Using Arduino to control SiPM Temperature Compensated HV Module

A7585D

Features

- 20-85V Output Voltage
- 10mA Output Current
- 1mV Output Voltage step
- Less than 300uV rms noise
- User Selectable Digital / Analog output voltage control
- Automatic temperature feedback on the output voltage
- Multi device support using I2C
- Arduino library designed to manage multiple devices
- Evaluation kit include
    - A7585DU HV with USB and I2C interface
    - Arudino Uno
    - Display with Keypad
    - Bread Board
    - Cables
    - 12v Power Supply
- Fully working examples designed to control the module with keypad and display or monitor multiple devices using serial port
- Module compatible with ZEUS software for stand alone usage

Description

The NIPM-12 SiPM Power Module is a compact and integrated solution to provide stable and noiseless power supply for single and array / matrix SiPM detectors.

High resolution Output Voltage and Output Current measurements enable the NIPM-12 to be used for I-V detector characterization.

Digital (UART, I2C and USB with adapter) and analog control interface are runtime selectable by a single pin or a digital command.

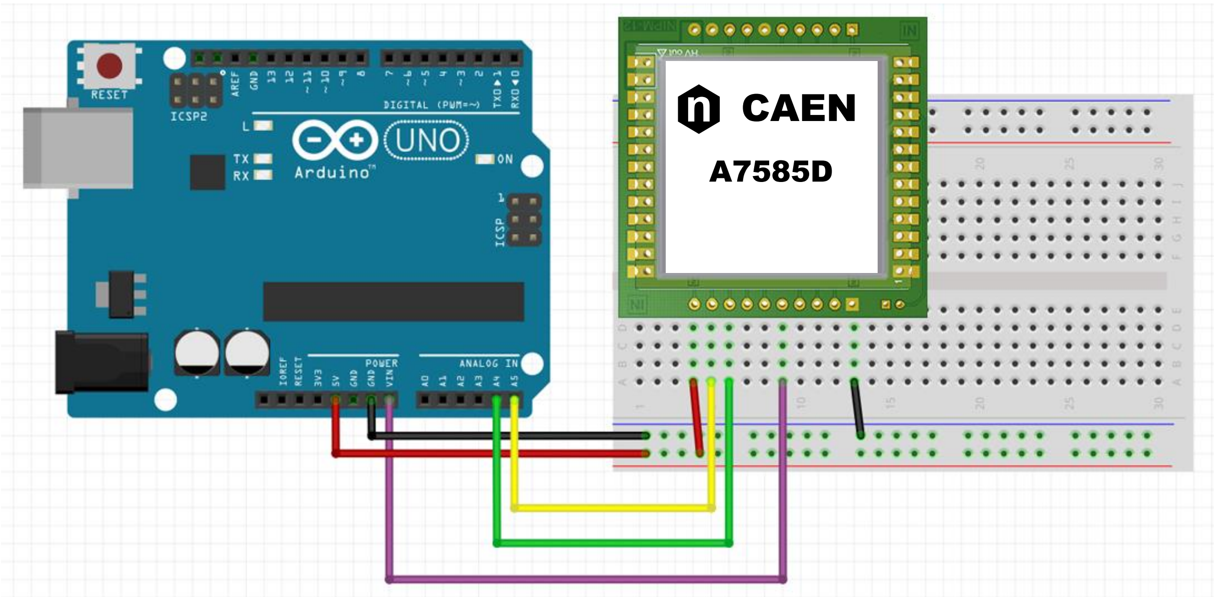
The module integrates a temperature HV loop that regulates the SiPM output voltage as a programmable function of the SiPM temperature coefficient.

The Arduino evaluation kit is a ready to use solution to discovery the capabilities of the A7585D integrated in a custom OEM solution. The A7585D library provide an high level abstraction of all commands supported by the module. It allows, for example, with a few lines of code to configure the module and readback voltage and current monitor values

ARDUINO

Designed in collaboration with: Nuclear Instruments

HARDWARE CONNECTIONS

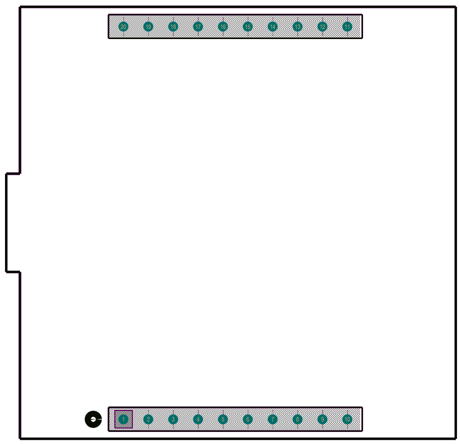


The A7585D can be controlled via UART, USB and I2C. While UART and USB are suited for PC control the I2C is the best solution to interface multiple modules with module with a single controller like an Arduino, a Raspberry PI or another microcontroller. I2C bus support device addressing: up to 120 A7585D/DU can be controlled by a single Arduino processor without any communication issue.
In application where a large number of channel is required, this design can be the ultimate solution in order to control and monitor multiple devices.
The I2C bus use just two wires to interface a master device with multiple slave. The master device is the Arduino Uno while the slave devices are the A7585DU modules.
The two wires are:

