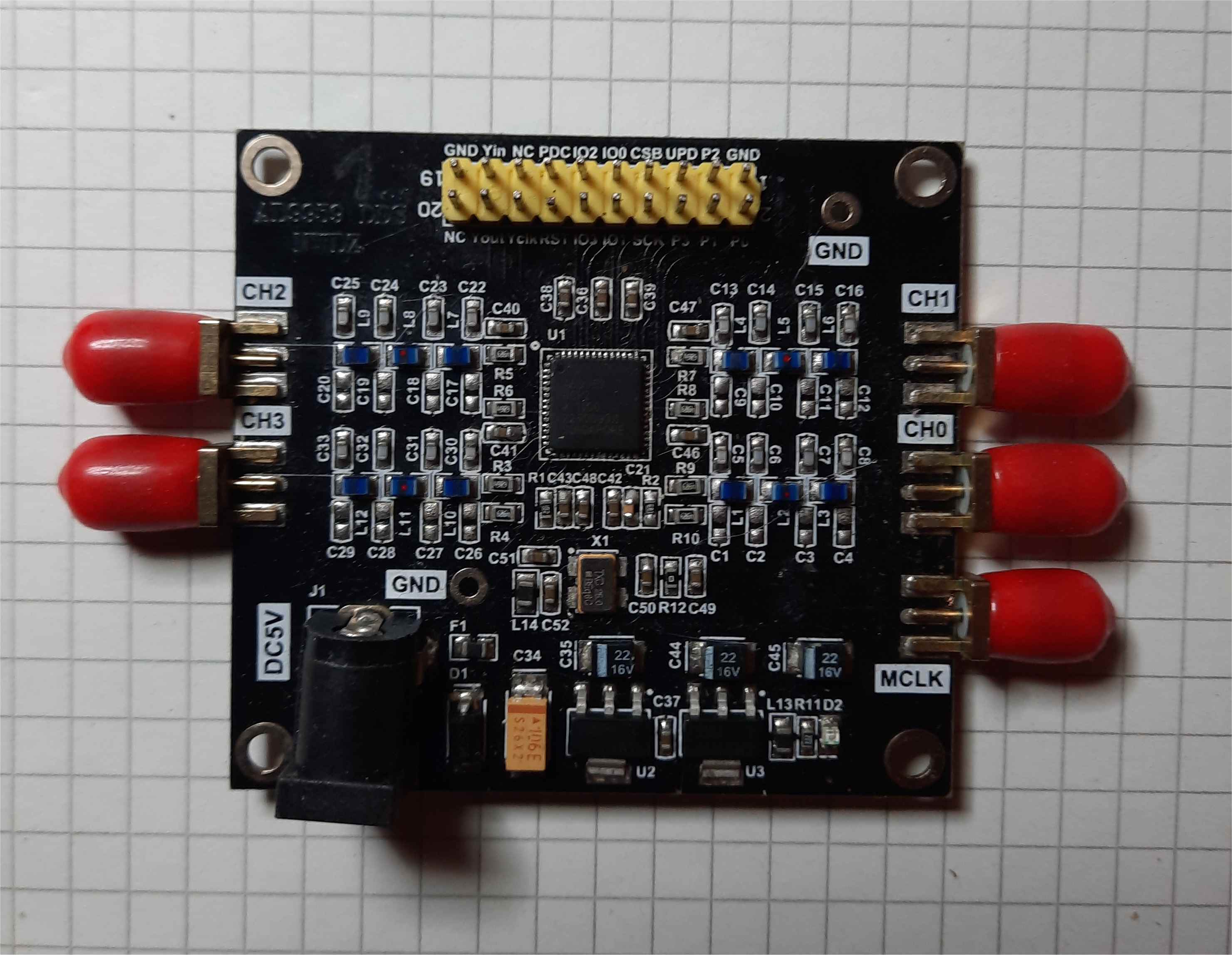
**A multicannel analog output extension for Red Pitaya**

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.

