  __u32  len;
  __u32  speed_hz;

  __u16  delay_usecs;
  __u8   bits_per_word;
  __u8   cs_change;
  __u32  pad;

*/

  int ret;
  struct spi_ioc_transfer buff; /* struct from library for writing */
  memset(&buff,0,sizeof(buff));

 /* enquire length of word */
  ret = ioctl(fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
  if (ret < 0)
  {
    perror("error get length of word");
    exit(1);
  }
/* enquire datarate */
  ret = ioctl(fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
   if (ret < 0)
   {
     perror("error get speed");
     exit(1);
   }
 /* transfer data */
    buff.tx_buf = (__u64)(__u32)out;
    buff.rx_buf = (__u64)(__u32)in;
    buff.len = len;
    ret = ioctl(fd, SPI_IOC_MESSAGE(1), &buff) ;
    if(ret < 0)
    {
       perror("error send/receive - ioctl");
       exit(1);
    }
    return ret;
}

int detect_dds(int det_spi_fd, int choice)
{

    unsigned char det_buff_tx[16];
    unsigned char det_buff_rx[16];
    int i, det_laenge;
    int cs_auswahl;
    if(choice == 1)
        cs_auswahl = SEL1;
    else if(choice == 2)
       cs_auswahl = SEL2;
    else
       cs_auswahl = SEL1;

// procedure of access tot he first AD 9959:
// first write to addr 0 (instruction), value = 0x02 = fullduplex(data transfer)

//    printf("in Detect_DDS\n");
```

```
    det_laenge = 2;
    det_buff_tx[0] = schreiben | 0;
    det_buff_tx[1] = fullduplex;

    pin_write(cs_auswahl,LOW);
    spi_transfer(det_spi_fd,det_buff_tx, det_buff_rx,det_laenge);
    pin_write(cs_auswahl,HIGH);

// spi_transfer directly writes to buff_tx to 9959 and reads his last answer to buff_rx.

//     for(i = 0; i < det_laenge; i++)
//       printf("datacontent: %d\t%x\n",i,det_buff_rx[i]);

// An update is necessary after every operation of data, except when channel-select!

    pin_write(UPD,HIGH);
    usleep(1);
    pin_write(UPD,LOW);
    usleep(1);

// now switch to read adr 0. Following data in buff_tx[1] are irrelevant
    det_laenge =  2;
    det_buff_tx[0] = lesen | 0;
    det_buff_tx[1] = fullduplex;

    pin_write(cs_auswahl,LOW);
      spi_transfer(det_spi_fd,det_buff_tx, det_buff_rx,det_laenge);
    pin_write(cs_auswahl,HIGH);

 //   for(i = 0; i < det_laenge; i++)
 //     printf("datacontent: %d\t%x\n",i,det_buff_rx[i]);

// now there will be the last written datavalue appear in buff_rx[1] (fullduplex, that is 0x02)

    if(det_buff_rx[1] == fullduplex)
      return 0;
    else
      return -1;
 }

// auto-sync:

int set_automatic_sync(int sync_spi_fd)
{
    unsigned char sync_buff_tx[16];
    unsigned char sync_buff_rx[16];
    int sync_laenge;

// at first configure master, here automatic mode synchronisation
// Master gets in funktionsregister (2), bit 6 = 1.

// M
    printf("automatic sync executed\n");

    sync_laenge = 3;
    sync_buff_tx[0] = schreiben | 0x02; // fnct-reg2
    sync_buff_tx[1] = 0x00;
    sync_buff_tx[2] = 0x41; // activate at first master
    pin_write(SEL1,LOW);
      spi_transfer(sync_spi_fd,sync_buff_tx, sync_buff_rx,sync_laenge);
    pin_write(SEL1,HIGH);

    pin_write(UPD,HIGH);
```

```
    usleep(10);
    pin_write(UPD,LOW);
    usleep(10);

// S

    sync_laenge = 3;
    sync_buff_tx[0] = schreiben | 0x02;  // fnct-reg2
    sync_buff_tx[1] = 0x00;
    sync_buff_tx[2] = 0x01;    // slave as slave (redundant, but for clarify)
    pin_write(SEL2,LOW);
       spi_transfer(sync_spi_fd,sync_buff_tx, sync_buff_rx,sync_laenge);
    pin_write(SEL2,HIGH);

    pin_write(UPD,HIGH);
    usleep(10);
    pin_write(UPD,LOW);
    usleep(10);