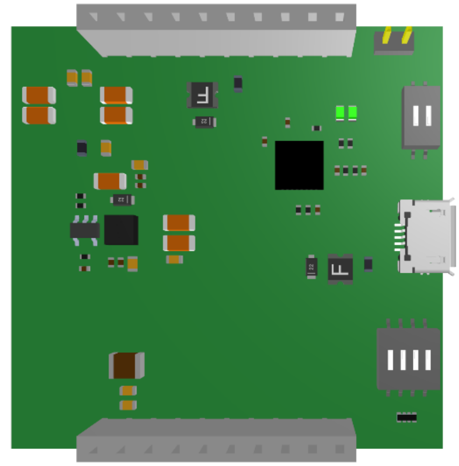
- SDA: data wire. Bidirectional, transport data between master and slaves. SDA is the green wire in the scheme
- SCL: clock wire. Generated by the master. SCL is the yellow wire in the scheme

Each device on the I2C has an address that must be unique. Read carefully the module address section if you need to operate with multiple A7585DU
The A7585DU must operate with a supply voltage greater than 6V. It is indeed impossible to connect the Vin pin to the Arduino 5V. An external power supply is required. Power Supply can be externally provided with the 12v power supply included in the kit through the Arduino power supply connector.
The 12V wire (violet) connect the VIN pin of the Arduino to the input voltage of the A7585DU
The A7585DU require an additional power supply of 5V (red) in order to power the I/O buffer for the I2C interface. This buffer allows to interface the A7585DU on the I2C with any chip operating with a voltage between 1.8V and 5.5V.
The black wire connects the Arduino ground to the HV module ground.
This is the minimal configuration to control with I2C the A7585DU module