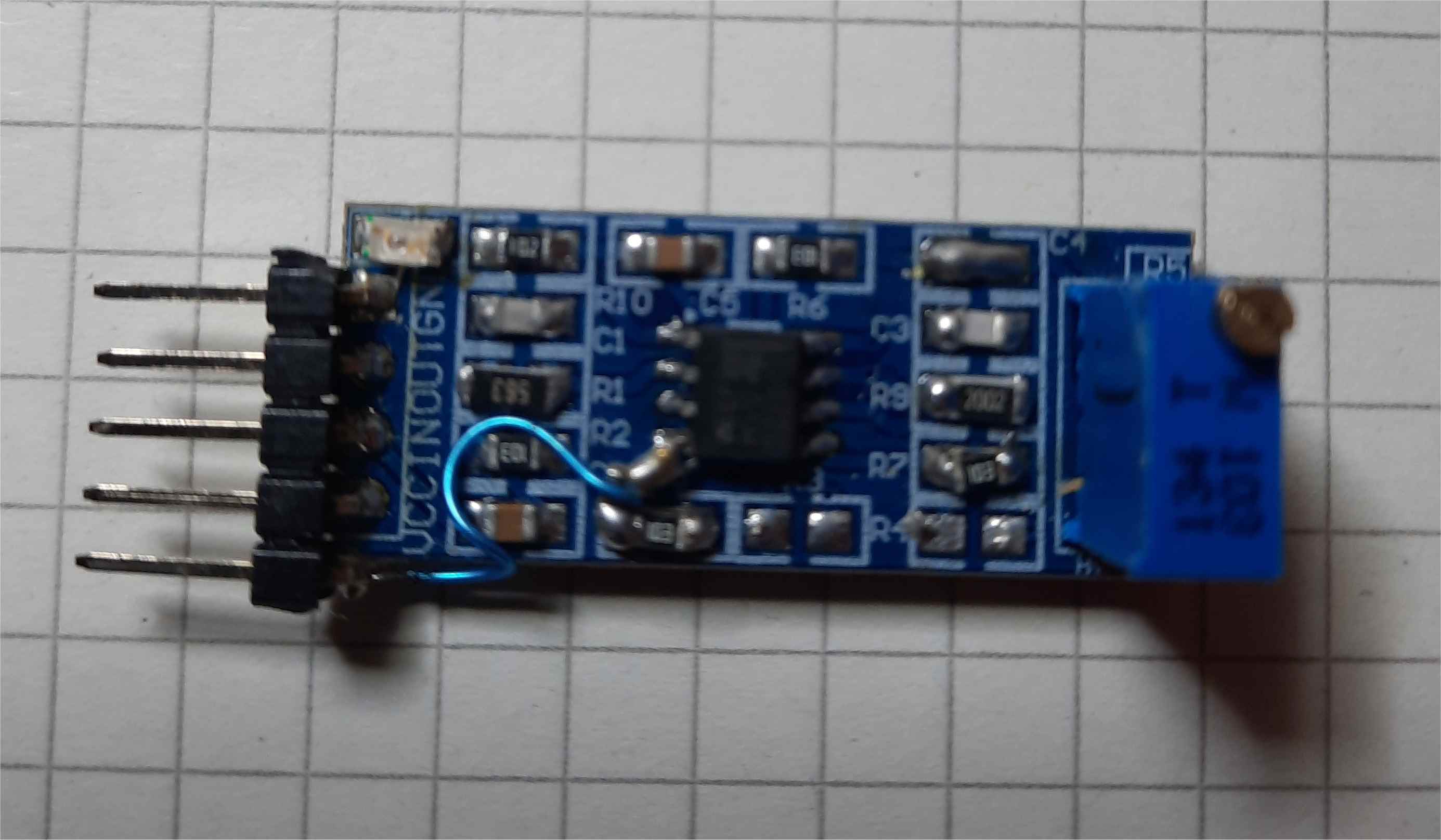
The solution for this problem was the choice of a cheap DDS- Module. Those are offered by a wellknown trading place and known as „RF Signal Source AD9959 Signal Generator 4 Channel DDS Module os12“ or so. The price today is 75,02 €. Another brand I bought in 2021, known as „Nobsound“ only costs 66,99 €. 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:



