```
// ---------------------------------------------------------------------
// This is a firmware code for UWED
// Hardware platform rfDUINO
// Other components are two channel 16-bit DAC and one channel built in 10-bit ADC
// Developed by Alar Ainla, 2016-2018. Whitesides Group. Harvard University
// Commands are sent in ASCII text format, data is sent in binary format.
// ---------------------------------------------------------------------

//Include RFDUINO definitions
#include <RFduinoBLE.h>
//Use Wire library for I2C communications
#include <Wire.h>

//-------------------------------------------------------------
// --- DEVICE CONFIGURATIONS
//-------------------------------------------------------------
//I2C Pins
#define SDApin 2
#define SCLpin 3
//Switch 1: voltage vs current measurement (0-> Current, 1->Voltage). Digital output
#define SwitchIV 5
//Switch 2: reference electrode (0->3 electrode, 1->2 electrode). Digital output
#define SwitchEl 4
//Analog signal input pin
#define ADCIn 6

//--- BLE parameters ---
//Device BLE name
#define DEVICE_NAME "UWED"

//--- Other hardware definitions ---
//DAC reference electrode channel
#define DAC_RefCH 0
//DAC working electrode channel
#define DAC_WCH 1

//-------------------------------------------------------------
// --- ALL VARIABLES AND SETTINGS
//-------------------------------------------------------------

//--- System setting ---
volatile unsigned int refPot=0;             //Reference electrode potential DAC value
volatile unsigned int wPot=0;               //Working electrode potential DAC value
char inpMode=0;                             //Input mode 0-potentiometric, 1-amperometric
char nrEl=3;                                //Number of electrodes. 2 or 3
volatile uint_fast16_t timerBaseStep = 25;  //This is timer base step in ms (for interupts). Tested frequencies value --> Measured: 15 --> 67.5Hz, 10 --> 100Hz, 25 -->40.42Hz
volatile uint_fast16_t timerBaseStep1 = 25; //This is timer base step 1 (odd): this value is used for all measurements but DPV, in which case both timerBaseStep1 and 2 are used.
volatile uint_fast16_t timerBaseStep2 = 0;  //This is timer base step 2 (even): which is used only for DPV. In all other cases it should be 0 and only value "1" is used. Default
int refIncOdd=0;    //Reference channel increment at odd steps
int refIncEven=0;   //Reference channel increase at even steps
int wIncOdd=0;      //Working channel increase at odd steps
int wIncEven=0;     //Working channel increase at even steps
int numberOfSteps=0;   //Number of steps to make

//--- System variables ---
volatile bool sRunning=false;   //Currently in running state
volatile long t1000steps=0;     //time in ms what it took to make 1000steps
volatile int  N=0;              //Current step number
volatile unsigned int last_refPot=0;
volatile unsigned int last_wPot=0;
volatile unsigned int last_Inp=0;
volatile unsigned int last_N=0;
//These are actually the output values
volatile unsigned int outRefPot_odd=0;
volatile unsigned int outRefPot_even=0;
volatile unsigned int outInp_odd=0;
volatile unsigned int outInp_even=0;
volatile unsigned int outN=0;
volatile unsigned long outT=0;
volatile bool outSend=false;

volatile bool last_updated=false;     //Every time reading is updated this is pulled to true, when BLE sending is done it's pulled back to false
volatile bool cycleX=false;           //Every time interrupt is called this will switch polarity, these are two different parts of the process