» PIN FUNCTION



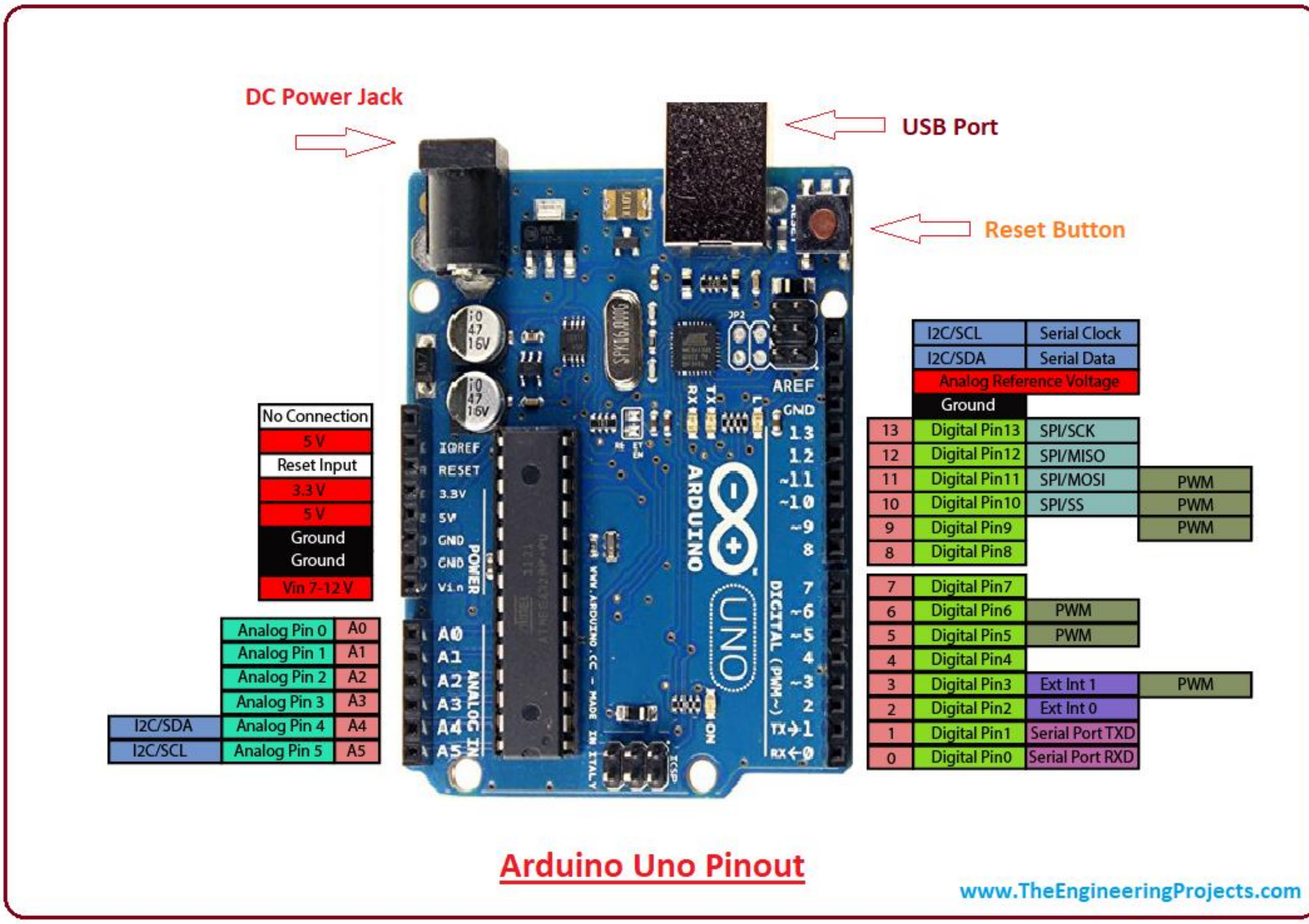
Top view



Bottom view

N	NAME	TYPE	DESCRIPTION
1	ANALOG IN	IN	Analog Control Mode: Reference voltage to regulate output voltage Digital Control Mode: Thermometer input
2	I2C ADDRESS 0	IN	I2C Address bit 1. Internally pull-up
3	I2C ADDRESS 1	IN	I2C Address bit 0. Internally pull-up
4	POWER CONTROL	IN	Output enable pin: <u>Analog Control Mode</u>: when 1 HV output is on, when 0 HV output is off <u>Digital Control Mode</u>: On power on, when 1 HV output is on, when 0 HV output is off during operation: 0—>1 transition HV output is enabled, 1—>0 transition HV output is disabled
5	MODE SELECT	IN	Select control mode (pin must be stable on power on): 0 Digital, 1 Analog Control
6	GND	POWER	
7	IMON	OUT	Analog I monitor output. Proportional to the output current in the range 0÷5V
8	VMON	OUT	Analog V monitor output. Proportional to the output voltage in the range 0÷5V
9	GND	POWER	
10	HV OUT	HV	HV Output
11	I2C VDD	POWER	Input pin to power the I2C voltage translator. 1.8V to 5V
12	I2C SCL	IN	Serial Clock pin of the I2C slave bus
13	I2C SDA	INOUT	Serial Data pin of I2C slave bus.
14	UART TX	OUT	UART TX pin. 115200 bps,8,1,n. 5V ONLY
15	UART RX	IN	UART RX pin. 115200 bps,8,1,n. 5V ONLY
16	POWER SUPPLY	POWER	USB disconnected: power supply USB connected: Vin > 6V module powered from Vin, otherwise USB powered.
17	TEMPERATURE SENSOR CONNECTED	OUT	When high the temperature measurement is enabled
18	OVER-CURRENT	OUT	Indicates the presence of HV power on output pin When logic 1 the HV output is enabled, when 0 the HV is off.
19	OUTPUT STATUS	OUT	Indicates the presence of HV power on output pin When logic 1 the HV output is enabled, when 0 the HV is off.
20	GND	POWER	
21	USB		USB connection

