APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi Analogue Input

DOCUMENT DATE:	**12th APRIL 2019**
DOCUMENT REVISION:	**1.1**

WWW.4DSYSTEMS.COM.AU

Description

This application note demonstrates how to perform Analog-to-Digital Conversion (ADC) and how to print and graph the results using a Diabo16 display module. The 4DGL code of the Designer project can be copied and pasted to an empty ViSi project and it should compile normally. The code can also be integrated to that of an existing ViSi project.

Before getting started, the following are required:

- Any of the following 4D Diablo 16 display modules and gen4 Diablo 16 Intelligent modules:

gen4-uLCD-24D series	gen4-uLCD-28D series	gen4-uLCD-32D series
gen4-uLCD-35D series	gen4-uLCD-43D series	gen4-uLCD-50D series
gen4-uLCD-70D series		
uLCD-35DT	uLCD-43D series	uLCD-70DT

  Visit www.4dsystems.com.au to see the latest display module products that use the Diablo16 processor.

- 4D Programming Cable / uUSB-PA5/uUSB-PA5-II for non-gen4 displays(uLCD-xxx)
- 4D Programming Cable & gen4-PA, / gen4-IB / 4D-UPA for gen4 displays (gen4-uLCD-xxx)
- Workshop 4 IDE (installed according to the installation document)
- A 10-kiloohm potentiometer, a breadboard, and connecting wires
- When downloading an application note, a list of recommended application notes is shown. It is assumed that the user has read or has a working knowledge of the topics presented in these recommended application notes.

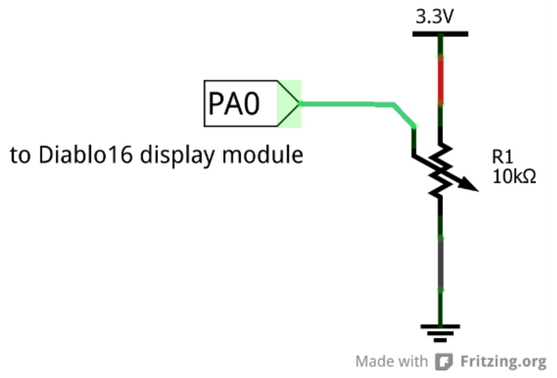
Content

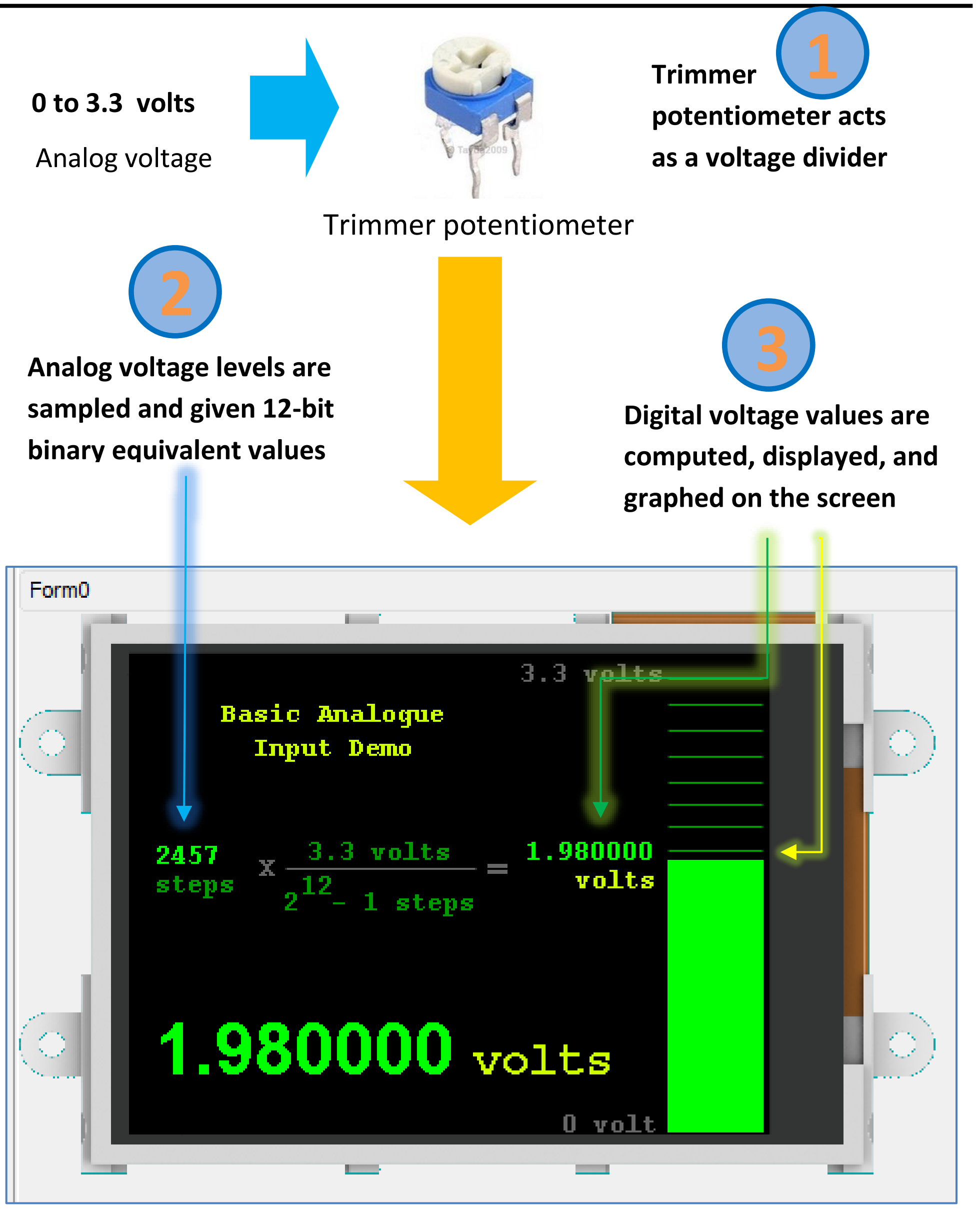
Application Overview

The Designer environment enables the user to write 4DGL code in its natural form to program the display module. 4DGL is a graphics oriented language allowing rapid application development, and the syntax structure was designed using elements of popular languages such as C, Basic, Pascal and others. Programmers familiar with these languages will feel right at home with 4DGL.

