Viareggio
June 11th, 2020

Abstract

CAEN **A7585D/DU SiPM Power Supply** (developed in collaboration with Nuclear Instruments S.r.L) is a compact and integrated solution to provide stable and noiseless bias for Silicon Photomultipliers (SiPM) arranged in single pixel, matrix or array [1].
In this Application Note we give a complete step-by-step guide to the usage of A7585DU controlled by Arduino UNO[2]. We describe the hardware setup, the A7585D/DU open-source library and we give practical examples of coding to control one or more modules. The provided open-source C library exposes all commands supported by the module at high-level abstraction and allows the user to easily perform configurations and readback of current and voltage values.

Introduction

The A7585D/DU Power Supply is meant to be used for **integration in SiPM readout systems** for detectors biasing purposes. The module features both analog and digital control (UART, I2C and optional USB) in order to ease the prototyping, testing and integration processes. For this reason, CAEN makes available a ready-to-use control software (ZEUS, refer to [1]) as well as an open-source C library with some coding examples to help users in embedding the A7585D/DU in their own systems.
In this Application Note we show how to use the provided library (available on GitHub) to control one or more A7585DU SiPM Power Supply. For this purpose, we suggest the usage of a low-cost Arduino UNO platform [2], a compatible Display with keypad like Deuligne by SNOOTLAB [3], a breadboard, compatible connection cables and a 12 V Power Supply wall adapter.

Hardware Setup

The A7585DU 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 a single controller like an Arduino [4], a Raspberry PI [5] or another suitable microcontroller. In this Application Note, we use an Arduino UNO platform [2].
I2C bus supports device addressing: up to 120 A7585D/DU can be controlled by a single processor without any communication issue.In applications where a large number of channels is required to bias many SiPMs, this design can be the ultimate solution in order to control and monitor multiple devices.
The I2C bus uses just two wires to interface a master device to multiple slaves. In this case, the master device is the Arduino UNO controller, while the slave devices are the A7585DU modules. The two wires are:

- **SDA**: bidirectional data transmission between master and slaves (green wire in Fig.1)
- **SCL**: clock signal generated by the master (yellow wire in Fig.1)

Each device on the I2C has an address that must be unique. Carefully read Sec. if you need to operate with multiple A7585DU.
The A7585DU must operate with a supply voltage greater than 6V. Therefore, it is not possible to connect the *Vin* pin of the A7585DU to the Arduino 5V. Indeed, an external power supply is required, for example connecting a standard 12V power supply to the Arduino UNO dedicated connector. The 12V wire (violet wire in Fig.1) connects the *VIN* pin of the Arduino to the input voltage of the A7585DU.

Moreover, the A7585DU requires an additional 5V power supply (red wire in Fig.1) in order to power the I/O buffer for the I2C interface. This buffer allows the user to interface the A7585DU on the I2C with any chip operating with a voltage between 1.8V and 5.5V.
Finally, the black wire in Fig.1 connects the Arduino UNO ground to the A7585DU ground. This is the minimal configuration to control the A7585DU module through the I2C bus.

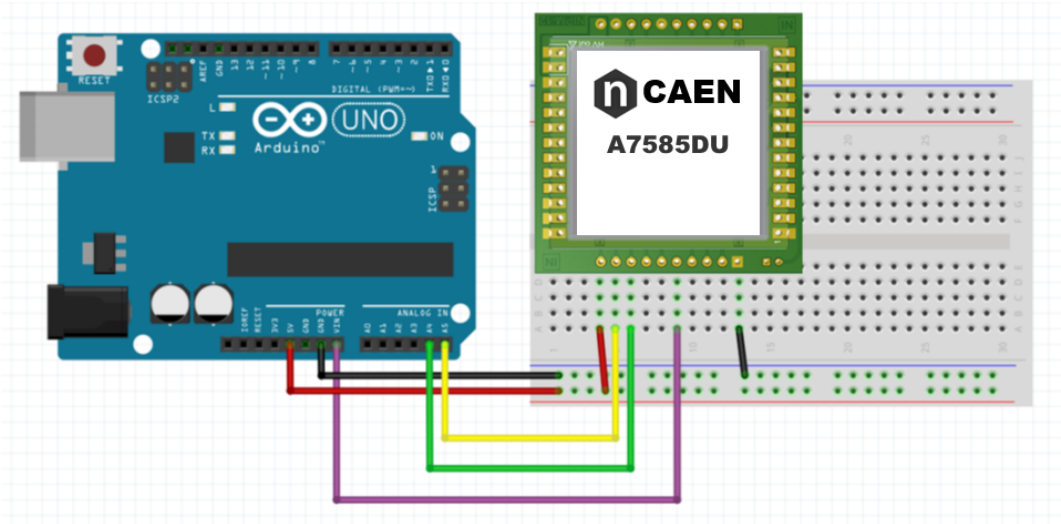

Fig. 1: Needed hardware connections between Arduino UNO and one A7585DU Power Supply plugged onto a breadboard. Refer to [1] for the A7585DU pinout.