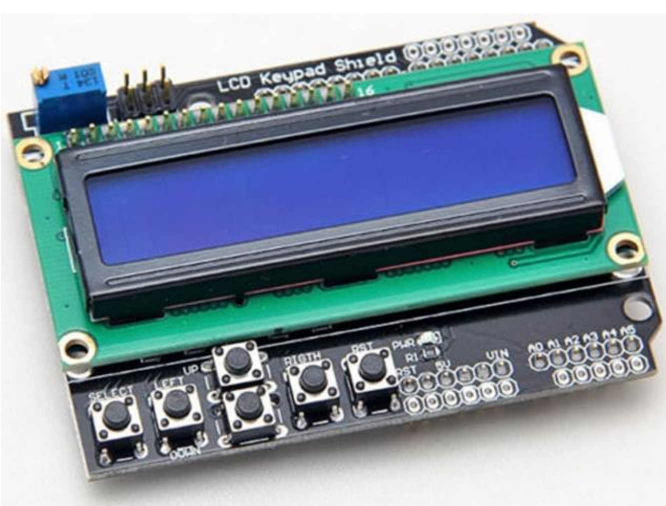
» ARDUINO UNO PINOUT



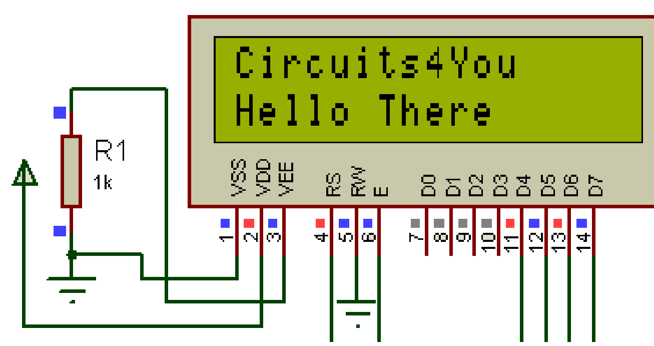
Arduino Uno Pinout

www.TheEngineeringProjects.com

» DISPLAY SHIELD

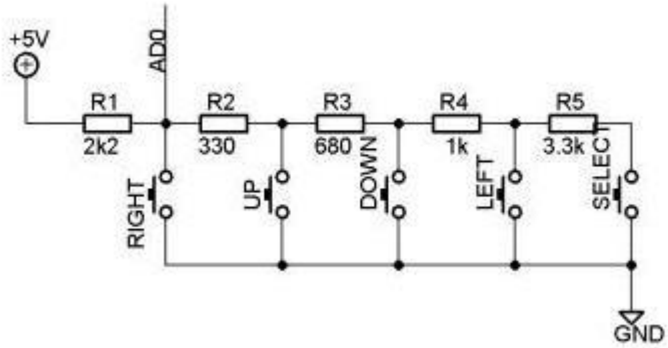


The kit include a display shield with keypad. The shield has a 16x2 LCD screen with blue backlight and 5 key buttons.
The LCD display is connected in 4 wires mode to the Arduino.
The LiquidDisplays library is used in order to simplify the interfacing to the display with very basic commands like goto, print string.



4 wire are used for data transfer (D4...D7) and are connected to pin 4,5,6,7 while Rs is connected to 8 and Enable is connected to 9

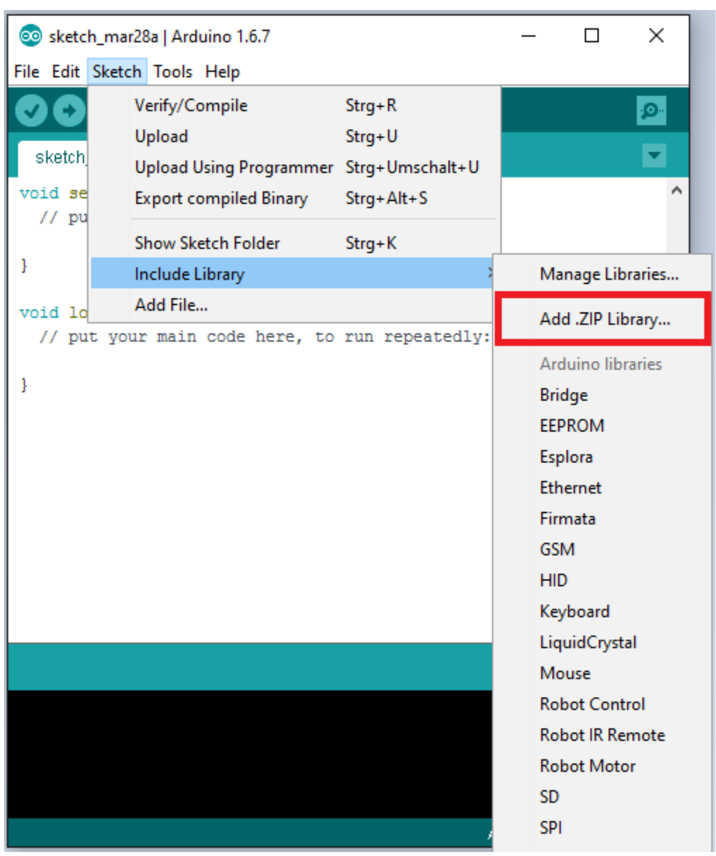
» DISPLAY SHIELD



In order to reduce the number of required pin buttons are connected to a voltage divider. Pressing a button the AD0 voltage changes.
Reading from A0 pin and comparing the voltage to a lookup table it is possible to understand the pressed button

» INSTALL LIBRARY

In order to use the examples of this application note it is necessary to install A7585D library and LiquidCrystals library.
A7585D library is a Nuclear Instruments library developed to interface with A7585D with an high level class while LiquidCrystals library is an opensource library for LCD display control.
To install the libraries open Arduino software, Sketch menu -> Add .ZIP library.
Select the two library zip file from the repository



» A7585 LIBRARY

In the following section a full description of all function of the A7585A library is provided

```
bool Init(int IICAddress)
        IN
        IICAddress: address of the A7585A in 7 bit format
        OUT
        bool: true if a valid module has been identified
```

Initialize the library providing the I2C module Address. The default module address is 0x70

```
void Set_MaxV(float v)
        IN
        v: HV output compliance voltage
```

Set the hv output maximum output voltage (protection)

```
void Set_MaxI(float v)
        IN
        v: HV maximum output currect
```

Set the hv output maximum output current. The module is switched off if current exceeds the limit

----

```
void Set_Enable(bool v)
        IN
        v: Enable the output
```

Enable the HV output. If the HV fails due to overcurrent Set_Enable false and true to restore HV

```
void Set_RampVs(float v)
        IN
        v: Ramp speed in V/s
```

Set the HV output ramp speed in V/S

```
void Set_TemperatureSensor(HVTemperatureSensors SensorModel, float term_m2,
float term_m, float term_q )
        IN
        SensorModel: Select between standard thermometer already calibrated and
        custom model
        term_m2: quadratic fitting parameter between temperature and ref voltage
        term_m: linear fitting parameter between temperature and ref voltage
        term_q: offset
```

Configure the thermometer connected to the A7585A in order to operate in temperature compensated mode