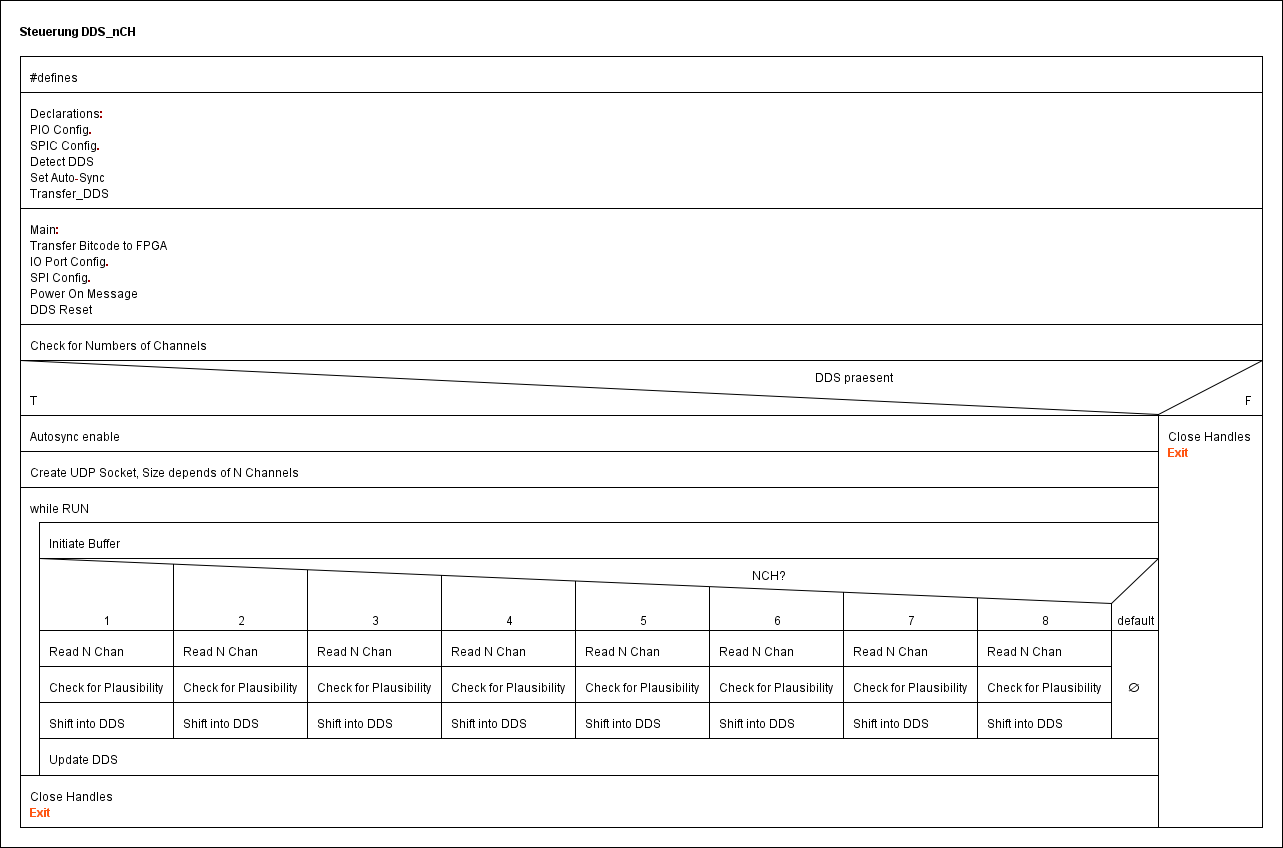
Important ist the yellow header on top, it offers an SPI-Interface and can be easily connected to a Red Pitaya or even to a Raspberry Pi or other Microcontroller. The knowledge therefore you can get from datasheet from Analog Devices, it is an AD 9959. This device has an SPI compatible interface and 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);