The Diablo16 has four pins (PA0, PA1, PA2, and PA3) which can be configured as analog inputs. These four pins can operate in any of the three analog modes – the standard, averaged, and high-speed modes. These modes differ in sampling rate and resolution. **The maximum analog input voltage for all modes is 3.3 volts**. In the application developed in this document, the standard mode (12-bit, >40k samples per second) is used.

The application works in a manner illustrated in the diagram on the next page. First, the wiper of a trimmer potentiometer, acting as a voltage divider, is connected to pin PA0 of the Diablo16 display module.




0 to 3.3 volts
Analog voltage
Trimmer potentiometer
1
Trimmer potentiometer acts as a voltage divider
2
Analog voltage levels are sampled and given 12-bit binary equivalent values
3
Digital voltage values are computed, displayed, and graphed on the screen
Form0
3.3 volts
Basic Analogue Input Demo
2457 steps
x
3.3 volts
2^12 – 1 steps
=
1.980000 volts
1.980000 volts
0 volt

The voltage level at pin PA0 can be of any value between 0 and 3.3 volts. Thru ADC, an analog voltage level sampled by pin PA0 is given a 12-bit binary equivalent value. Given the reference voltage, which is 3.3 volts, the digital voltage value can now be computed and displayed onscreen.

Text and graphics objects such as lines and rectangles are used to display and graph the results onscreen. The 4DGL commands for printing text and drawing lines and rectangles are easy and straight forward, the basics of which are discussed in the following application notes:

Designer Getting Started - First Project
Designer or ViSi How to Draw Circles and Rectangles

This document comes with a Designer file, the 4DGL code of which is discussed in the section "Understanding the Code". The code can be used to develop more complex applications related to ADC. The last section shows how the display module and trimmer potentiometer are connected.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **Designer** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note

Designer Getting Started - First Project

For instructions on how to launch Workshop 4, how to open a **ViSi** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

Create a New Project

For instructions on how to create a new **Designer** project, please refer to the section "**Create a New Project**" of the application note

Designer Getting Started - First Project

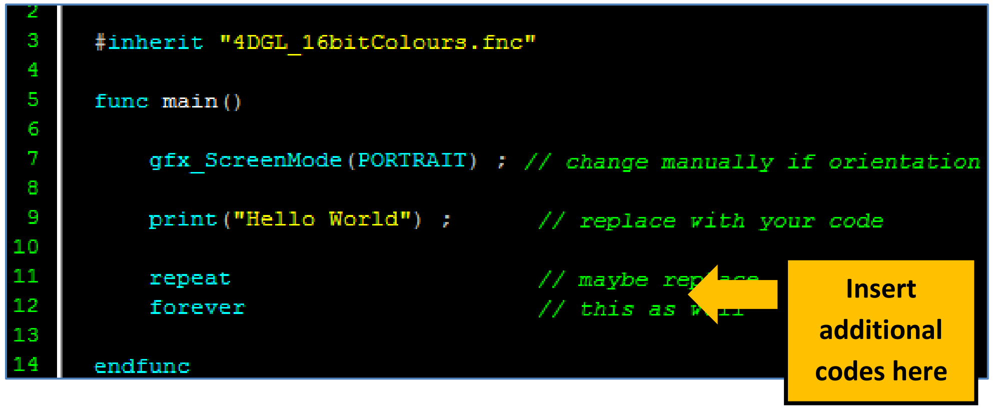
For instructions on how to create a new **ViSi** project, please refer to the section "**Create a New Project**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

Design the Project

The Program Skeleton

Everything is now ready to start designing a project. Workshop automatically provides the program skeleton, which contains basic parts needed to start designing an application. The user is advised to have a copy of the DIABLO16 Internal Functions Manual on hand. The document contains detailed information on the different functions used in this application note. To start coding, place additional codes before line 11. If repetitive operation is desired, the programmer can also insert additional codes between lines 11 and 12 instead. The application note Designer Getting Started - First Project shows the basics of creating a Designer program.



```
 2
 3  #inherit "4DGL_16bitColours.fnc"
 4
 5  func main()
 6
 7      gfx_ScreenMode(PORTRAIT) ; // change manually if orientation
 8
 9      print("Hello World") ;      // replace with your code
10
11      repeat                      // maybe replace
12      forever                     // this as well
13
14  endfunc
```

Understanding the Code

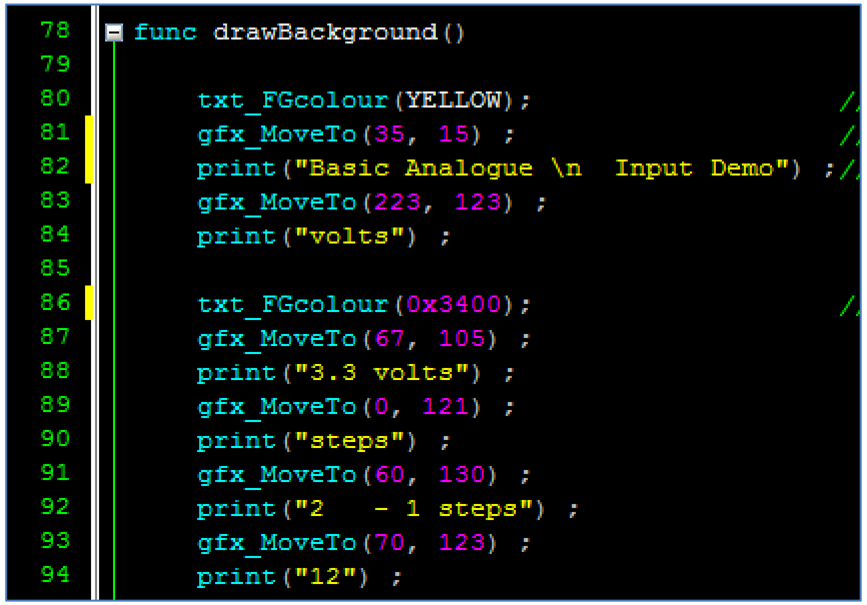
The code is well commented for the benefit of the user. The following sections will serve as a supplementary explanation. Open the attached Designer file to have a better and complete view of the code. It can be broken down into five separate steps:

1. Draw background labels
2. Configure pin PA0 as an analog input
3. Read from pin PA0
4. Compute for the equivalent digital voltage
5. Display and graph the results on the screen

Items 1 and 2 are executed only once at the start of the program. Items 3 to 5 are executed indefinitely using a loop.