In our library and examples we included the code to set a display shield with keypad. We used a Deuligne LCD [3]. More details are given in Sec.. The *LiquidCrystals* library is used in order to simplify the interfacing to the display with very basic commands like *goto*, *print string*, etc.

Fig. 2: The display used for this Application Note.

Libraries Installation and Description

In order to go through the examples of this Application Note, it is needed to install the A7585D/DU library and *LiquidCrystals* library. A7585D/DU library is a open-source library provided by CAEN to interface with A7585D/DU Power Supply

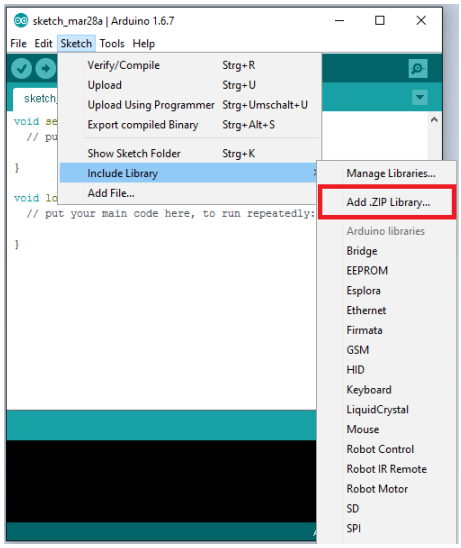

Fig. 3: How to add the needed libraries to Arduino software.

using a high level class, while *LiquidCrystals* library is an open-source library for LCD display control.
In order to install the libraries, open Arduino UNO software and surf into Sketch→ Include Library→ Add .ZIP library, as shown in Fig.3. Then, select the two libraries zip file available at the GitHub repository.

In the following we give a full description of all functions of the A7585D/DU library:

Initialize the library providing I2C module address. Default address is 0x70

```
bool Init(int IICAddress)
```

Set the HV output maximum compliance voltage (v)

```
void Set_MaxV(float v)
```

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

```
void Set_MaxI(float v)
```

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

```
void Set_Enable (bool v)
```

Set the HV output ramp speed in V/s

```
void Set_RampVs(float v)
```

Configure the thermometer connected to the module in order to operate in temperature-compensated mode

```
void Set_TemperatureSensor(HVTemperatureSensors SensorModel, float term_m2, float term_m, float
    term_q)
/*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
*/
```

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

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

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

Set 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 Set_SiPM_Tcoef (float tcomp)
```

Emergency switch off withouth HV ramp

```
void EmergencyOff()
```

Compensation of the current measured zeroing the measurement and removing biasing

```
void SetIO()
```

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_DigitalFeedback(bool on)
```

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

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

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

```
uint8_t GetDigitalPinStatus()
```

Read the power supply voltage

```
float GetVin()
```

Read the HV output

```
float GetVout()
```

Read the output current

```
float GetIout()
```

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

```
float GetVref()
```

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

```
float GetTref()
```

Read output voltage target

```
float GetVtarget()
```

Read the DAC set point in Volt

```
float GetVtargetSP()
```

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

```
float GetVcorrection()
```

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

```
bool GetVCompliance()
```

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

```
bool GetICompliance()
```

Return the product code of the device

```
uint8_t GetProductCode()
```

Return the firmware version of the device

```
uint8_t GetFWVer()
```

Return the hardware version of the device

```
uint8_t GetHWVer()
```

Return the serial number of the device

```
uint32_t GetSerialNumber()
```

Read the current HV status (true is on)

```
bool GetHVOn()
```

When true, the Arduino is connected to the module

```
bool GetConnectionStatus()
```

Save configuration on flash

```
void StoreCfgOnFlash()
```

How to control a single A7585DU

After connecting the CAEN A7585DU to the Arduino UNO as explained in Sec., it is possible to access the GitHub library repository and load the *DemoA7585.ino* sketch. This demo will init the A7585DU module and generate a ramp between 25V and 80V, increasing/decreasing the output voltage of 1V every second. This example demonstrates the compliance of the output setpoint swings between 20 and 85V but the maximum output is set to 80V for protection, limiting the full swing.

DemoA7585.ino

```
#include <A7585lib.h>


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

#define DEV_ADDRESS 0x46

```

```
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");

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);
}
```

From Arduino software, download the code on the processor and surf into Tool→ Serial Monitor. Set speed to 115200 and it will be possible to see the readback of the voltage set point.

