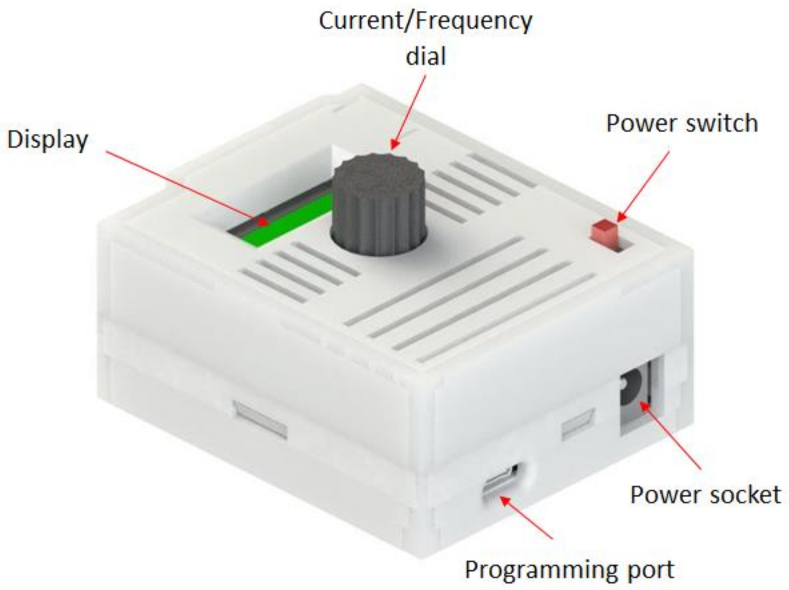
Reference Manual



Description

OpenXstim is a dual-channel programmable constant current stimulator for transcutaneous electrical stimulation. The stimulator is capable of delivering up to 100 mA of biphasic current at a maximum of 10kHz stimulation frequency for each stimulation channel simultaneously. The stimulator board is designed as a popular Arduino Uno shield (recommended to use Arduino uno r4 minima) with pin-to-pin capability for easy programming and operation. An on-board rotary encoder also allows manual control of the stimulation intensity. The stimulation current is automatically sensed by the microcontroller and displayed on an on-board display.