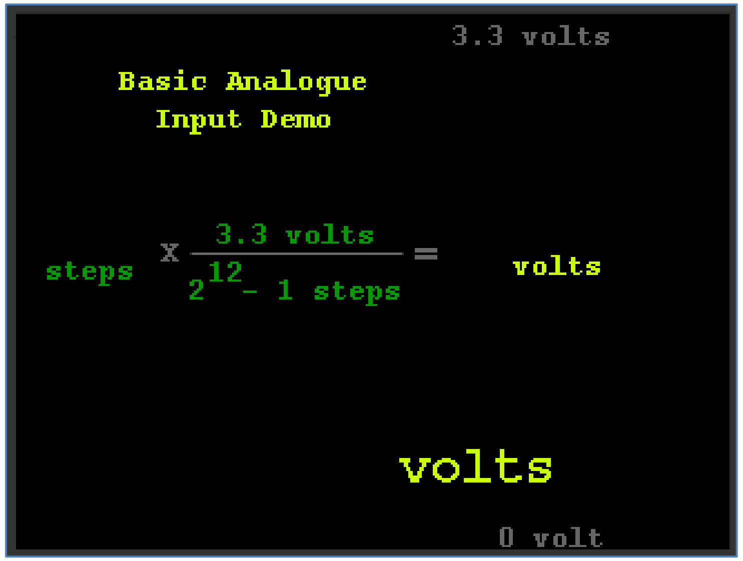
Draw the Background Labels

The code for drawing the background is defined in a separate function, **drawBackground( )**, so as to keep the main function simple. The function **drawBackground( )** is defined just after the main function.

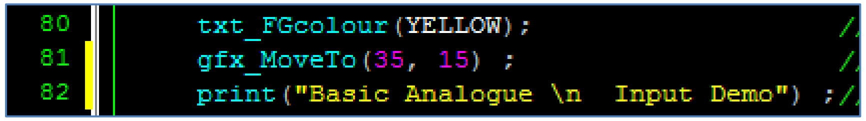


```
78  func drawBackground()
79
80      txt_FGcolour(YELLOW);                          /
81      gfx_MoveTo(35, 15) ;                           /
82      print("Basic Analogue \n  Input Demo") ;/
83      gfx_MoveTo(223, 123) ;
84      print("volts") ;
85
86      txt_FGcolour(0x3400);                          /
87      gfx_MoveTo(67, 105) ;
88      print("3.3 volts") ;
89      gfx_MoveTo(0, 121) ;
90      print("steps") ;
91      gfx_MoveTo(60, 130) ;
92      print("2    - 1 steps") ;
93      gfx_MoveTo(70, 123) ;
94      print("12") ;
```

Scroll down on the Designer window to see the complete code. All that this function does is draw the static labels for the display, as shown below.



To print the text “Basic Analogue Input Demo”:

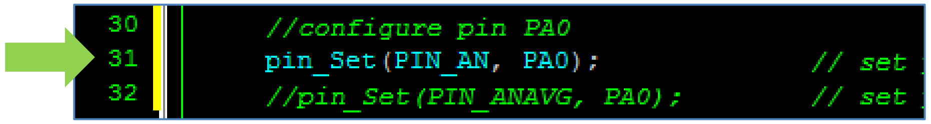


```
80      txt_FGcolour(YELLOW);                          /
81      gfx_MoveTo(35, 15) ;                           /
82      print("Basic Analogue \n  Input Demo") ;/
```

The statement in line 80 will change the text font colour to yellow. The statement in line 81 will move the origin to point (35, 15) on the screen. From this point the text will be printed. Omitting line 81 will result to the text being printed starting from point (0,0) (top left corner) which is the default origin. The special character ‘\n’ is for a newline. The other labels are printed using the same functions.

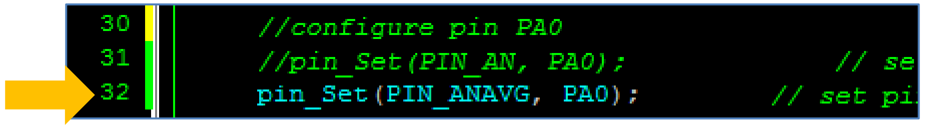
Configure Pin PA0 as an Analog Input

To configure pin PA0 as an analog input (standard mode), use the function below.



```
        //configure pin PA0
        pin_Set(PIN_AN, PA0);                  // set
        //pin_Set(PIN_ANAVG, PA0);             // set
```

To configure pin PA0 as an analog input (averaged mode), use the function below.



```
        //configure pin PA0
        //pin_Set(PIN_AN, PA0);                    // se
        pin_Set(PIN_ANAVG, PA0);               // set pi
```

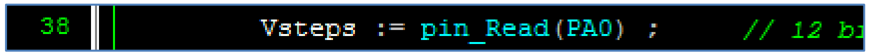
Standard mode results in a single sample being immediately read. This mode can read over 40000 values per second at a 12-bit resolution.

Averaged mode results in 16 samples being immediately read and their average returned. Averaged mode can read approximately 20000 values per second at a 12-bit resolution.

High-speed mode collects a user specified number of samples at a user specified rate/frequency and can execute a user function when complete. The updated value updates approximately 250000 times across 1-4 channels. This mode operates at a 10-bit resolution. The use of this mode will be demonstrated along with the use of the **gfx_Scope ( )** function in a separate application note.