volatile bool cycleDone=false;        //Every time scan is completed this is pulled true
volatile bool readADCNow=false;       //Read ADC now
volatile long errorcount=0;
volatile int t1000counter=0;
volatile long t0;
volatile long ADCsum=0; //Sum 10x ADC

// BLE data and information
bool connection = false;    // Connection will be true when in a BLE connection and false otherwise
bool advertising = false;   // Advertising will be true when the BLE radio is advertising and false otherwise
char datax[21];    //Data buffer
char outIndex='0'; //This is output index character, which goes from 0 to 9

//-------------------------------------------------------------
// --- MAIN DEVICE INITIALIZATION
//-------------------------------------------------------------
void setup()
{
  //BLE
  RFduinoBLE.deviceName = DEVICE_NAME;
  RFduinoBLE.begin();
  RFduinoBLE.send(datax, 20);
  delay(3000);
  //Other hardware
  initHardware();
  delay(2000);
  timerInit();
  analogReadResolution(10);
  pinMode(ADCIn,INPUT); //Input signal
  //Done
}

//-------------------------------------------------------------
// --- MAIN LOOP
//-------------------------------------------------------------
//This is running continuously, unless device is in timed interupt
void loop()
{
   if(cycleDone){ //Send message if one scan cycle is completed
        //If DPV mode return back to default timestep 1
        if(timerBaseStep2>0)
        {
```

```
          timerBaseStep=timerBaseStep1;
          setTimer();
        }
      sendResponse("m:DONE");
      cycleDone=false;
   }
   if(sRunning&&outSend) //New data gathered send it away
   {
      sendData();
      outSend=false;
   }
}

//-----------------------------------------------------------------
// --- INTERPRETING COMMANDS
//-----------------------------------------------------------------
//This is interpreting incoming commands by ID character
//param is numerical input parameter of the command
bool action(char ID, long param)
{
  switch(ID)
  {
    case 'A': //Set reference electrode potential value
              refPot=(unsigned int)param;
              setDAC(DAC_RefCH, refPot);
              sendResponse("a:DONE");
              return true;
    case 'B': //Set working electrode potential value
              wPot=(unsigned int)param;
              setDAC(DAC_WCH, wPot);
              sendResponse("b:DONE");
              return true;
    case 'C': //Set input mode (0-potentsiometric (default), 1-amperometric)
              inpMode=(char)param;
              digitalWrite(SwitchIV,inpMode==0);
              sendResponse("c:DONE");
              return true;
    case 'D': //Set number of electrodes used 2 or 3
              nrEl=(char)param;
              digitalWrite(SwitchEl,nrEl==2);
              sendResponse("d:DONE");
              return true;
    case 'E': //Set base timer 15 ... 1000 ms (timerBaseStep & timerBaseStep1)
              timerBaseStep=(uint_fast16_t)param;
              if(timerBaseStep<15){ timerBaseStep=15; }
              if(timerBaseStep>1000){ timerBaseStep=1000; }
              timerBaseStep1=timerBaseStep;
              timerInit();
              sendResponse("e:DONE");
              return true;
    case 'F': //Get time for 100 cycles
              sendResponse("f:"+String(t1000steps,DEC));
              return true;
    case 'G': //Set scan rate for electrode reference channel to increase at odd steps
              if(param>32767){ refIncOdd=32767; }else
              if(param<-32768){ refIncOdd=-32768; }else