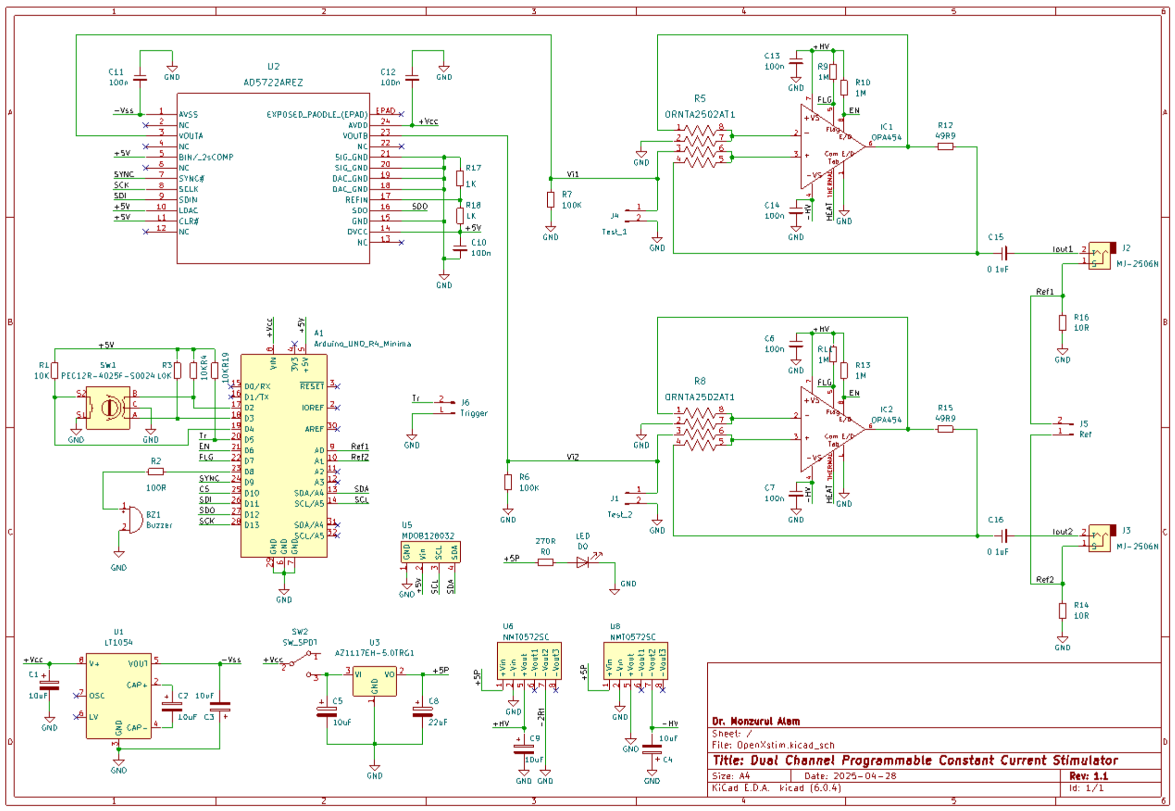
Specifications

- Dimensions: 6.86 cm x 5.33 cm (Arduino Uno Shield)
- Current intensity: 0 – 110 mA (per channel)
- Compliance voltage: ±48 volt
- Stimulation frequency: up-to 10kHz
- Compatibility: Arduino Uno
- Idle bias current: 50 mA
- Full-load supply current: 260 mA
- Supply voltage: 6 – 12 volts DC (preferred: 9V 650 mAh battery)

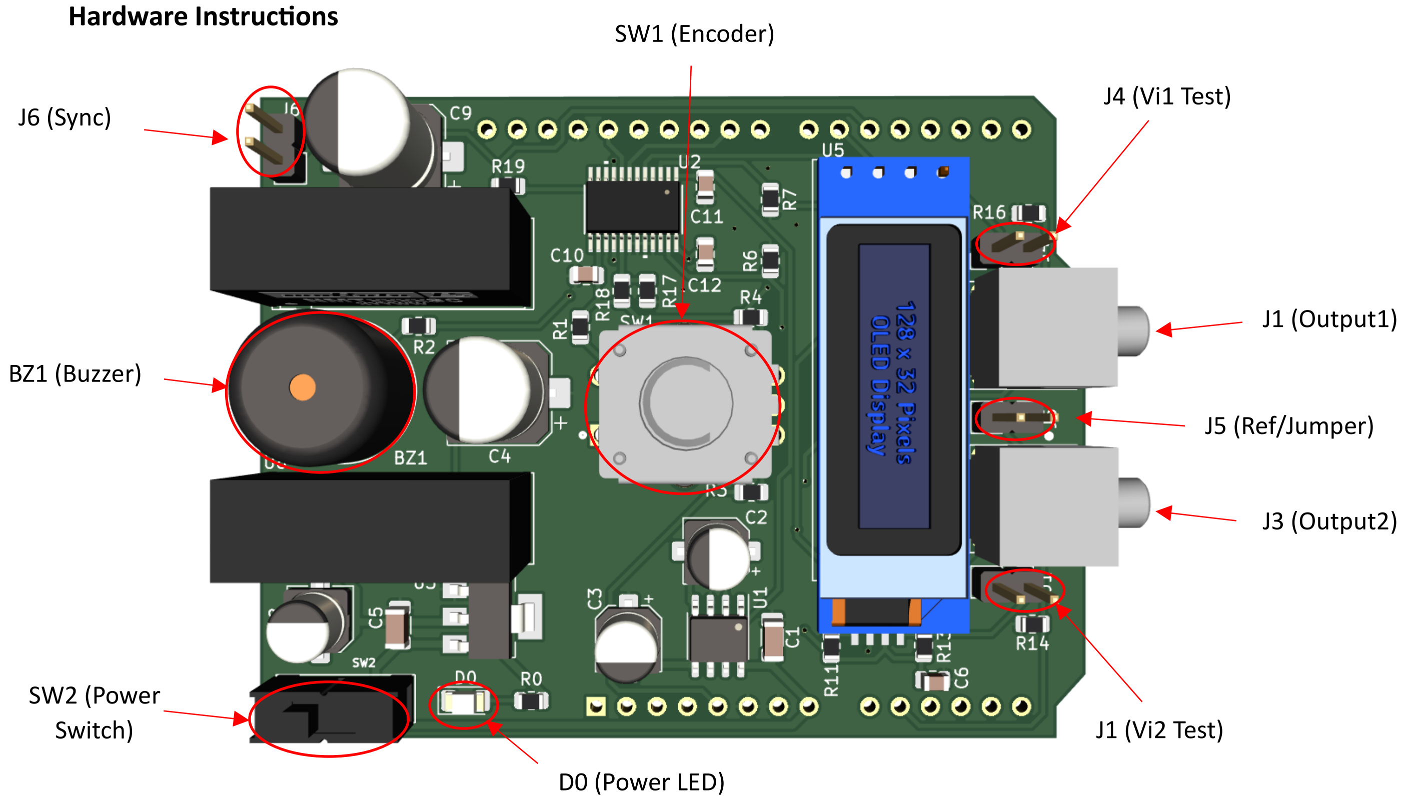
Warnings

This device is intended for EDUCATION AND RESEARCH PURPOSES ONLY. It is not approved by any regulatory authority for medical or other uses. The user assumes all responsibility for injury or damage due to the misuse of this device.