Read from Pin PA0

The statement

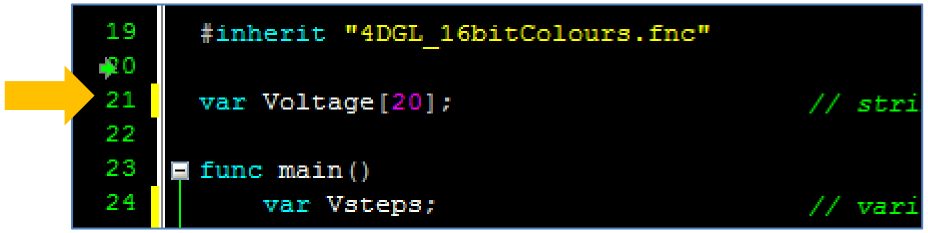


```
            Vsteps := pin_Read(PA0) ;        // 12 bi
```

will return the result of ADC to the variable **Vsteps**. Since ADC resolution is 12 bits, the range of possible values for **Vsteps** is 0 to 4095. Take note that this line is now inside the infinite for loop.

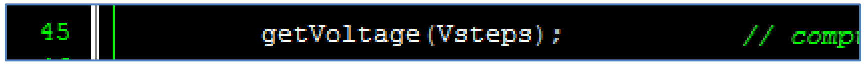
Compute for the Equivalent Digital Voltage

Note that at the beginning part of the code, a global array is declared for storing the computed digital voltage value.



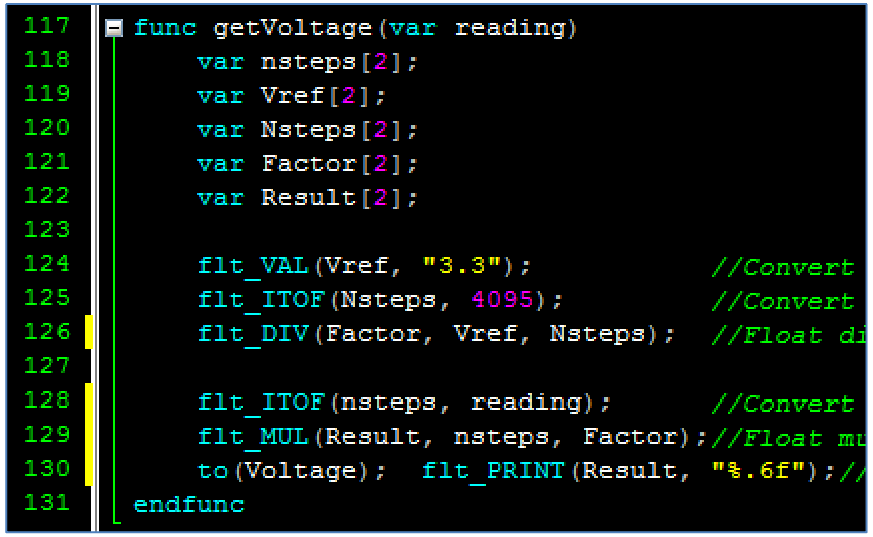
```
#inherit "4DGL_16bitColours.fnc"

var Voltage[20];                             // stri

func main()
    var Vsteps;                              // vari
```

The function below is then used to compute the equivalent digital voltage value of the reading held by **Vsteps**.



```
            getVoltage(Vsteps);              // comp
```

This function is defined after the main function.

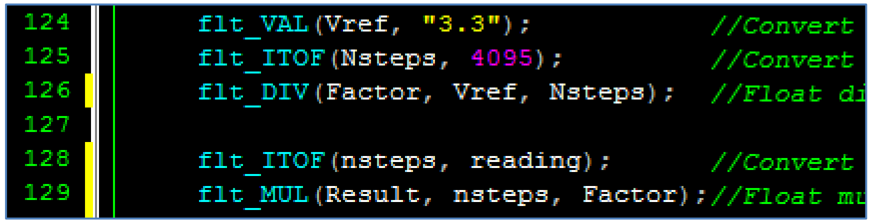


```
func getVoltage(var reading)
    var nsteps[2];
    var Vref[2];
    var Nsteps[2];
    var Factor[2];
    var Result[2];

    flt_VAL(Vref, "3.3");           //Convert
    flt_ITOF(Nsteps, 4095);         //Convert
    flt_DIV(Factor, Vref, Nsteps);  //Float di

    flt_ITOF(nsteps, reading);      //Convert
    flt_MUL(Result, nsteps, Factor);//Float mu
    to(Voltage);  flt_PRINT(Result, "%.6f");//
endfunc
```

After **getVoltage ( )** has been called, the computed equivalent digital voltage value, which is actually a string, is then stored in the global array **Voltage**. The function performs the computation below.

$$\boldsymbol{Voltage} = \boldsymbol{reading}\ steps\ x\ \frac{\mathbf{3.3}\ volts}{\mathbf{4095}\ steps}$$

Observe that inside the function, there are several closely related functions used.

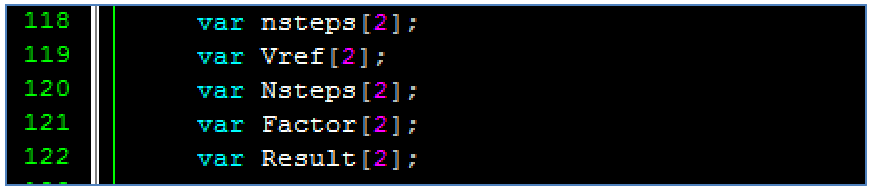


```
    flt_VAL(Vref, "3.3");           //Convert
    flt_ITOF(Nsteps, 4095);         //Convert
    flt_DIV(Factor, Vref, Nsteps);  //Float di

    flt_ITOF(nsteps, reading);      //Convert
    flt_MUL(Result, nsteps, Factor);//Float mu
```

These are 4DGL floating point functions. Note that when performing the computation

$$\boldsymbol{Voltage} = \boldsymbol{reading}\ steps\ x\ \frac{\mathbf{3.3}\ volts}{\mathbf{4095}\ steps}$$

floating point numbers are involved. Floating point numbers in 4DGL are stored in float variables, which are two-word arrays.