How to control A7585DU with a Display shield

For this example, we used a *Deuligne* LCD Display by SNOOTLAB [3], which is based on Arduino. After connecting the display to the A7585DU as shown in Fig.4, it is possible to access the GitHub library repository and load the *DisplayA7585.ino* sketch. The display will be able to communicate with the A7585DU and, using buttons on the display keypad, it is possible to set HV output voltage (up/down key) and enable/disable the HV output (right key).

DisplayA7585.ino

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

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

#define DEV_ADDRESS 0x46

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

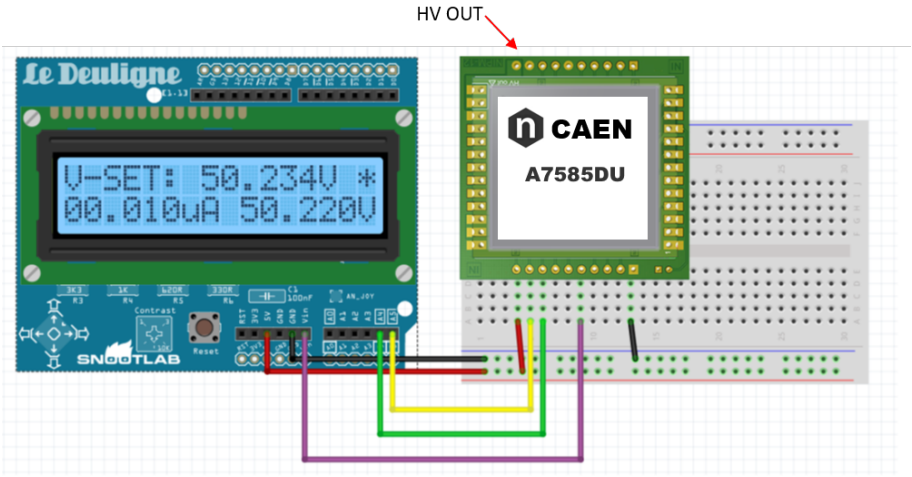

Fig. 4: Connection of the Deuligne display to the A7585DU.

```
11
12 int lcd_key     = 0;
13 int adc_key_in  = 0;
14
15 #define btnRIGHT  0
16 #define btnUP     1
17 #define btnDOWN   2
18 #define btnLEFT   3
19 #define btnSELECT 4
20 #define btnNONE   5
21 float vset=50;
22 bool on_off=false;
23 uint32_t  tdow;
24 uint32_t tl=0;
25 bool upd=false,dwd=false;
26 float inc=0.01;
27
28
29 A7585 A7585_dev;
30 void InitNIPM(int device_address)
31 {
32 A7585_dev.Init(device_address);
33 A7585_dev.Set_Mode(0);
34 A7585_dev.Set_MaxV(80);
35 A7585_dev.Set_MaxI(10);
36 A7585_dev.Set_RampVs(25);
37 A7585_dev.Set_V(50);
38 }
39
40
41 int read_LCD_buttons(){               // read the buttons
42 adc_key_in = analogRead(0);       // read the value from the sensor
43
44 // my buttons when read are centered at these valies: 0, 144, 329, 504, 741
45 // we add approx 50 to those values and check to see if we are close
46 // We make this the 1st option for speed reasons since it will be the most likely result
47
48 if (adc_key_in > 1000) return btnNONE;
49
50 // For V1.1 us this threshold
51 if (adc_key_in < 50)   return btnRIGHT;
52 if (adc_key_in < 250)  return btnUP;
53 if (adc_key_in < 450)  return btnDOWN;
54 if (adc_key_in < 650)  return btnLEFT;
55 if (adc_key_in < 850)  return btnSELECT;
56
57 // 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.
}


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, cur1;

lcd.setCursor(0,0);             // set the LCD cursor   position
lcd.print("V-SET:");            // print a message on the LCD
lcd.setCursor(6,0);             // set the LCD cursor   position
sprintf(tmp, "%d.%02d", (int)vset, (int)(vset*100)%100);
lcd.print(tmp);                 // print a message on the LCD
lcd.setCursor(11,0);
lcd.print("V");                 // print a message on the LCD

delta=millis()-tdow;
lcd.setCursor(15,0);
lcd.print(on_off?"*":"-");

cur1 = A7585_dev.GetIout();
volt1 = A7585_dev.GetVout();


lcd.setCursor(8,1);             // set the LCD cursor   position
sprintf(tmp, "%2d.%03duA", (int)cur1,(int32_t) (cur1*1000)%1000);
lcd.print(tmp);                 // print a message on the LCD

lcd.setCursor(0,1);             // set the LCD cursor   position
sprintf(tmp, "%2d.%03dV ", (int)volt1,(int32_t) (volt1*1000)%1000);
lcd.print(tmp);                 // print a message on the LCD



lcd_key = read_LCD_buttons();   // read the buttons

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

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;


}
```