              { refIncOdd=(int)param; }
              sendResponse("g:DONE");
              return true;
    case 'I': //Set scan rate for reference electrode channel to increase at even steps
              if(param>32767){ refIncEven=32767; }else
              if(param<-32768){ refIncEven=-32768; }else
              { refIncEven=(int)param; }
              sendResponse("i:DONE");
              return true;
              return true;
    case 'J': //Set scan rate for working electrode channel to increase at odd steps
              if(param>32767){ wIncOdd=32767; }else
              if(param<-32768){ wIncOdd=-32768; }else
              { wIncOdd=(int)param; }
              sendResponse("j:DONE");
              return true;
    case 'K': //Set scan rate for reference channel to increase at even steps
              if(param>32767){ wIncEven=32767; }else
              if(param<-32768){ wIncEven=-32768; }else
              { wIncEven=(int)param; }
              sendResponse("k:DONE");
              return true;
    case 'L': //Set number of step to make during the scan
              if(param>1000){ numberOfSteps=1000; }else
              if(param<1){ numberOfSteps=1; }else
              { numberOfSteps=(int)param; }
              sendResponse("l:DONE");
              return true;
    case 'M': //Run the sequence
              sRunning=true;
              N=0;
              return true;
    case 'N': //Halt the sequence
              sRunning=false;
              N=0;
              //If DPV mode return back to default timestep 1
              if(timerBaseStep2>0)
              {
                timerBaseStep=timerBaseStep1;
                timerInit();
              }
              sendResponse("n:DONE");
              return true;
    case 'O': //Set base timer 2 for DPV. 0 (equal step mode) or 15 ... 1000 ms (DPV mode) (timerBaseStep2)
              timerBaseStep2=(uint_fast16_t)param;
              if(timerBaseStep2>0)
              {
                if(timerBaseStep2<15){ timerBaseStep2=15; }
                if(timerBaseStep2>1000){ timerBaseStep2=1000; }
              }
              sendResponse("o:DONE");
              return true;
    default:
      return false;
  }
}

//-----------------------------------------------------------------
// --- PRECISION TIMING - Timer interupt on Timer 2
//-----------------------------------------------------------------
```

```
//This function initializes timer and set the period defined by timerBaseStep
void timerInit()
{
  NRF_TIMER2->TASKS_STOP = 1;  // Stop timer
  NRF_TIMER2->MODE = TIMER_MODE_MODE_Timer;  // taken from Nordic dev zone
  NRF_TIMER2->BITMODE = TIMER_BITMODE_BITMODE_16Bit;
  NRF_TIMER2->PRESCALER = 9;  // 32us resolution
  NRF_TIMER2->TASKS_CLEAR = 1; // Clear timer
  // With 32 us ticks, we need to multiply by 31.25 to get milliseconds
  NRF_TIMER2->CC[0] = timerBaseStep * 31;
  NRF_TIMER2->CC[0] += timerBaseStep / 4;
  NRF_TIMER2->INTENSET = TIMER_INTENSET_COMPARE0_Enabled << TIMER_INTENSET_COMPARE0_Pos;  // taken from Nordic dev zone
  NRF_TIMER2->SHORTS = (TIMER_SHORTS_COMPARE0_CLEAR_Enabled << TIMER_SHORTS_COMPARE0_CLEAR_Pos);
  attachInterrupt(TIMER2_IRQn, TIMER2_Interrupt);    // also used in variant.cpp to configure the RTC1
  NRF_TIMER2->TASKS_START = 1;  // Start TIMER
}

//Adjust timer value
void setTimer()
{
  NRF_TIMER2->CC[0] = timerBaseStep * 31;
  NRF_TIMER2->CC[0] += timerBaseStep / 4;
}

// This is the main timer interupt service, which is called every "timerBaseStep" ms.