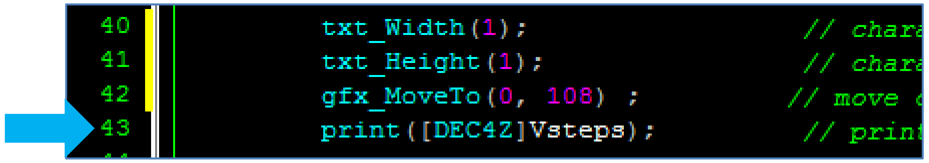
```
    var nsteps[2];
    var Vref[2];
    var Nsteps[2];
    var Factor[2];
    var Result[2];
```

The lines containing floating point functions have been commented for the benefit of the user. These functions are documented in section 2.21 of the DIABLO16 Internal Functions Manual. Discussion of floating point numbers and functions will be covered in a separate application note.

Display and Graph the Results on the Screen

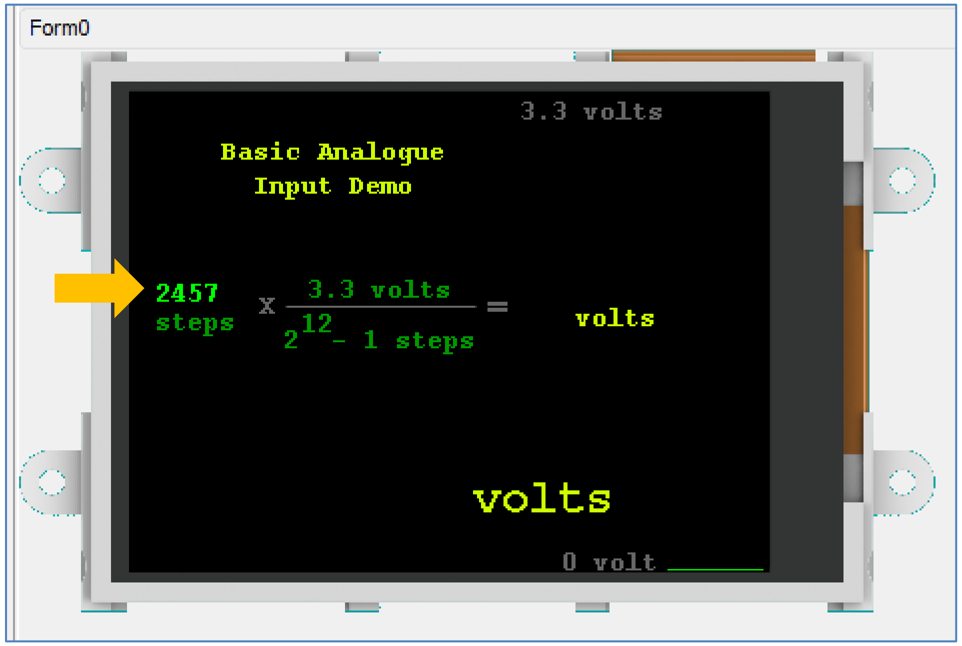
Print the Results on the Screen

To print the value of **Vsteps** before it is scaled:



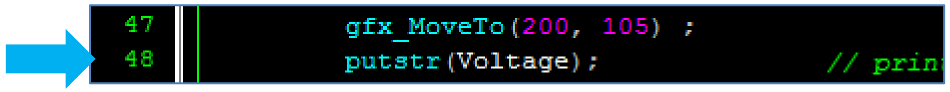
```
        txt_Width(1);                       // chara
        txt_Height(1);                      // chara
        gfx_MoveTo(0, 108) ;               // move c
        print([DEC4Z]Vsteps);               // print
```

The result is shown below.



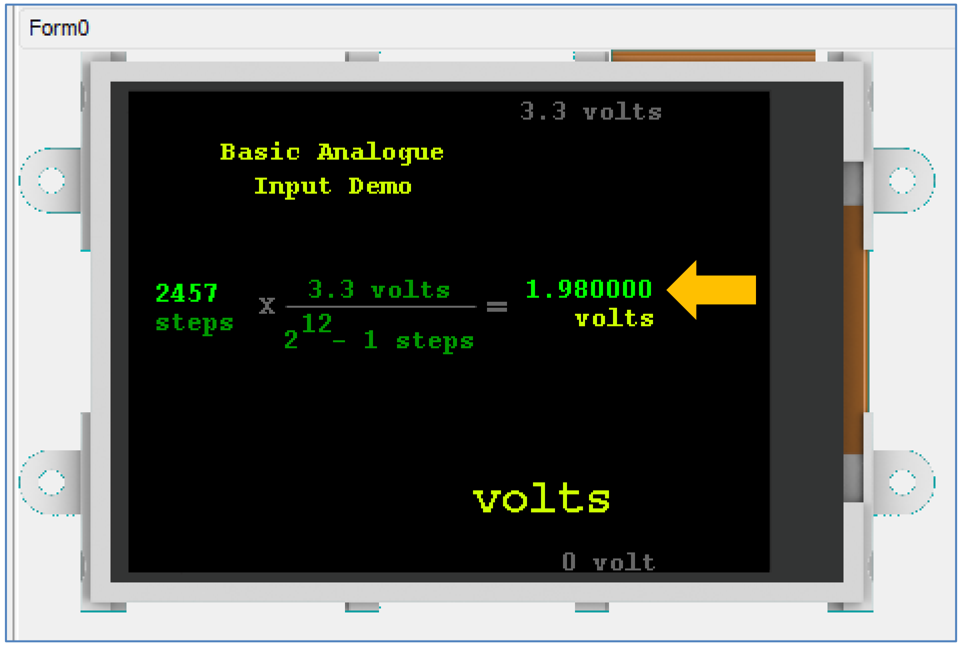
[DEC4Z] is the format for the number to be printed. See more number formats in section 2.4.7 of the DIABLO16 Internal Functions Manual.