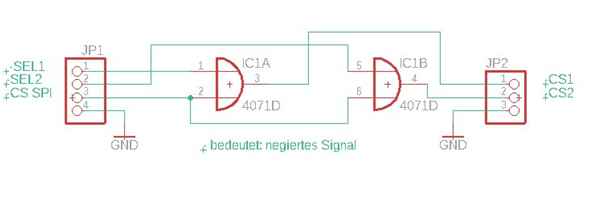
pin\_unexport(UPD);

free(udp\_buffer);

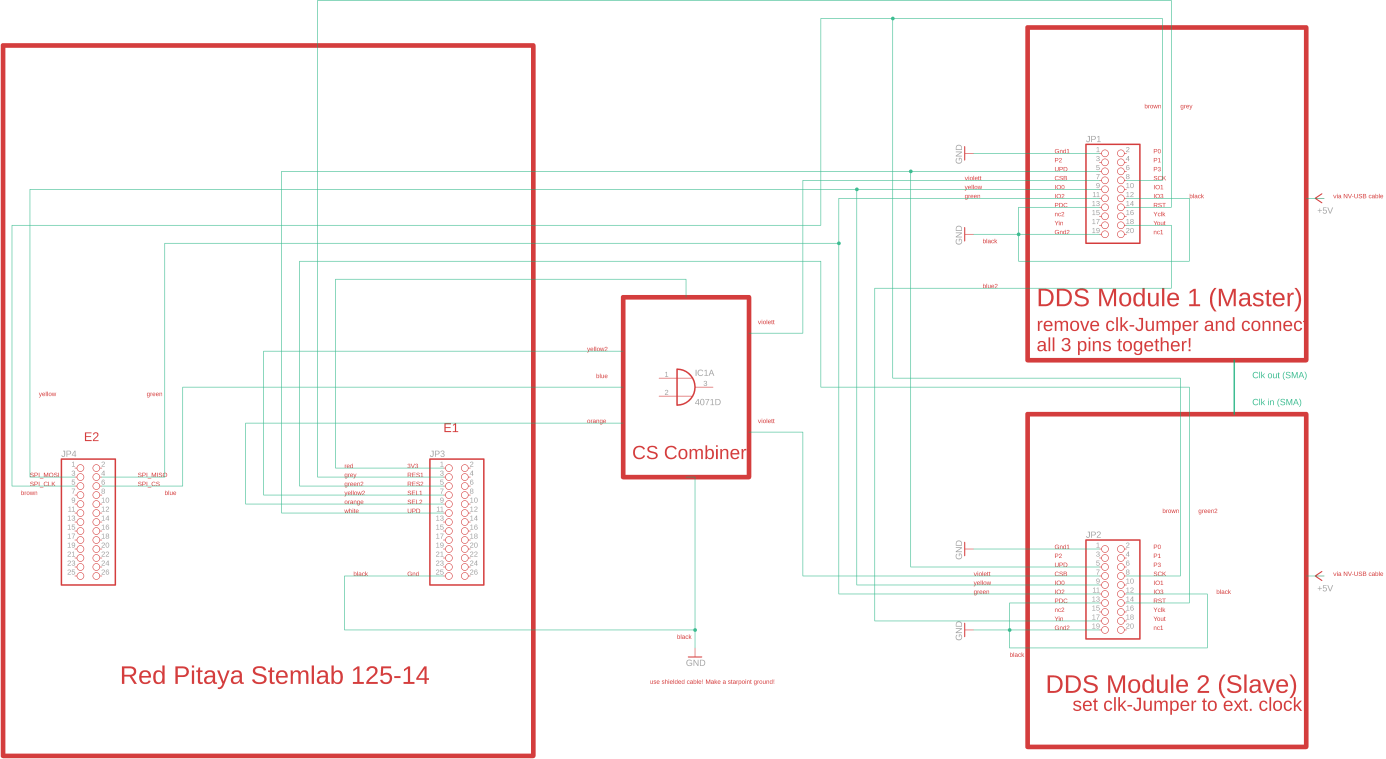
return (0);

}

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.