void TIMER2_Interrupt(void)
{
  long tmp;
  if (NRF_TIMER2->EVENTS_COMPARE[0] != 0)
  {
    cycleX=!cycleX; //multiplexes every time

    //Measures timing
    if(t1000counter==0) t0=millis();
    if(t1000counter==1000) { t1000steps=millis()-t0; t1000counter=0; }else{ t1000counter++; }

    if(cycleX) //Main step
    {
        //Sum more ADC
        for(char i=0; i<128; i++) ADCsum=ADCsum+analogRead(ADCIn);
        last_refPot=refPot;
        last_wPot=wPot;
        last_Inp=ADCsum/4; //ADCvalue (256 x 10-bit input)/4 --> 16-bit end value;
        last_N=N;
        //Make steps if measurement is running
        if(sRunning)
        {
            N=N+1; //Increase step number
            if(N>numberOfSteps){ sRunning=false; cycleDone=true; } //Check if desigerd number of steps are done
        }
        //If running the update
        if(sRunning)
        {
            if(N%2==0) // If even step
            {
                //Set reference potential
                tmp=refPot; tmp=tmp+refIncEven; //Make step
                if(tmp>65535) { tmp=65535; }else if(tmp<0){ tmp=0; }else; // Check that end value is in the range
                refPot=(unsigned int)tmp; setDAC(DAC_RefCH, refPot); //Set value
                //Set working potential (same as reference)
                tmp=wPot; tmp=tmp+wIncEven;
                if(tmp>65535) { tmp=65535; }else if(tmp<0){ tmp=0; }else;
                wPot=(unsigned int)tmp; setDAC(DAC_WCH, wPot);
                //In small cycle even steps come last - push into even step register
                outRefPot_even=last_refPot;
                outInp_even=last_Inp;
                outN=last_N;
                outT=millis();
                outSend=true; //Small cycle done. Now this triggers sending
                //If DPV mode adjust also timestep, otherwise do nothing
                if(timerBaseStep2>0)
                {
                  timerBaseStep=timerBaseStep1; //1 is used for odd steps
                  setTimer();
                }
            }else // If odd step
            {
                //Set reference potential
                tmp=refPot; tmp=tmp+refIncOdd;
                if(tmp>65535) { tmp=65535; }else if(tmp<0){ tmp=0; }else;
                refPot=(unsigned int)tmp; setDAC(DAC_RefCH, refPot);
                //Set working potential
                tmp=wPot; tmp=tmp+wIncOdd;
                if(tmp>65535) { tmp=65535; }else if(tmp<0){ tmp=0; }else;
                wPot=(unsigned int)tmp; setDAC(DAC_WCH, wPot);
                //In small cycle odd steps come first - push into odd step register
                outRefPot_odd=last_refPot;
                outInp_odd=last_Inp;
                //If DPV mode adjust also timestep, otherwise do nothing
                if(timerBaseStep2>0)
                {
                  timerBaseStep=timerBaseStep2; //2 is used for even steps
                  setTimer();
                }
            }
        }
        ADCsum=0;
    }else //Second step - just measure ADC. ADC us sanoked 128x
    {
      for(char i=0; i<128; i++) ADCsum=ADCsum+analogRead(ADCIn);
    }
    NRF_TIMER2->EVENTS_COMPARE[0] = 0;
  }
}

//--------------------------------------------------------------
// --- BLE DATA COMMUNICATION
//--------------------------------------------------------------

// --- Other BLE functions -------------
void RFduinoBLE_onAdvertisement(bool start)
{
  advertising = start;
}
```

```
void RFduinoBLE_onConnect()
{
  connection = true;
}

void RFduinoBLE_onDisconnect()
{
  connection = false;
}

void RFduinoBLE_onReceive(char *data, int len)
{
    cmdLine(data,len); //Process
}

//Command line processor. Return false if command isn't right or true if successfully processed
//All commands have structure "X(param)", where "X" is command identifing capital ASCII character
//and paremeter is numerical parameter in text format. Some commands do not have parameter.
//but they always need to have parenthesis, thus "()"
bool cmdLine(char* data, int len)
{
    char ID;    //Command ID
    long param=0; //Parameter of the function. Default is zero
    char datac[25];
    int begi=-1, endi=-1; //begin and end index: BEGIN "(" and END ")"
    for(char i=0; i<len; i++)
    {
        if(data[i]=='(') begi=i;
        if(data[i]==')') endi=i;
    }
    if((begi>0)&&(endi>begi)) //Looks like valid command
    {
      ID=data[begi-1]; //ID character is just one before parenthesis
      if((endi-begi)>1) //There is also a parameter. Some commands dont have parameter
      {
        strncpy(&datac[0],&data[begi+1],endi-begi-1);
        datac[endi-begi]=0x00; //Null terminated
        param=atol(datac); //Convert parameter to number
      }
    }else
    {
      return false; //Something went wrong, not correct command syntax
    }
    //Now interprete the command and take action
    return action(ID, param);
}
//Send respons back (text format)
void sendResponse(String value)
{
    char tmpx[20];
    value.toCharArray(&tmpx[1],19); //prepare to send to BLE
    tmpx[0]=outIndex;
    outIndex++; if(outIndex>'9'){ outIndex='0'; }
    RFduinoBLE.send(tmpx,value.length()+1); //Send to BLE
}
//Response 2: this sends Data and uses mixed binary format.