----

```
void Set_Filter(float alfa_v, float alfa_i, float alfa_t)
```

IN

```
alfa_v: out voltage monitor filter coefficient
alfa_i: out current monitor filter coefficient
alfa_t: temperature monitor filter coefficient
```

Set the filter coefficient applied to the data monitor filter [0…0.9999]. A value closer to 1 means an higher number of averages and an higher resolution while a value closer to 0 means no filtering and faster response

----

```
void Set_SiPM_Tcoef (float tcomp)
```

IN

```
tcomp: Thermal coefficient compensation
```

This is the coefficient provided by SiPM manufacturer expressed is V/°C to compensate the gain variation due to the temperature. A typically value is -34mV/°C

```
void EmergencyOff()
```

Switch off HV without ramp

```
void SetI0()
```

Compensate the current measured zeroing the measurement and removing biasing

----

```
void Set_DigitalFeedback(bool on)
```

IN

```
on: Enable the internal PID controller
```

The internal PID controller use the voltage value read by the feedback ADC in order to compensate small static error in the feedforward output setpoint

----

```
void Set_IIC_baseaddress(uint8_t ba)
```

IN

```
ba: new base address for the A7585D
```

Change the base address of the A7585D. The module address will be the ba added to the status of the address pins.

```
uint8_t GetDigitalPinStatus()
```

OUT

```
uint8_t: binary encoded pin status
```

Read the status of all digital I/O of the module

```
float GetVin()
```
OUT
`float: power supply voltage`

Read the power supply voltage

```
float GetVout()
```
OUT
`float: Output voltage`

Read the power HV out voltage

```
float GetIout()
```
OUT
`float: Output current`

Read the power HV output current

```
float GetVref()
```
OUT
`float: Reference pin voltage`

Read the voltage on the reference (analog control only) pin

```
float GetTref()
```
OUT
`float: Temperature of the SiPM`

Read the temperature of the SiPM through the temperature sensor connected to the Vref pin

----

```
float GetVtarget()
```
OUT
`float: Current voltage target`

Read current output voltage target

```
float GetVtargetSP()
```
OUT
`float: Current output voltage set on the module`

Read the current DAC set point in Volt

```
float GetVcorrection()
```
OUT
```
float: correction voltage applied to the target
```

Read the correction voltage applied to the target in order to compensate the temperature

---

```
bool GetVCompliance()
```
OUT
```
bool: flag indicating that the module output voltage is limited by
compliance
```

When true, the output voltage is clamped to the compliance voltage

---

```
bool GetICompliance()
```
OUT
```
bool: flag indicating a overcurrent protection
```

When true, the module has been switched off due to over current. To restore the module disable and enable the output.

```
uint8_t GetProductCode()
```
OUT
```
uint8_t: product code of the device
```

Return the product code of the device

```
uint8_t GetFWVer()
```
OUT
```
uint8_t: firmware version
```

Return the firmware version of the device

```
uint8_t GetHWVer()
```
OUT
```
uint8_t : hardware version
```

Return the hardware version of the device

```
uint32_t GetSerialNumber()
```
OUT
```
uint32_t: serial number of the device
```

Return the serial number of the device

```
bool GetHVOn()
```

OUT

`bool: hv status`

Read the current HV status (true is on)

```
bool GetConnectionStatus()
```

OUT

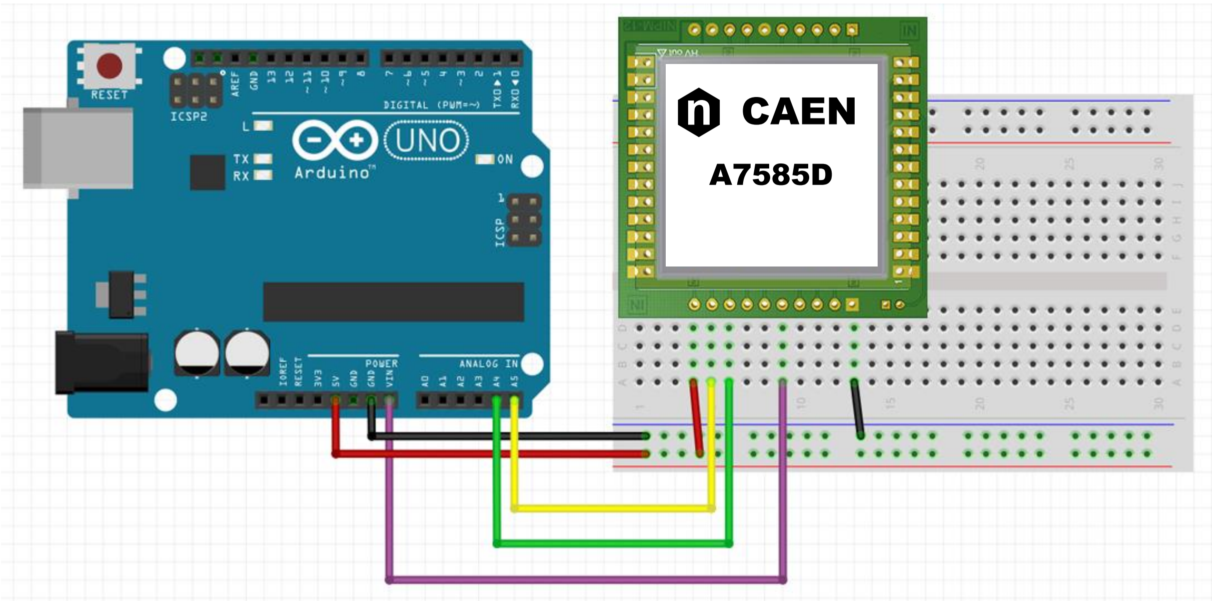
`bool: flag indicating that Arduino is connected to the module`