To print the content of **Voltage**,

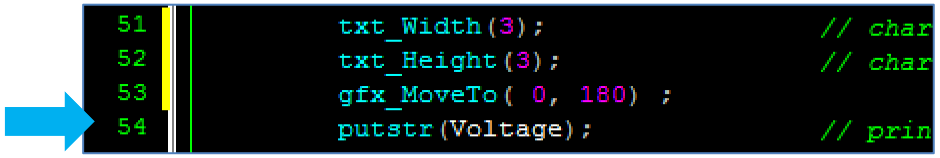


```
        gfx_MoveTo(200, 105) ;
        putstr(Voltage);                    // prin
```

The statement indicated above is an alternative function for printing strings on the screen. The result is shown below.

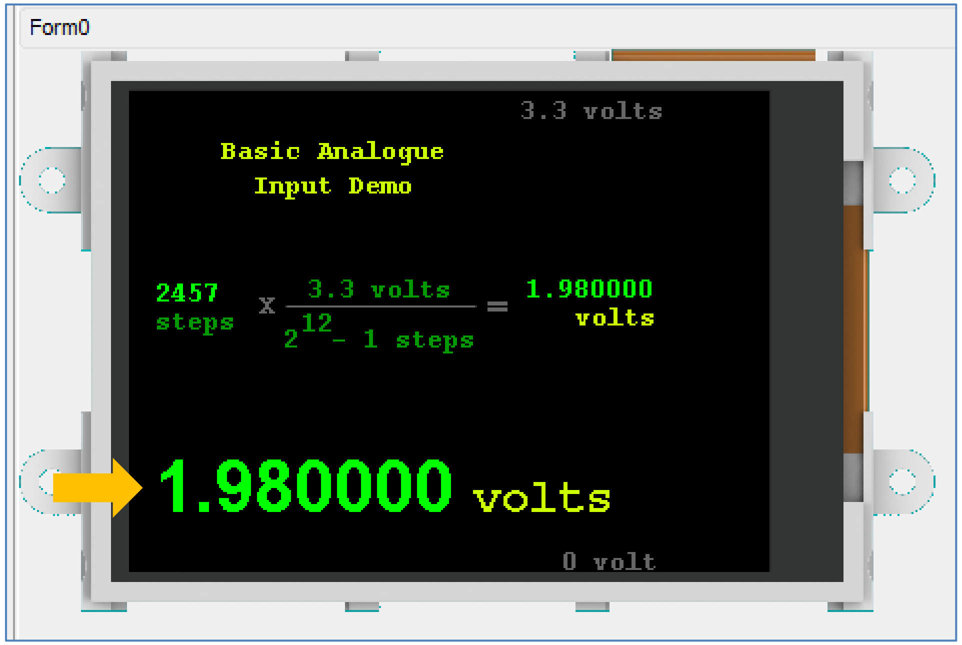


The contents of **Voltage** is printed again in a larger font size through the use of the functions **txt_Width ( )** and **txt_Height ( ).** Here, the font dimensions are multiplied by a factor of 3.



```
            txt_Width(3);                    // char
            txt_Height(3);                   // char
            gfx_MoveTo( 0, 180) ;
            putstr(Voltage);                 // prin
```

The outcome is shown below.



Graph the Results on the Screen

Two rectangles and multiple lines are used to make a visual representation of the voltage readings. Again, the variable **Vsteps** can be of any value from 0 to 4095. This range is scaled down to a range of 0 to 230, since the rectangles to be used have a varying height of only 0 to 230 pixels.

```
//scale down the range 0 to 4095 of Vsteps to a r
//so we can graph the results onscreen
Vsteps *=   8 ;                 // 0 to 32760
Vsteps /= 142 ;                 // 0 to 230
```

Remember that the screen has a limited VGA resolution (320x480 pixels for the uLCD-35DT).

Shorthand in 4DGL

The statement

```
Vsteps *=   8 ;                 // 0 to 32760
```

is the same as

```
Vsteps := Vsteps * 8;
```

The statement

```
Vsteps /= 142 ;                 // 0 to 230
```

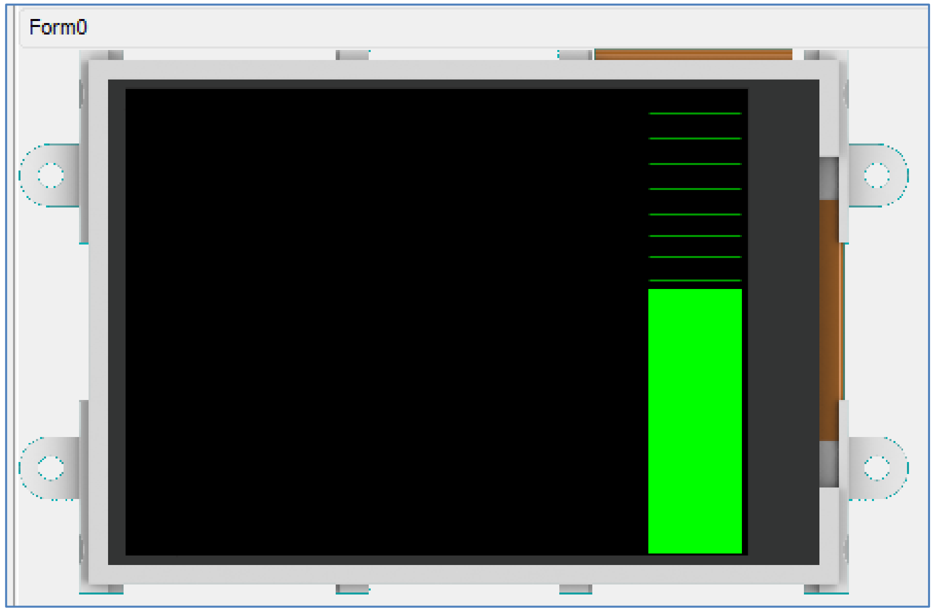
is the same as

```
Vsteps := Vsteps / 8;
```

Black and green rectangles, the varying height dimensions of which are determined by **Vsteps,** are then used to show a bar graph.

```
//use the scaled value of Vsteps to draw rectangles
gfx_RectangleFilled( 290, 5, 320, 235 - Vsteps, BLACK) ;//

// draw multiple horizontal lines
for(n := 5; n < 320; n += 10)
    gfx_Line(290, n, 320, n, GREEN);
next

gfx_RectangleFilled( 290,  235 - Vsteps, 320, 235, LIME  )
```

Multiple green lines are also added.