void sendData()
{
    char tmpx[20];
    tmpx[0]=outIndex;
    outIndex++; if(outIndex>'9'){ outIndex='0'; }
    tmpx[1]='M';

    tmpx[2]=(char)(0xFF&(outRefPot_odd>>8));
    tmpx[3]=(char)(0xFF&outRefPot_odd);
    tmpx[4]=(char)(0xFF&(outInp_odd>>8));
    tmpx[5]=(char)(0xFF&outInp_odd);

    tmpx[6]=(char)(0xFF&(outRefPot_even>>8));
    tmpx[7]=(char)(0xFF&outRefPot_even);
    tmpx[8]=(char)(0xFF&(outInp_even>>8));
    tmpx[9]=(char)(0xFF&outInp_even);

    tmpx[10]=(char)(0xFF&(outT>>24));
    tmpx[11]=(char)(0xFF&(outT>>16));
    tmpx[12]=(char)(0xFF&(outT>>8));
    tmpx[13]=(char)(0xFF&(outT));

    tmpx[14]=(char)(0xFF&(last_wPot>>8));
    tmpx[15]=(char)(0xFF&last_wPot);

    tmpx[16]=(char)(0xFF&(outN>>8));
    tmpx[17]=(char)(0xFF&outN);

    tmpx[18]=0;
    tmpx[19]=0;

    RFduinoBLE.send(tmpx,20); //Send to BLE
}

//-----------------------------------------------------------------
// --- HARDWARE COMMUNICATION
//-----------------------------------------------------------------
// Initialize hardware, IO ports and ADC and
void initHardware()
{
    //Configure I2C bus
    Wire.speed=400;
    Wire.begin();
    Wire.beginOnPins(SCLpin, SDApin);
    //Initialize I2C DAC
    setupDAC(true);
    //Define switch pins
    pinMode(SwitchIV, OUTPUT);
    pinMode(SwitchEl, OUTPUT);
    digitalWrite(SwitchIV, HIGH); //Potential
    digitalWrite(SwitchEl, LOW);  //3-electrode
    //Set DACs
    setDAC(0, 0);
    setDAC(1, 0);
}

//Following commands are used to setup DAC
//We use: AD5667RBRMZ-2. 16-bit nanoDAC with dual channel and I2C interface

//Initialize DAC
```

```
void setupDAC(bool internalref)
{
  sendToDAC(0x20,0x00,0x03); //First setup command sets both DAC outputs into normal operation mode
  if(internalref) { sendToDAC(0x38,0x00,0x01); } //Use internal reference
  else { sendToDAC(0x38,0x00,0x00); } //Use external reference
  sendToDAC(0x30,0x00,0x03); //Dont use LDAC, update output by software
}

//Set output value of DACs
//channel 0 is channel A, channel 1 is channel B.
//value is 16-bit integer, which is set to DAC
//This command takes about 450us to operate
void setDAC(byte channel, unsigned int value)
{
  byte lowB=value&0xFF;
  byte highB=(value>>8)&0xFF;
  byte chB=0x18+channel;
  sendToDAC(chB,highB,lowB);
}

//Send a raw command to DAC
void sendToDAC(byte b1, byte b2, byte b3)
{
  byte error;
  Wire.beginTransmission(0x0F); //DAC address pin is connected to ground
  Wire.write(b1); Wire.write(b2); Wire.write(b3);
  error = Wire.endTransmission();
}
```