When true, the Arduino is connected to the module

```
void StoreCfgOnFlash ()
```

Save configuration on flash

» CONTROL A SINGLE MODULE



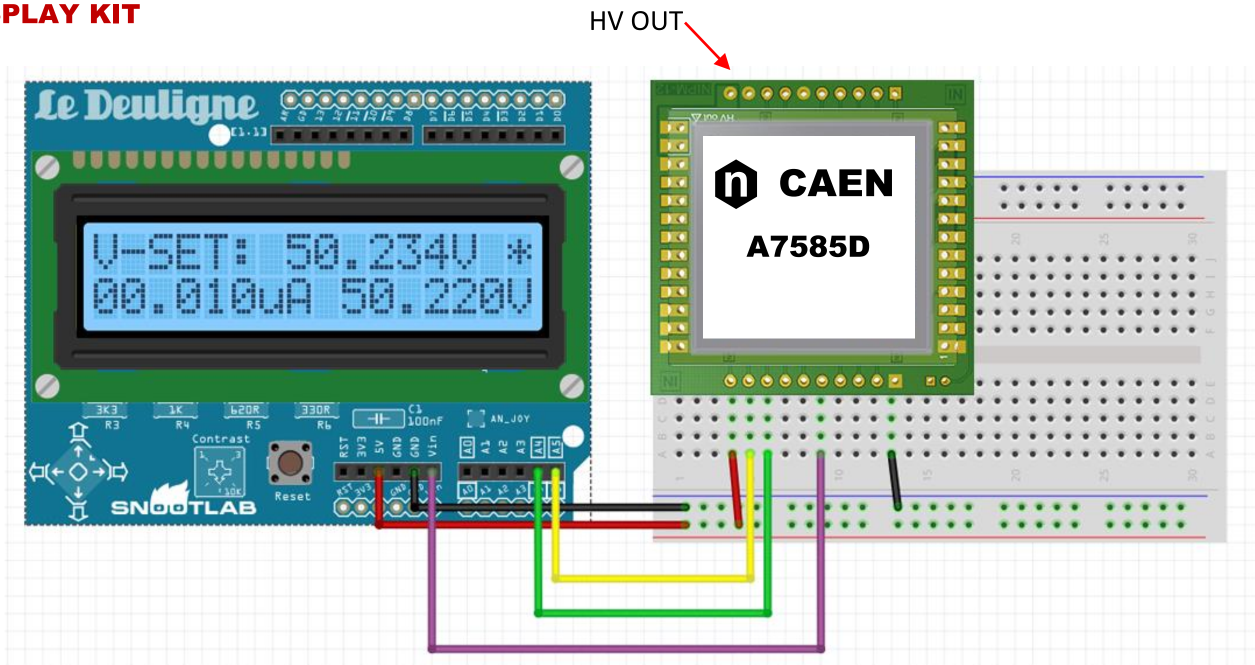
Connect the module A7585DU included in the kit to the Arduino as explained in the section hardware connection and load the TestA7585 sketch. This demo will init the A7585D module and generate a ramp between 25V and 80V increasing/decreasing the output voltage of 1V every second.
The demonstrate the compliance the output setpoint swings between 20 and 85V but the compliance is set to 80V limiting the full swing

```
#include <A7585lib.h>
#include <Wire.h>
#include <stdio.h>
#include <stdlib.h>

#define DEV_ADDRESS 0x70

A7585 A7585_dev;
double current_voltage;
double add_step = 1;
void InitNIPM(int device_address)
{
  A7585_dev.Init(device_address);
  A7585_dev.Set_Mode(0);
  A7585_dev.Set_MaxV(80);
  A7585_dev.Set_MaxI(10);
  A7585_dev.Set_RampVs(25);
  A7585_dev.Set_V(50);
}
void setup() {
  Serial.begin(115200);
  Wire.begin();
  Serial.println("Starting A7585 HV demo app. This app will ramp the HV");
```

» CONTROL A SINGLE MODULE

```
InitNIPM(DEV_ADDRESS);
  if (A7585_dev.GetConnectionStatus())
    Serial.println("Probe connection successful");
  else
    Serial.println("Error connecting device ");

  A7585_dev.Set_V(50);
  current_voltage=25;
  A7585_dev.Set_Enable(true);


}

void loop() {
  float cvout, ciout;
  if (current_voltage>85) {add_step=-1;}
  if (current_voltage<25) {add_step=1;}
  current_voltage += add_step;

  A7585_dev.Set_V(current_voltage);
  cvout = A7585_dev.GetVout();
  ciout = A7585_dev.GetIout();

  Serial.print(" V: "); Serial.print(cvout); Serial.print(" I: ");
Serial.println(ciout);
  delay(1000);
}
```

Download the firmware and open the Arduino serial monitor (Tool -> Serial Monitor), set speed to 115200 and you will se the readback of the voltage set point



Connect the keypad/display shield to the Arduino as in picture above. The shield will have a series of pass-through connection where you can connect the module A7585DU included in the kit.
Display will not use the I2C pin and they will be available to communicate with the module.
Use the up/down key to set output voltage and right key to enable/disable HV.