// M

    sync_laenge = 3;
    sync_buff_tx[0] = schreiben | 0x02; // fnct-reg2
    sync_buff_tx[1] = 0x00;
    sync_buff_tx[2] = 0xC1; // now set auto-sync enable
    pin_write(SEL1,LOW);
         spi_transfer(sync_spi_fd,sync_buff_tx,sync_buff_rx,sync_laenge);
    pin_write(SEL1,HIGH);
     pin_write(UPD,HIGH);
    usleep(10);
    pin_write(UPD,LOW);
    usleep(10);

// S

   sync_laenge = 3;
   sync_buff_tx[0] = schreiben | 0x02; // FU-Reg2
   sync_buff_tx[1] = 0x00;
   sync_buff_tx[2] = 0x81;    // now set auto-sync enable
   pin_write(SEL2,LOW);
       spi_transfer(sync_spi_fd,sync_buff_tx, sync_buff_rx,sync_laenge);
   pin_write(SEL2,HIGH);

    pin_write(UPD,HIGH);
    usleep(10);
    pin_write(UPD,LOW);
    usleep(10);

    return 0;
}

int TransferDDS(int tra_spi_fd, int marker, int chan, long t_data)
{
   int tra_laenge;
   unsigned char tra_buff_tx[16];
   unsigned char tra_buff_rx[16];
   long tra_tw;
   int c_val;
   int cs_val;

    switch(chan)
    {
      case 0:
        c_val = 0x10;
        cs_val = SEL1;
```

```
      break;
      case 1:
        c_val = 0x20;
        cs_val = SEL1;
      break;
      case 2:
        c_val = 0x40;
        cs_val = SEL1;
      break;
      case 3:
         c_val = 0x80;
        cs_val = SEL1;
      break;
      case 4:
        c_val = 0x10;
        cs_val = SEL2;
      break;
      case 5:
        c_val = 0x20;
        cs_val = SEL2;
      break;
      case 6:
        c_val = 0x40;
        cs_val = SEL2;
      break;
      case 7:
        c_val = 0x80;
        cs_val = SEL2;
      break;
      default:
        c_val = 0x10;
        cs_val = SEL1;
      break;
    }

// frequencies

    if(marker == 4)
    {
       tra_laenge = 2;
       tra_buff_tx[0] = schreiben | 0;
       tra_buff_tx[1] = fullduplex | c_val;
       pin_write(cs_val,LOW);
          spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx,tra_laenge);
       pin_write(cs_val,HIGH);

       tra_tw = t_data;
       tra_laenge = 5;
       tra_buff_tx[0] = schreiben | 0x04;
       tra_buff_tx[1] = ((tra_tw >> 24) & 0x0FF);
       tra_buff_tx[2] = ((tra_tw >> 16) & 0x0FF);
       tra_buff_tx[3] = ((tra_tw >> 8 ) & 0x0FF);
       tra_buff_tx[4] = (tra_tw & 0x0FF);

       pin_write(cs_val,LOW);
         spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx, tra_laenge);
       pin_write(cs_val,HIGH);
    }

    // phases

    if(marker == 5)
    {
      tra_laenge = 2;
      tra_buff_tx[0] = schreiben | 0;
```

```
      tra_buff_tx[1] = fullduplex | c_val;
      pin_write(cs_val,LOW);
         spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx, tra_laenge);
      pin_write(cs_val,HIGH);

      tra_tw = t_data;
      tra_laenge = 3;
      tra_buff_tx[0] = schreiben | 0x05;
      tra_buff_tx[1] = ((tra_tw >> 8) & 0x03F);
      tra_buff_tx[2] = (tra_tw  & 0x0FF);

      pin_write(cs_val,LOW);
        spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx, tra_laenge);
      pin_write(cs_val,HIGH);

    }

// amplitudes

    if(marker == 6)
    {
      tra_laenge = 2;
      tra_buff_tx[0] = schreiben | 0;
      tra_buff_tx[1] = fullduplex | c_val;
      pin_write(cs_val,LOW);
         spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx, tra_laenge);
      pin_write(cs_val,HIGH);

      tra_tw = t_data;
      tra_tw |= 0x1000; // enable amplitude-multiplier
      tra_laenge = 4;
      tra_buff_tx[0] = schreiben | 0x06;
      tra_buff_tx[1] =0x00;
      tra_buff_tx[2] = ((tra_tw >> 8) & 0x013);
      tra_buff_tx[3] = (tra_tw  & 0x0FF);

      pin_write(cs_val,LOW);
        spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx, tra_laenge);
      pin_write(cs_val,HIGH);