To learn more about drawing lines and rectangles, refer to the following application notes.

Designer Getting Started - First Project
Designer or ViSi How to Draw Circles and Rectangles

A 200-millisecond delay is then added to make the displayed value more readable. In effect, five readings are displayed every second.

```
pause(200); //to slow down sampling rate
```

Note: The various analog modes can interfere with the operation of the touch screen if their functions are called too frequently. It is recommended to limit the calls of the analog functions to a maximum of once every millisecond. Please refer to the Internal Functions documentation for further information on this topic.

Run the Program

For instructions on how to save a **Designer** project, how to connect the target display to the PC, how to select the program destination, and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

Designer Getting Started - First Project

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination (this option is not available for Goldelox displays), and how to compile and download a program, please refer to the section "**Run the Program**" of the application note

ViSi Getting Started - First Project for Picaso and Diablo16

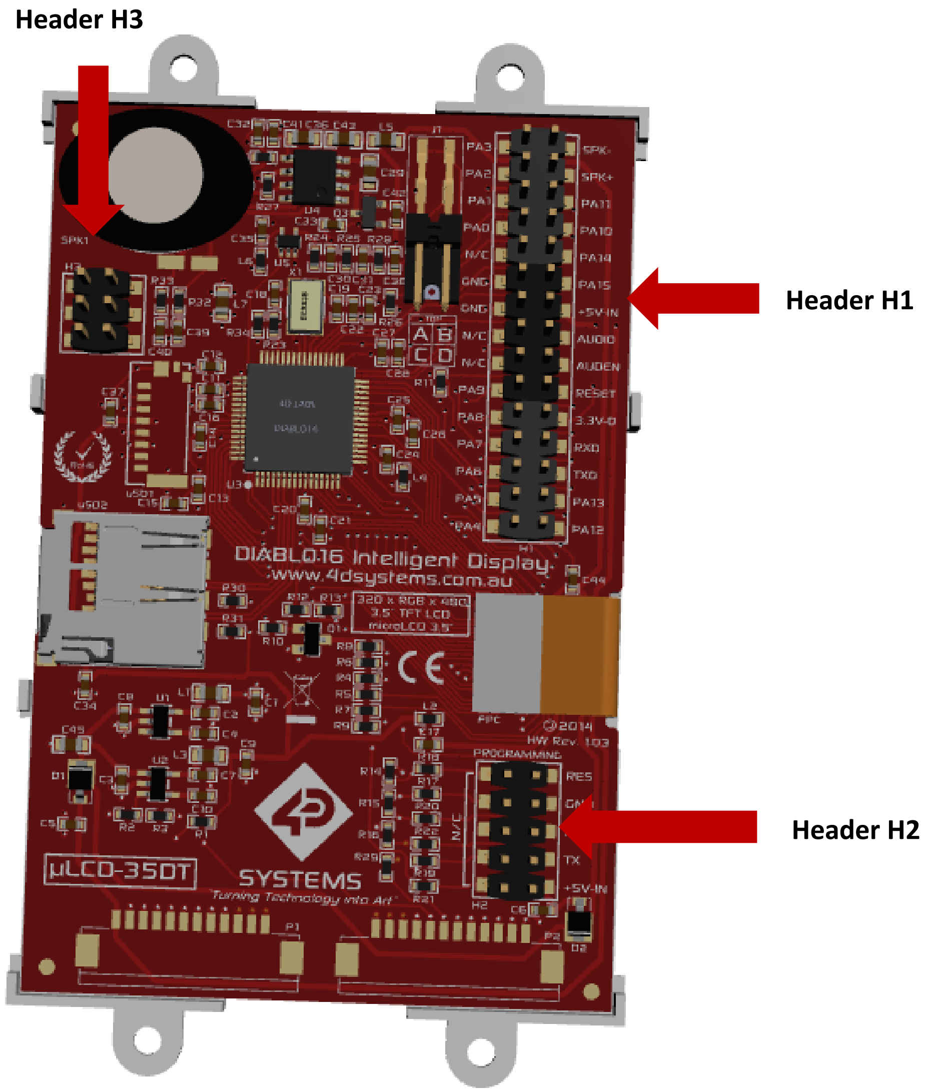
The uLCD-35DT display module is commonly used as an example, but the procedure is the same for other displays.

Connect the Trimmer Potentiometer

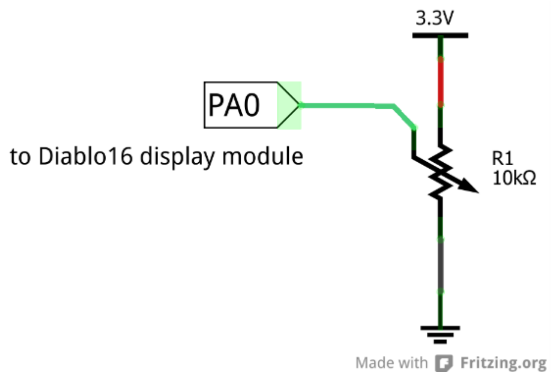
Datasheet and Pin Configuration

Download the datasheet for your display module to get acquainted with the pin configurations. The datasheet for a display module can be downloaded from its product page: http://www.4dsystems.com.au/products/. Here the uLCD-35DT will be used as a model.

Section 3 of the uLCD-35DT datasheet shows the device's pin configuration.