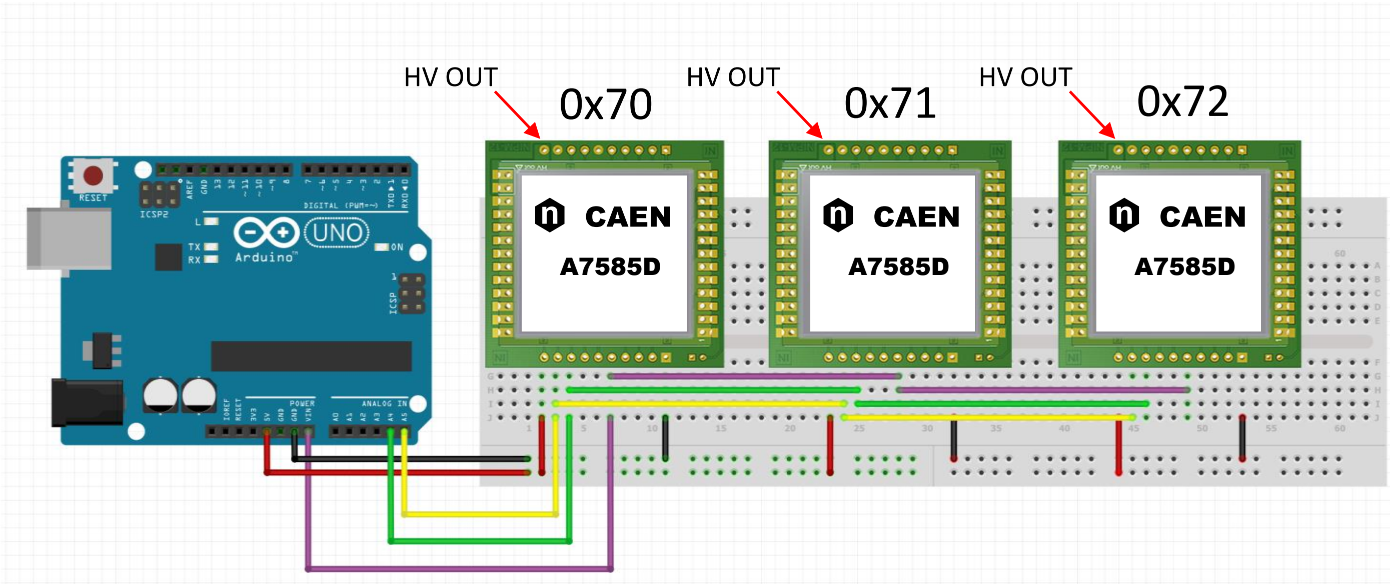
```
#include <LiquidCrystal.h>
#include <A7585lib.h>

#include <Wire.h>
#include <stdio.h>
#include <stdlib.h>

#define DEV_ADDRESS 0x70

LiquidCrystal lcd(8, 9, 4, 5, 6, 7);      // select the pins used on the LCD panel

int lcd_key     = 0;
int adc_key_in  = 0;

#define btnRIGHT  0
#define btnUP     1
#define btnDOWN   2
#define btnLEFT   3
#define btnSELECT 4
#define btnNONE   5
float vset=50;           // current set point
bool on_off=false;       // current status
uint32_t  tdow;          // ms key is pressed
uint32_t tl=0;           // ms from last update
bool upd=false,dwd=false;
float inc=0.01;          // current speed
A7585 A7585_dev;         // device A7585
```

```
void setup() {
  lcd.begin(16, 2);              // start the library
  InitNIPM(DEV_ADDRESS);

  A7585_dev.Set_V(50);
  A7585_dev.Set_Enable(false);


}

void loop() {



  uint32_t delta;
  char tmp[16];
  float volt1, curl;

   lcd.setCursor(0,0);              // LCD goto 0,0
   lcd.print("V-SET:");             // print V-SET
   lcd.setCursor(6,0);              // LCD goto 6,0
// prepare voltage string
   sprintf(tmp, "%d.%02d", (int)vset, (int)(vset*100)%100);
   lcd.print(tmp);                   // print voltage Set Point String
   lcd.setCursor(11,0);
   lcd.print("V");

   lcd.setCursor(15,0);
   lcd.print(on_off?"*":"-");        // print * if module is on or - if is off

// read current and voltage monitor from the module
   curl = A7585_dev.GetIout();
   volt1 = A7585_dev.GetVout();

// print current and voltage  monitor on the display
   lcd.setCursor(8,1);
   sprintf(tmp, "%2d.%03duA", (int)curl,(int32_t) (curl*1000)%1000);
   lcd.print(tmp);

   lcd.setCursor(0,1);
   sprintf(tmp, "%2d.%03dV ", (int)volt1,(int32_t) (volt1*1000)%1000);
   lcd.print(tmp);

 delta=millis()-tdow; // calculate key down time

   lcd_key = read_LCD_buttons();    // read the buttons

// depending on which button was pushed, we perform an action
   switch (lcd_key){

       case btnRIGHT:{                            //ON/OFF
            if (millis()-tl > inc)
            {
```

```
                on_off=!on_off;
                A7585_dev.Set_Enable(on_off);
                tl=millis();
              }
              break;
         }
         case btnLEFT:{
               break;
         }
         case btnUP:{
                                  //INCREASE VOLTAGE

              if (millis()-tl > inc)                      //KEY DOWN MANAGE
              {
               if (vset<80)
                vset += 0.01;
              //Add 10mV
                A7585_dev.Set_V(vset);                    //Set Voltage
               tl=millis();
              }
               break;
         }
         case btnDOWN:{
              //DECREASE VOLTAGE
              if (millis()-tl > inc)
              {
               if (vset>22)
                vset -= 0.01;
                A7585_dev.Set_V(vset);
                tl=millis();
              }
               break;
         }
         case btnSELECT:{


               break;
         }
         case btnNONE:{
               tdow = millis();
               tl=0;
               break;
         }
     }

     //Change increment speed
           if (delta<1000)
           inc=500;
         if (delta>1000)
           inc=150;
         if (delta>2500)
           inc=0;
  }
```

```
void InitNIPM(int device_address)
{
  A7585_dev.Init(device_address);
  A7585_dev.Set_Mode(0);
  A7585_dev.Set_MaxV(80);
  A7585_dev.Set_MaxI(10);
  A7585_dev.Set_RampVs(25);
  A7585_dev.Set_V(50);
}


int read_LCD_buttons(){                  // read the buttons
    adc_key_in = analogRead(0);          // read the value from the sensor

    // my buttons when read are centered at these valies: 0, 144, 329, 504, 741
    // we add approx 50 to those values and check to see if we are close
    // We make this the 1st option for speed reasons since it will be the most
likely result

    if (adc_key_in > 1000) return btnNONE;

    // For V1.1 us this threshold
    if (adc_key_in < 50)   return btnRIGHT;
    if (adc_key_in < 250)  return btnUP;
    if (adc_key_in < 450)  return btnDOWN;
    if (adc_key_in < 650)  return btnLEFT;
    if (adc_key_in < 850)  return btnSELECT;

   // For V1.0 comment the other threshold and use the one below:
   /*
     if (adc_key_in < 50)   return btnRIGHT;
     if (adc_key_in < 195)  return btnUP;
     if (adc_key_in < 380)  return btnDOWN;
     if (adc_key_in < 555)  return btnLEFT;
     if (adc_key_in < 790)  return btnSELECT;
   */

    return btnNONE;                    // when all others fail, return this.
}
```