    }

// determine phasekohaence

      tra_laenge = 2;
      tra_buff_tx[0] = schreiben | 0;
      tra_buff_tx[1] = fullduplex | c_val;
      pin_write(cs_val,LOW);
         spi_transfer(tra_spi_fd, tra_buff_tx, tra_buff_rx, tra_laenge);
      pin_write(cs_val,HIGH);

      tra_laenge = 4;
      tra_buff_tx[0] = schreiben | 0x03; // CFR Reg.
      tra_buff_tx[1] = 0x00;
      tra_buff_tx[2] = 0x03; // default value
      tra_buff_tx[3] = 0x04;  //  activate auto phase acc

      pin_write(cs_val,LOW); // write master
         spi_transfer(tra_spi_fd,tra_buff_tx, tra_buff_rx,tra_laenge);
      pin_write(cs_val,HIGH);

      return 0;
}
```

```
int main (int argc, char **argv)
{
    int ret, laenge;
    long tw,tw_mirror;
    int int_spi_fd;
    int sock, client, rc, len,i,k,run,kanalanzahl;
    struct sockaddr_in cliAddr, servAddr;
    char udp_rx_buf[BUFSIZE];
    char udp_tx_buf[BUFSIZE];
    char *ptr;
    const int y = 1; /* for setsockopt */
    int parameter[24];
    float frequenz,amplitude;
    unsigned char chan_ctrl;

    unsigned char buff_tx[16];
    unsigned char buff_rx[16];
    int exflag;

   system("cat /opt/redpitaya/fpga/classic/fpga.bit > /dev/xdevcfg");

// place here modified code with conjunction of select wires
// or run generally with 2 modules.
   if(argc == 2)
   {
      kanalanzahl =  atoi(argv[1]);
      if((kanalanzahl < 1) || (kanalanzahl > 8))
      {
          printf("please select number of channels between1 and 8\n");
          exit(1);
      }
   }
   else
   {
      printf("please enter parameter for number of channels (1-8 )\n");
      exit(1);
   }

// from here nr of channels may be important!
// or use 2 modules fixed.


/* Prepare IO Ports */

   pin_export(RES1);
   pin_export(RES2);
   pin_export(SEL1);
   pin_export(SEL2);
   pin_export(UPD);

   pin_direction(RES1,OUT);
   pin_direction(RES2,OUT);
   pin_direction(SEL1,OUT);
   pin_direction(SEL2,OUT);
   pin_direction(UPD,OUT);

   pin_write(RES1,LOW);
   pin_write(RES2,LOW);
   pin_write(SEL1,HIGH);
   pin_write(SEL2,HIGH);
   pin_write(UPD,LOW);

// open SPI device:
```

```
    if ((int_spi_fd = open(device, O_RDWR)) < 0)
   {
      perror("error opening SPI device");
      exit(1);
   }

   /* set mode */
   ret = ioctl(int_spi_fd, SPI_IOC_WR_MODE, &mode);
   if (ret < 0)
   {
      perror("error set SPI-modue");
      exit(1);
   }
   /* request mode */
   ret = ioctl(int_spi_fd, SPI_IOC_RD_MODE, &mode);
   if (ret < 0)
   {
      perror("error get SPI-modue");
      exit(1);
   }
/* set length of word */
   ret = ioctl(int_spi_fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
   if (ret < 0)
   {
      perror("error set length of word");
      exit(1);
   }
   /* request length of word */
   ret = ioctl(int_spi_fd, SPI_IOC_RD_BITS_PER_WORD, &bits);
   if (ret < 0)
   {
      perror("error  get length of word");
      exit(1);
   }
/* set datarate */
   ret = ioctl(int_spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
   if (ret < 0)
   {
      perror("error set speed");
      exit(1);
   }
/* request datanrate */
   ret = ioctl(int_spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &speed);
   if (ret < 0)
   {
      perror("error get speed");
      exit(1);
   }
/* output for testing */
   printf("SPI-Device.....: %s\n", device);
   printf("SPI-Mode.......: %d\n", mode);
   printf("Length of word.....: %d\n", bits);
   printf("Speed: %d Hz (%d kHz)\n", speed, speed/1000);

   usleep(1);

// apply reset

   pin_write(RES1,HIGH);
   usleep(100);
   pin_write(RES1,LOW);
   usleep(100);

   pin_write(RES2,HIGH);
   usleep(100);
```

```
   pin_write(RES2,LOW);
   usleep(100);


// now find out, if both devices are present:
// therefor use universal routine ...

   if(detect_dds(int_spi_fd,1) == 0)
       printf("recognized board 1\n");
   else
   {
      printf("Error board 1\n");
      close(int_spi_fd);
      pin_unexport(RES1);
      pin_unexport(RES2);
      pin_unexport(SEL1);
      pin_unexport(SEL2);
      pin_unexport(UPD);

      return -1;
   }
   if(detect_dds(int_spi_fd,2) == 0)
       printf("recognized board 2\n");
   else
   {
      printf("Error board 2\n");
      close(int_spi_fd);
      pin_unexport(RES1);
      pin_unexport(RES2);
      pin_unexport(SEL1);
      pin_unexport(SEL2);
      pin_unexport(UPD);

      return -1;
    }

// enable synchronisation:

     set_automatic_sync(int_spi_fd);
     usleep(10);