How to control multiple A7585DU

In the following example we show how to control up to 16 A7585D/DU, with the possibility to easily extend the chain up to 127 modules. After connecting the display to the A7585DU as shown in Fig.5, it is possible to access the GitHub library repository and load the *MultipleDevA7585.ino* sketch. The code initializes all connected modules (with different I2C addresses) to generate 50V each and it enumerates them while scanning the I2C bus. The output is printed on the serial port.

MultipleDevA7585.ino

```
#include <A7585lib.h>


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

#define DEV_ADDRESS_BASE 0x40

A7585 A7585_dev[16];

```

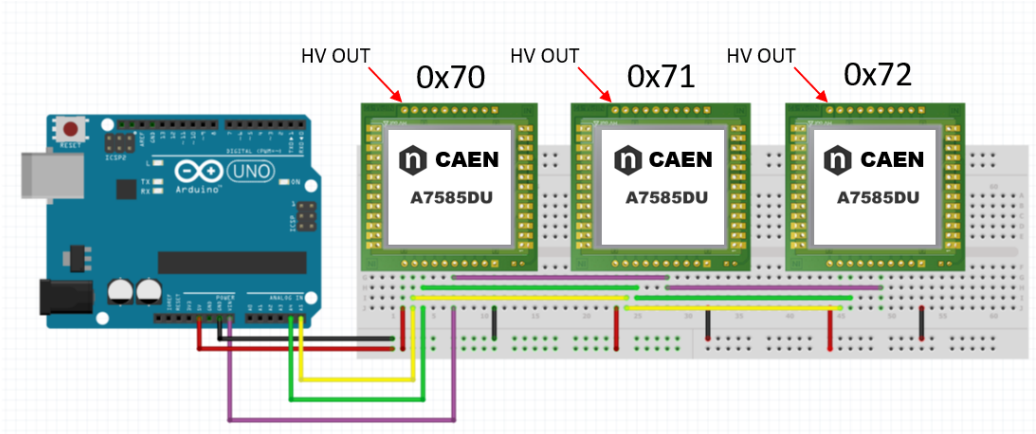

Fig. 5: Connection scheme for three A7585DU controlled by a single Arduino UNO.

```
12
13 void setup() {
14 int i;
15
16 Serial.begin(115200);
17 Wire.begin();
18 Serial.println("Starting A7585 HV demo app. This app will ramp the HV");
19
20 for (i=0;i<16;i++)
21 {
22 A7585_dev[i].Init(DEV_ADDRESS_BASE+i);
23 A7585_dev[i].Set_Mode(0);
24 A7585_dev[i].Set_MaxV(80);
25 A7585_dev[i].Set_MaxI(10);
26 A7585_dev[i].Set_RampVs(25);
27 A7585_dev[i].Set_V(50);
28
29 Serial.print("Device ID ");
30 Serial.print(DEV_ADDRESS_BASE+i);
31 if (A7585_dev[i].GetConnectionStatus())
32 Serial.println(" connected");
33 else
34 Serial.println(" not connected");
35 A7585_dev[i].Set_V(50);
36 A7585_dev[i].Set_Enable(true);
37 }
38
39
40
41
42 }
43
44 void loop() {
45
46 int i;
47 for (i=0;i<16;i++)
48 {
49
50
51 if (A7585_dev[i].GetConnectionStatus())
52 {Serial.print( DEV_ADDRESS_BASE + i); Serial.print(' ');}
53 else
54 {Serial.print('-'); Serial.print(' ');}
55 }
56 Serial.println(' ');
57 delay(1000);
58 }
```

From Arduino software, download the code on the processor and surf into Tool→ Serial Monitor. Set speed to 115200 and the software will output a new line every second, showing the address of the detected module.

References

[1] UM6377 –A7585D/DU User Manual.

[2] *Arduino UNO*. URL: https://store.arduino.cc/arduino-uno-rev3.

[3] *SNOOTLAB Deuligne LCD*. URL: http://shieldlist.org/snootlab/le-deuligne.

[4] *Arduino platform*. URL: https://store.arduino.cc/.

[5] *Raspberry PI platform*. URL: https://www.raspberrypi.org/.

CAEN Tools for Discovery

CAEN SpA
Via Vetraia 11
55049 – Viareggio Italy
Tel +39.0584.388.398
Fax +39.0584.388.959

info@caen.it www.caen.it

www.caen.it