Design



Hardware Instructions



J6 (Trigger): 2-pin connector to trigger the stimulator from external TTL pulse. The trigger pin is connected to Arduino Uno's GPIO port D5 (pin 20). One must write an associative code to enable the D5 port pin as an input mode to allow the trigger to Sync the stimulation output. Please ignore this port pin if you are not using external trigger.

BZ1 (Buzzer): Built-in active buzzer connected to Arduino Uno's GPIO port D8 (pin 23) though a resistor. A logic HIGH to this pin will generate a sound. One should write an associative code to enable the D8 port pin as an output mode to allow to generate the sound.

SW2 (Power): Sliding switch to power the ±HV circuit. It draws power through Arduino Uno's VIN pin. Please connect a 7-12 volts DC power supply (preferably 9V 650 mAh battery) to the barrel plug (not shown in the picture). To save power, turn ON the SW2 switch (slide outward of the board) only prior to stimulation. The adjacent red LED (D0) glows when the switch is ON.

J4-J1 (Test points): These points provide the voltage operation of the stimulator. To examine the operation of the stimulator, connect an oscilloscope probe to these connectors, positive to the right (pin 1) and ground to left (pin 2).

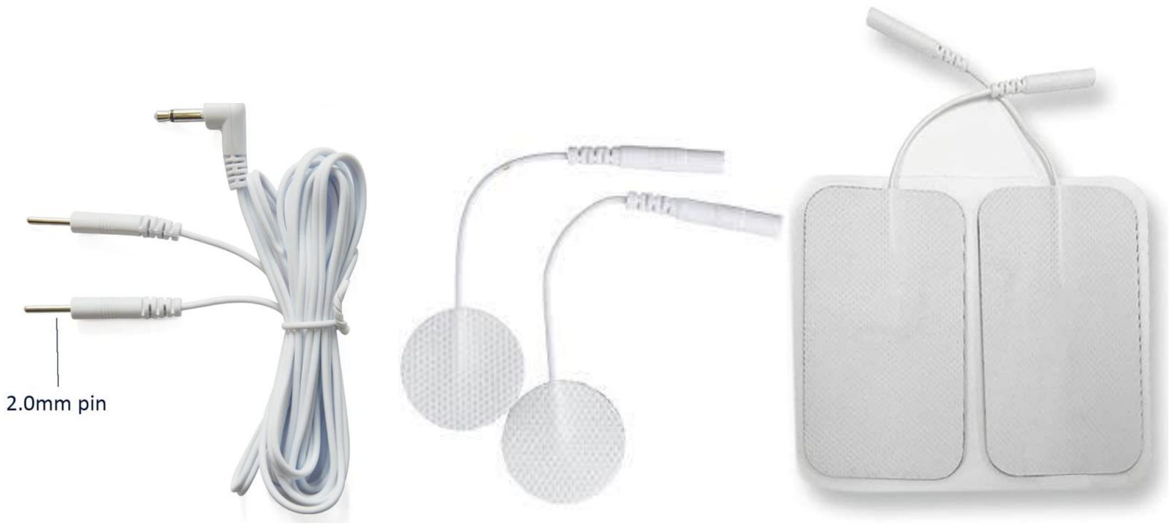
SW1 (Encoder): Rotate this to increase and decease the stimulation intensity. The Encoder also incorporates a push switch to allow additional programming option such as selecting or resetting values. The pulled-up switch is connected to the microcontroller's GPIO pin D3 (pin 18).

D0 (LED): The LED allows additional programming option such as blinking during stimulation to allow visual status of the stimulator operation. The LED is connected to the microcontroller's GPIO port D12 (pin 27) through a 270Ω resistor.

J5 (Ref/Jumper): This is a dual-purpose connector. The two pins of this connector is equipped with two 10Ω current sense resistors from the output terminals. The voltage drop across the resistor are fed to the analogue pins A0 & A1 (pin 9 & 10) of the Arduino uno microcontroller. An appropriate code should be written to allow the measurement of the sensed current through this ADC pins. Please note that the ADC can only read the positive part of the stimulation pulse. Alternatively, Once can connect an oscilloscope to this connector pins to measure the voltage drop across the current sense resistor to calculate the current value though the load. The 2nd operation of this connector is to utilize a jumper to allow a common reference point for the two-channel stimulation. This is particularly useful for brain or spinal cord stimulation where stimulation is provided across a common reference point.