// Start
   if(kanalanzahl == 1)
       printf("Programm was started with selection of %d channels!\n",kanalanzahl);
   else
     printf("Programm was started with selection of %d channels!\n",kanalanzahl);


// install UDP-buffer
   char *udp_buffer   = (char *) malloc(kanalanzahl * 8 * sizeof(char));

  /* install socket */
     sock = socket (AF_INET, SOCK_DGRAM, 0);
     if (sock < 0)
     {
       fprintf(stderr, "socket could not be opened\n");
       exit(1);
     }

  /* bind local server to port */
     memset(&servAddr, 0, sizeof (servAddr));
     servAddr.sin_family = AF_INET;
     servAddr.sin_addr.s_addr = htonl (INADDR_ANY);
     servAddr.sin_port = htons (PORT);
  /* allow immediate reuse of port */
```

```
     setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &y, sizeof(int));
     rc = bind(sock, (struct sockaddr *) &servAddr, sizeof (servAddr));
     if (rc < 0)
     {
       fprintf (stderr, "could not bind port\n");
       exit (1);
     }
     printf ("wait for data ...\n");

  /* serverloop */
     run = 1;
     while(run == 1)
     {


      /* initialize buffer */
        memset (udp_buffer, 0, kanalanzahl * 8);
    /* receive messages */
        len = sizeof (cliAddr);
        client = recvfrom (sock, udp_buffer, kanalanzahl * 8, 0,
                   (struct sockaddr *) &cliAddr, (socklen_t*)&len );
        if (client < 0)
        {
          fprintf(stderr, "could not receive data ...\n");
          continue;
        }

/* print out received message */
//        printf("Getting Data from %s, UDP-Port %u\n",
//            inet_ntoa(cliAddr.sin_addr), ntohs(cliAddr.sin_port));
//       for(i = 0; i < kanalanzahl*8; i++)
//
//        printf("Data: %d\n", udp_buffer[i]);


// convert content of buffer to integer:
// first 4 frequencies, they use 44 byte each
// calcualation:
//
//   fout = FTW*fs / 2^32
//   fout = frequency on output in Hz
//   FTW = frequency-tuning-word (32 bit raw-data for registers of AD9555)
//   fs = systemfrequency, here 25 MHz.
//   for 25 kHz the FTW is 2294967 (we cut off after decimal point, because it is precise enough)


// different cases for number of channels
    switch(kanalanzahl)
    {
       case 1:

// first frequencies,4 byte,

          parameter[0] = udp_buffer[0] + 256 * udp_buffer[1] + 65536 * udp_buffer[2] + 16777216 *
udp_buffer[3];



// then phase and amplitude, each 2 byte


          parameter[1] = udp_buffer[4] + 256 * udp_buffer[5];

          parameter[2] = udp_buffer[6] + 256 * udp_buffer[7];
```

```
#if DEBUG ==1

          for(i = 0; i< 3;i++)
             printf("%d\n",parameter[i]);
#endif

// check for abort, whenever a parameter is about MAX
// limit frequency to a proper value, (e.g. 1,16 MHz)

         if((parameter[0] > 2000000000) || (parameter[1] > 16383) || (parameter[2] > 1023))
          {
            run = 0;
            parameter[2] = 0;
          }

// from here on use function TransferDDS
// Parameter:
// Marker F,P,A =4,5,6
// Channel 0,1,2,3
// Data : Values for F,P,A

         TransferDDS(int_spi_fd,4,0,parameter[0]);

         TransferDDS(int_spi_fd,5,0,parameter[1]);

         TransferDDS(int_spi_fd,6,0,parameter[2]);

       break;

       case 2:
        parameter[0] = udp_buffer[0] + 256 * udp_buffer[1] + 65536 * udp_buffer[2] + 16777216 *
udp_buffer[3];
        parameter[1] = udp_buffer[4] + 256 * udp_buffer[5] + 65536 * udp_buffer[6] + 16777216 *
udp_buffer[7];

        parameter[2] = udp_buffer[8]  + 256 * udp_buffer[9];
        parameter[3] = udp_buffer[10] + 256 * udp_buffer[11];

        parameter[4] = udp_buffer[12] + 256 * udp_buffer[13];
        parameter[5] = udp_buffer[14] + 256 * udp_buffer[15];