» CONTROL MULTIPLE MODULE



The following example will control up to 16 module (can be easy extended to 127). In initialize all connected modules to generate 50V and enumerate in scanning the I2C bus. The output is printed on the serial port. Download the firmware MultiDevA7585 and open the Serial Monitor setting the speed 115200. The software will output every seconds a new line with the address of the module detected.

```
#include <A7585lib.h>


#include <Wire.h>
#include <stdio.h>
#include <stdlib.h>

#define DEV_ADDRESS_BASE 0x70

A7585 A7585_dev[16]; // Istantiate 16 independent class for the A7585D


void setup() {
  int i;

  Serial.begin(115200);  // Open serial port
  Wire.begin();
  Serial.println("Starting A7585 HV demo app. This app will ramp the HV");

  for (i=0;i<16;i++)
  {
    A7585_dev[i].Init(DEV_ADDRESS_BASE+i);
    A7585_dev[i].Set_Mode(0);
    A7585_dev[i].Set_MaxV(80);
    A7585_dev[i].Set_MaxI(10);
    A7585_dev[i].Set_RampVs(25);
    A7585_dev[i].Set_V(50);
```

```
    Serial.print("Device ID ");
    Serial.print(DEV_ADDRESS_BASE+i);
    if (A7585_dev[i].GetConnectionStatus())
      Serial.println(" connected");
    else
      Serial.println(" not connected");
    A7585_dev[i].Set_V(50);
    A7585_dev[i].Set_Enable(true);
  }


}

void loop() {

  int i;
  for (i=0;i<16;i++)
  {


    if (A7585_dev[i].GetConnectionStatus())
      {Serial.print( DEV_ADDRESS_BASE + i); Serial.print(' ');}
     else
      {Serial.print('-'); Serial.print(' ');}
  }
  Serial.println(' ');
  delay(1000);
}
```