J2-J3 (Outputs): These are 3.5 mm mono sockets to connect conventional TENS or EMS cables (see the following figure) for two channel stimulation output.

Assembly



Example image (for reference only): Typical TENS/EMS cable of 2.5 mm mono connector and pairs of TENS/EMS electrodes with 2.0 mm female connector.

[Please use high power (> 650 mAh) battery to have better operation of OpenXstim]

APPENDIX 1: Arduino uno example open-source code for OpenXstim2 v1.0

```
/***********************************************************************
// 16 June 2023, Australia
// Example code for setting up stimulation current for OpenXatim2 v1.0
// For queries please email Dr Monzurul Alam at md.malam@connect.polyu.hk
***********************************************************************/

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <stdint.h>
#include <AD57.h>

// set your stimulation parameters
#define Hz 20      // stimulation frequency
#define pulse_number 10   // number of pulses in a burst
#define pulse_width 50  // pulse width in microsecond
int current = 0;  // declare initial current intensity
int calibration = 4;  // set intensity calibration value

/*****************************************************
 * user variables
 *****************************************************/
// MUST : select the kind of AD57xx you have:
// AD5722 or AD5732 or AD5752
uint8_t DacType = AD5722;

// MUST : select the DAC channel to use (DAC_A, DAC_B or DAC_AB)
#define DAC_CHANNEL DAC_AB

/* MUST : define the output voltage range (for Bipolar use pnxxx)
 * p5V         +5V
 * p10V        +10V
 * p108V       +10.8V
 * pn5V        +/-5V
 * pn10V       +/-10V
 * pn108V      +/-10.8V
 */
#define output_range pn5V

// MUST : DEFINE THE SYNC PIN CONNECTION FOR AD57xx
/* The initialisation and pulling high and low is handled by the
 * library. The sketch only needs to define the right pin.
 */
char slave_pin_DAC = 9;

                                        // Define the SS1306 display connected to I2C pins
```

```
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels
#define LED_BLINK 12 // LED to blink during stimulation
// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

// Define rotary encoder pins
#define ENC_A 3
#define ENC_B 2

int16_t initial_value;
int16_t baseline = 2048;
int16_t value = baseline;
unsigned int Intensity = 0; // stimulation intensity
unsigned long _lastIncReadTime = micros();
unsigned long _lastDecReadTime = micros();
int _pauseLength = 2500;
int _fastIncrement = 1;
char buffer[64];  // buffer must be big enough to hold all the message

void setup() {
  pinMode(LED_BLINK, OUTPUT);
  // Set encoder pins and attach interrupts
  pinMode(ENC_A, INPUT_PULLUP);
  pinMode(ENC_B, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(ENC_A), read_encoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(ENC_B), read_encoder, CHANGE);
  // Start the serial monitor to show output
  Serial.begin(115200);
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3D for 128x64
    Serial.println(F("SSD1306 allocation failed"));
    for(;;);
  }
  //serialTrigger(F("Press any key to begin +5V to -5V  Bi-polar constant loop
output range mode"));
  /* enable or disable debug information from the library
   *  0 = disable (default.. no need to call)
   *  1 = enable
   */
  AD57.debug(0);
  Serial.println(F("Starting DAC"));
  // initialise the AD57xx and connections to and
  // from the AD57xx
  init_AD57xx();
  // set max count depending on AD57xx : 12, 14 or 16 bits
  if (DacType == AD5752)    initial_value = 0xffff;
  else if (DacType == AD5732) initial_value = 0x3fff;
  else if (DacType == AD5722) initial_value = 0x0fff;
}

//**Main program**//
void loop() {
  value = baseline + Intensity;
  // generate stimulation pulses
  for (int i=0; i<pulse_number; i++) {
    // positive phase
    AD57.setDac(DAC_CHANNEL, value);
    delayMicroseconds(pulse_width);
    // negetive phase
    AD57.setDac(DAC_CHANNEL, -value);
    delayMicroseconds(pulse_width);
  }
  // no pulses
  AD57.setDac(DAC_CHANNEL, baseline);
  delayMicroseconds(pulse_width);
  delay(1000/Hz); // stimulation delay
  //digitalWrite(LED_BLINK, !digitalRead(LED_BLINK)); // toggle LED

  // print the value to serial display