#if DEBUG == 1

        for(i = 0; i< 6;i++)
            printf("%d\n",parameter[i]);
#endif

        if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 16383) ||
(parameter[3] > 16383)\
          || (parameter[4] > 1023) || (parameter[5] > 1023))
          {
            run = 0;
            parameter[4] = 0;
            parameter[5] = 0;
          }

         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);

         TransferDDS(int_spi_fd,5,0,parameter[2]);
         TransferDDS(int_spi_fd,5,1,parameter[3]);

         TransferDDS(int_spi_fd,6,0,parameter[4]);
         TransferDDS(int_spi_fd,6,1,parameter[5]);
```

```
       break;

       case 3:

         k = 0;
         for(i = 0; i < 3; i++)
         {
            parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k] + 65536 * udp_buffer[2+k] + 16777216
* udp_buffer[3+k];

            k += 4;
         }

         for(i= 3; i < 9; i++)
         {

           parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k];

           k += 2;
         }

#if DEBUG == 1

          for(i = 0; i < 9; i++)
             printf("%d\n",parameter[i]);

#endif

         if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 2000000000)
|| (parameter[3] > 16383)\
          || (parameter[4] > 16383) || (parameter[5] > 16383) || (parameter[6] > 1023) || (parameter[7]
> 1023) \
          || (parameter[8] > 1023))
          {
            run = 0;
            parameter[6] = 0;
            parameter[7] = 0;
            parameter[8] = 0;
         }

         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);
         TransferDDS(int_spi_fd,4,2,parameter[2]);

         TransferDDS(int_spi_fd,5,0,parameter[3]);
         TransferDDS(int_spi_fd,5,1,parameter[4]);
         TransferDDS(int_spi_fd,5,2,parameter[5]);

         TransferDDS(int_spi_fd,6,0,parameter[6]);
         TransferDDS(int_spi_fd,6,1,parameter[7]);
         TransferDDS(int_spi_fd,6,2,parameter[8]);


       break;

       case 4:
          k = 0;
          for(i = 0; i < 4; i++)
          {
             parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k] + 65536 * udp_buffer[2+k] +
16777216 * udp_buffer[3+k];

             k += 4;
          }
```

```
          for(i= 4; i < 12; i++)
          {

            parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k];

            k += 2;
          }
#if DEBUG == 1

          for(i = 0; i < 12; i++)
             printf("%d\n",parameter[i]);

#endif

         if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 2000000000)
|| (parameter[3] > 2000000000)\
          || (parameter[4] > 16383) || (parameter[5] > 16383) || (parameter[6] > 16383) ||
(parameter[7] > 16383) \
          || (parameter[8] > 1023) || (parameter[9] > 1023) || (parameter[10] > 1023) || (parameter[11]
> 1023))
          {
            run = 0;
            parameter[8] = 0;
            parameter[9] = 0;
            parameter[10] = 0;
           parameter[11] = 0;
         }


         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);
         TransferDDS(int_spi_fd,4,2,parameter[2]);
         TransferDDS(int_spi_fd,4,3,parameter[3]);

         TransferDDS(int_spi_fd,5,0,parameter[4]);
         TransferDDS(int_spi_fd,5,1,parameter[5]);
         TransferDDS(int_spi_fd,5,2,parameter[6]);
         TransferDDS(int_spi_fd,5,3,parameter[7]);

         TransferDDS(int_spi_fd,6,0,parameter[8]);
         TransferDDS(int_spi_fd,6,1,parameter[9]);
         TransferDDS(int_spi_fd,6,2,parameter[10]);
         TransferDDS(int_spi_fd,6,3,parameter[11]);


     break;

     case 5:

          k = 0;
          for(i = 0; i < 5; i++)
          {
             parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k] + 65536 * udp_buffer[2+k] +
16777216 * udp_buffer[3+k];

             k += 4;
          }

          for(i= 5; i < 15; i++)
          {
            parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k];

            k += 2;
          }
```

```
#if DEBUG == 1

          for(i = 0; i < 15; i++)
             printf("%d\n",parameter[i]);

#endif

         if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 2000000000)
|| (parameter[3] > 2000000000) \
          ||(parameter[4] > 2000000000) \
          || (parameter[5] > 16383) || (parameter[6] > 16383) || (parameter[7] > 16383) ||
(parameter[8] > 16383) || (parameter[9] > 16383) \
          || (parameter[10] > 1023) || (parameter[11] > 1023) || (parameter[12] > 1023) ||
(parameter[13] > 1023) || (parameter[14] > 1023))
          {
            run = 0;
            parameter[10] = 0;
            parameter[11] = 0;
            parameter[12] = 0;
            parameter[13] = 0;
            parameter[14] = 0;
         }

         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);
         TransferDDS(int_spi_fd,4,2,parameter[2]);
         TransferDDS(int_spi_fd,4,3,parameter[3]);
         TransferDDS(int_spi_fd,4,4,parameter[4]);

         TransferDDS(int_spi_fd,5,0,parameter[5]);
         TransferDDS(int_spi_fd,5,1,parameter[6]);
         TransferDDS(int_spi_fd,5,2,parameter[7]);
         TransferDDS(int_spi_fd,5,3,parameter[8]);
         TransferDDS(int_spi_fd,5,4,parameter[9]);

         TransferDDS(int_spi_fd,6,0,parameter[10]);
         TransferDDS(int_spi_fd,6,1,parameter[11]);
         TransferDDS(int_spi_fd,6,2,parameter[12]);
         TransferDDS(int_spi_fd,6,3,parameter[13]);
         TransferDDS(int_spi_fd,6,4,parameter[14]);

     break;

     case 6:

          k = 0;
          for(i = 0; i < 6; i++)
          {
             parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k] + 65536 * udp_buffer[2+k] +
16777216 * udp_buffer[3+k];

             k += 4;
          }

          for(i= 6; i < 18; i++)
          {

            parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k];

            k += 2;
          }

#if DEBUG == 1

          for(i = 0; i < 18; i++)
```

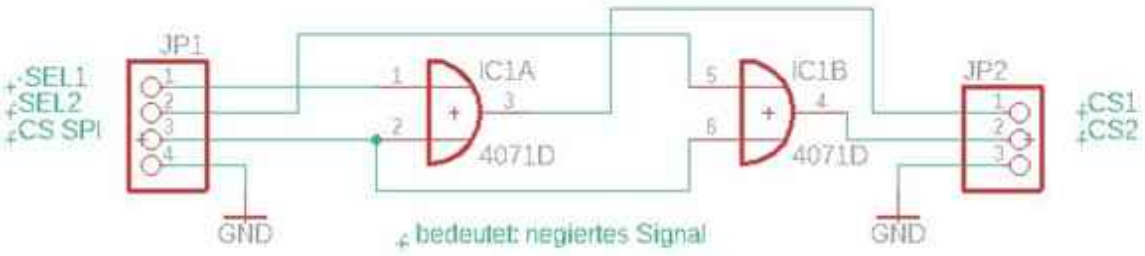
```
            printf("%d\n",parameter[i]);

#endif

         if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 2000000000)
|| (parameter[3] > 2000000000)\
          ||(parameter[4] > 2000000000) || (parameter[5] > 2000000000) \
          || (parameter[6] > 16383) || (parameter[7] > 16383) || (parameter[8] > 16383) ||
(parameter[9] > 16383) || (parameter[10] > 16383) \
          || (parameter[11] > 16383) \
          || (parameter[12] > 1023) || (parameter[13] > 1023) || (parameter[14] > 1023) ||
(parameter[15] > 1023) || (parameter[16] > 1023) \
          || (parameter[17] > 1023))
          {
            run = 0;
            parameter[12] = 0;
            parameter[13]  = 0;
            parameter[14] = 0;
            parameter[15] = 0;
            parameter[16] = 0;
            parameter[17] = 0;

         }

         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);
         TransferDDS(int_spi_fd,4,2,parameter[2]);
         TransferDDS(int_spi_fd,4,3,parameter[3]);
         TransferDDS(int_spi_fd,4,4,parameter[4]);
         TransferDDS(int_spi_fd,4,5,parameter[5]);

         TransferDDS(int_spi_fd,5,0,parameter[6]);
         TransferDDS(int_spi_fd,5,1,parameter[7]);
         TransferDDS(int_spi_fd,5,2,parameter[8]);
         TransferDDS(int_spi_fd,5,3,parameter[9]);
         TransferDDS(int_spi_fd,5,4,parameter[10]);
         TransferDDS(int_spi_fd,5,5,parameter[11]);

         TransferDDS(int_spi_fd,6,0,parameter[12]);
         TransferDDS(int_spi_fd,6,1,parameter[13]);
         TransferDDS(int_spi_fd,6,2,parameter[14]);
         TransferDDS(int_spi_fd,6,3,parameter[15]);
         TransferDDS(int_spi_fd,6,4,parameter[16]);
         TransferDDS(int_spi_fd,6,5,parameter[17]);

     break;

     case 7:

          k = 0;
          for(i = 0; i < 7; i++)
          {
             parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k] + 65536 * udp_buffer[2+k] +
16777216 * udp_buffer[3+k];

             k += 4;
          }

          for(i= 7; i < 21; i++)
          {

            parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k];

            k += 2;
          }
```

```
#if DEBUG == 1

          for(i = 0; i < 21; i++)
             printf("%d\n",parameter[i]);