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(WHITE);
  display.setCursor(0, 0);
  // Calcluate and display set current intesity value
  current = Intensity / calibration;
  sprintf(buffer, "Current = %d mA", current);
  display.println(buffer);
  display.display();
  //Intensity = Intensity + 1;   // for debugging only
}

//**Interrupt service routine**//
void read_encoder() {
  // Encoder interrupt routine for both pins. Updates Intensity
  // if they are valid and have rotated a full indent

  static uint8_t old_AB = 3;  // Lookup table index
  static int8_t encval = 0;   // Encoder value
  static const int8_t enc_states[]  = {0,-1,1,0,1,0,0,-1,-1,0,0,1,0,1,-1,0};
// Lookup table

  old_AB <<=2;  // Remember previous state

  if (digitalRead(ENC_A)) old_AB |= 0x02; // Add current state of pin A
  if (digitalRead(ENC_B)) old_AB |= 0x01; // Add current state of pin B

  encval += enc_states[( old_AB & 0x0f )];

  // Update Intensity if encoder has rotated a full indent, that is at least 4
steps
  if( encval > 3 ) {        // Four steps forward
    int changevalue = 5;
    if((micros() - _lastIncReadTime) < _pauseLength) {
      changevalue = _fastIncrement * changevalue;
    }
    _lastIncReadTime = micros();
    Intensity = Intensity + changevalue;              // Update Intensity
    encval = 0;
  }
  else if( encval < -3 ) {  // Four steps backward
    int changevalue = -5;
    if((micros() - _lastDecReadTime) < _pauseLength) {
      changevalue = _fastIncrement * changevalue;
    }
    _lastDecReadTime = micros();
    if (Intensity > 0){
      Intensity = Intensity + changevalue;              // Update Intensity
    }
    else{
      Intensity = 0;
    }
    encval = 0;
  }
}

/* initialize the connection with the board  and set the default values */
void init_AD57xx()
{
  // set output voltage range in binary
  // if you want to test 2scomp, see remark 1 in AD57.h, use
  // AD57.begin(DAC_CHANNEL, output_range, COMP2);
  AD57.begin(DAC_CHANNEL, output_range);

  Serial.print("get range ");
  Serial.println(AD57.getRange(DAC_CHANNEL));

  // enable therminal shutdown and overcurrent protection
  // also stop CLR as, starting from 0v after OP_CLEAR op_code)
  // this can be changed for full negative after clear with
  // AD57.setControl(OP_INSTR,(SET_TSD_ENABLE |SET_CLAMP_ENABLE|SET_CLR_SET));
  AD57.setControl(OP_INSTR,(SET_TSD_ENABLE |SET_CLAMP_ENABLE|STOP_CLR_SET));
  // clear the DAC outputs
  AD57.setControl(OP_CLEAR);
}

bool check_status(bool disp)
{
    uint16_t stat = AD57.getStatus();
    bool ret = 0;

    if (stat & stat_err_TO)
    {
      Serial.println(F("Therminal overheat shutdown"));
      ret = 1;
    }

    if (stat & stat_err_CA)
    {
      Serial.println(F("DAC - A overcurrent detected"));
      ret = 1;
    }

    if (stat & stat_err_CB)
    {
      Serial.println(F("DAC - B overcurrent detected"));
      ret = 1;
    }

    if (disp == 1)
    {
      Serial.println(F("Display settings\n"));

      if (stat & stat_TS)
         Serial.println(F("Therminal shutdown protection enabled"));
      else
         Serial.println(F("Therminal shutdown protection disabled"));

      if (stat & stat_CLR)
        Serial.println(F("DAC output set to midscale or fully negative with
OP_CLEAR command"));
      else
        Serial.println(F("DAC output set to 0V with OP_CLEAR command"));

      if (stat & stat_CLAMP)
         Serial.println(F("Overcurrent protection enabled"));
      else
         Serial.println(F("Overcurrent protection disabled"));
      if (stat & stat_SDO) // you will never see this one :-)
         Serial.println(F("Serial data out disabled"));
      else
         Serial.println(F("Serial data out enabled"));

      Serial.println();
    }

    return(ret);
}

/* serialTrigger prints a message, then waits for something
 * to come in from the serial port.
 */
void serialTrigger(String message)
{
  Serial.println();
  Serial.println(message);
  Serial.println();

  while (!Serial.available());

  while (Serial.available())
    Serial.read();
}

/* errorLoop loops forever.*/
void errorLoop()
{
  // power-off both channels
  AD57.setPower(STOP_PUA|STOP_PUB);

  Serial.print(F("Critical error, looping forever"));
  for (;;);
}
```