#endif

         if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 2000000000)
|| (parameter[3] > 2000000000)\
          ||(parameter[4] > 2000000000) || (parameter[5] > 2000000000) || (parameter[6] > 2000000000) \
          || (parameter[7] > 16383) || (parameter[8] > 16383) || (parameter[9] > 16383) ||
(parameter[10] > 16383) || (parameter[11] > 16383) \
          || (parameter[12] > 16383)|| (parameter[13] > 16383)  \
          || (parameter[14] > 1023) || (parameter[15] > 1023) || (parameter[16] > 1023) ||
(parameter[17] > 1023) || (parameter[18] > 1023) \
          || (parameter[19] > 1023) || (parameter[20] > 1023))
          {
            run = 0;
            parameter[14] = 0;
            parameter[15] = 0;
            parameter[16] = 0;
            parameter[17] = 0;
            parameter[18] = 0;
            parameter[19] = 0;
            parameter[20] = 0;
         }

         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);
         TransferDDS(int_spi_fd,4,2,parameter[2]);
         TransferDDS(int_spi_fd,4,3,parameter[3]);
         TransferDDS(int_spi_fd,4,4,parameter[4]);
         TransferDDS(int_spi_fd,4,5,parameter[5]);
         TransferDDS(int_spi_fd,4,6,parameter[6]);

         TransferDDS(int_spi_fd,5,0,parameter[7]);
         TransferDDS(int_spi_fd,5,1,parameter[8]);
         TransferDDS(int_spi_fd,5,2,parameter[9]);
         TransferDDS(int_spi_fd,5,3,parameter[10]);
         TransferDDS(int_spi_fd,5,4,parameter[11]);
         TransferDDS(int_spi_fd,5,5,parameter[12]);
         TransferDDS(int_spi_fd,5,6,parameter[13]);

         TransferDDS(int_spi_fd,6,0,parameter[14]);
         TransferDDS(int_spi_fd,6,1,parameter[15]);
         TransferDDS(int_spi_fd,6,2,parameter[16]);
         TransferDDS(int_spi_fd,6,3,parameter[17]);
         TransferDDS(int_spi_fd,6,4,parameter[18]);
         TransferDDS(int_spi_fd,6,5,parameter[19]);
         TransferDDS(int_spi_fd,6,6,parameter[20]);

     break;

     case 8:

          k = 0;
          for(i = 0; i < 8; i++)
          {
             parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k] + 65536 * udp_buffer[2+k] +
16777216 * udp_buffer[3+k];

             k += 4;
          }
```

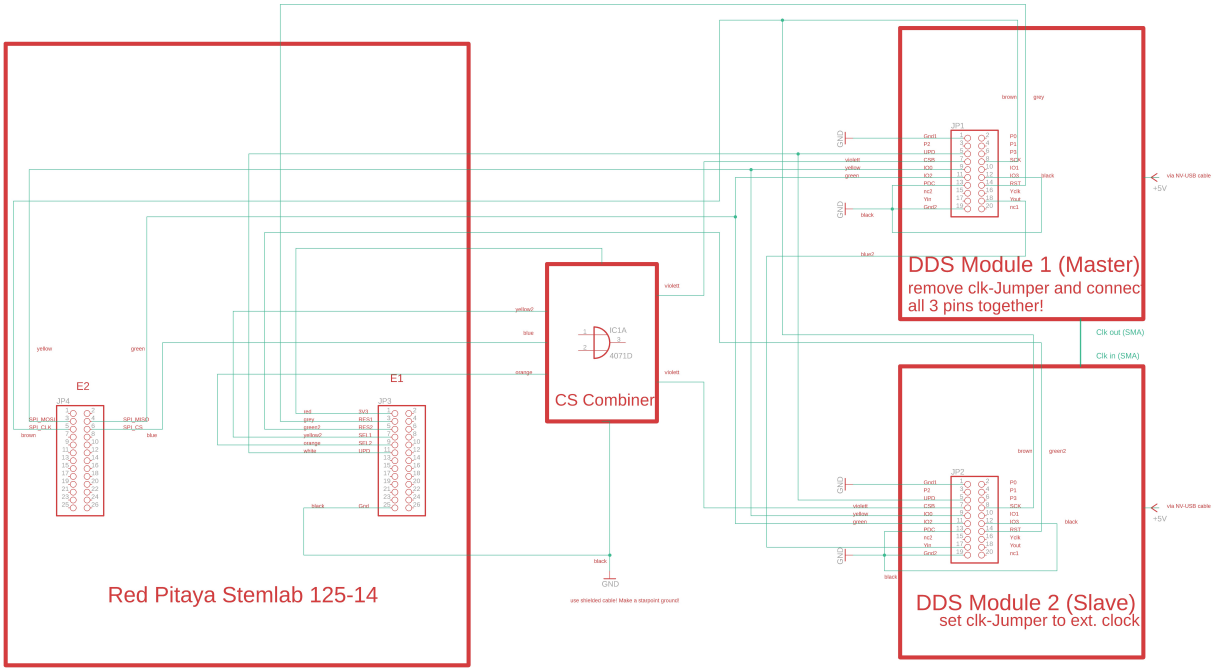
```
          for(i= 8; i < 24; i++)
          {

            parameter[i] = udp_buffer[0+k] + 256 * udp_buffer[1+k];

            k += 2;
          }

#if DEBUG == 1

          for(i = 0; i < 24; i++)
             printf("%d\n",parameter[i]);

#endif

         if((parameter[0] > 2000000000) || (parameter[1] > 2000000000) || (parameter[2] > 2000000000)
|| (parameter[3] > 2000000000)\
          ||(parameter[4] > 2000000000) || (parameter[5] > 2000000000) || (parameter[6] > 2000000000)
|| (parameter[7] > 2000000000)\
          || (parameter[8] > 16383) || (parameter[9] > 16383) || (parameter[10] > 16383) ||
(parameter[11] > 16383) || (parameter[12] > 16383) \
          || (parameter[13] > 16383)|| (parameter[14] > 16383) || (parameter[15] > 16383) \
          || (parameter[16] > 1023) || (parameter[17] > 1023) || (parameter[18] > 1023) ||
(parameter[19] > 1023) || (parameter[20] > 1023) \
          || (parameter[21] > 1023) || (parameter[22] > 1023) || (parameter[23] > 1023))
          {
            run = 0;
            parameter[16] = 0;
            parameter[17] = 0;
            parameter[18] = 0;
            parameter[19] = 0;
            parameter[20] = 0;
            parameter[21] = 0;
            parameter[22] = 0;
            parameter[23] = 0;
         }

         TransferDDS(int_spi_fd,4,0,parameter[0]);
         TransferDDS(int_spi_fd,4,1,parameter[1]);
         TransferDDS(int_spi_fd,4,2,parameter[2]);
         TransferDDS(int_spi_fd,4,3,parameter[3]);
         TransferDDS(int_spi_fd,4,4,parameter[4]);
         TransferDDS(int_spi_fd,4,5,parameter[5]);
         TransferDDS(int_spi_fd,4,6,parameter[6]);
         TransferDDS(int_spi_fd,4,7,parameter[7]);

         TransferDDS(int_spi_fd,5,0,parameter[8]);
         TransferDDS(int_spi_fd,5,1,parameter[9]);
         TransferDDS(int_spi_fd,5,2,parameter[10]);
         TransferDDS(int_spi_fd,5,3,parameter[11]);
         TransferDDS(int_spi_fd,5,4,parameter[12]);
         TransferDDS(int_spi_fd,5,5,parameter[13]);
         TransferDDS(int_spi_fd,5,6,parameter[14]);
         TransferDDS(int_spi_fd,5,7,parameter[15]);

         TransferDDS(int_spi_fd,6,0,parameter[16]);
         TransferDDS(int_spi_fd,6,1,parameter[17]);
         TransferDDS(int_spi_fd,6,2,parameter[18]);
         TransferDDS(int_spi_fd,6,3,parameter[19]);
         TransferDDS(int_spi_fd,6,4,parameter[20]);
         TransferDDS(int_spi_fd,6,5,parameter[21]);
         TransferDDS(int_spi_fd,6,6,parameter[22]);
         TransferDDS(int_spi_fd,6,7,parameter[23]);

     break;    // belongs to switch
```

```
     default:
     break;

    }  // end switch

// run IO-Update only once to ensure phasecohaerency!

    pin_write(UPD,HIGH);
    usleep(1);
    pin_write(UPD,LOW);
    usleep(1);

  }  // end run

  pin_unexport(RES1);
  pin_unexport(RES2);
  pin_unexport(SEL1);
  pin_unexport(SEL2);
  pin_unexport(UPD);
  free(udp_buffer);
  return (0);
}
```

Furthermore, we need a little hardware to combine SEL1,SEL2 and CS.



At last I show the connection scheme of 2 DDS boards to Red Pitaya:



So, that is all. Don't forget the modifications of master DDS. The boards, I use here (seen in in the first picture) don't have the Jumper mentioned in the circuit diagram above. Perhaps it is possible to connect the clock outputs together, after doing some modification on the master-board , I did'nt try it. If we need more than 2 DDS module, we have to add buffers to external signals, above all to clock and keep impedance to 50 ohms.

Joachim in February 2022.