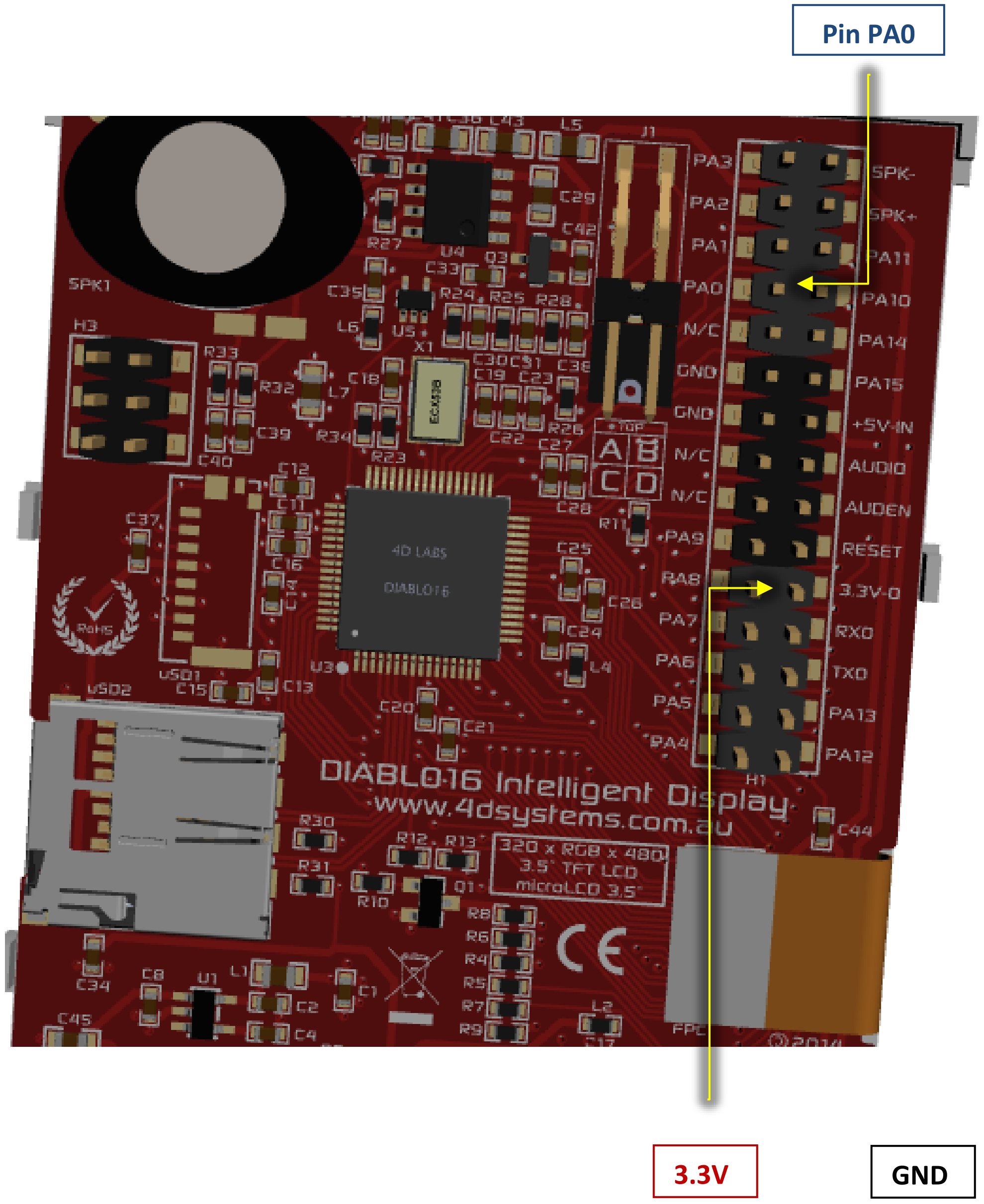


The Trimmer Potentiometer



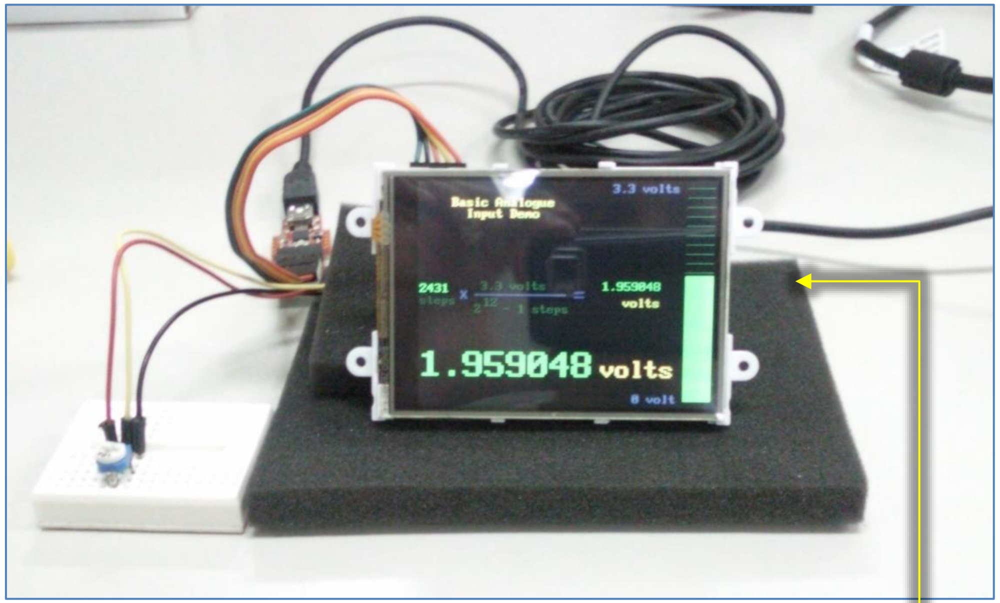
Header H1

Pin PA0 is in header H1 as indicated in the image below. The Diablo16 also has a 3.3-volt output, which can be used to supply voltage across the trimmer potentiometer.

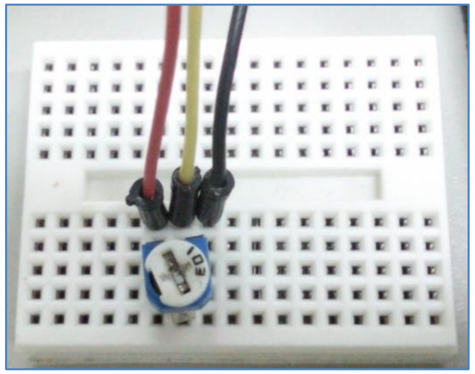


The Project

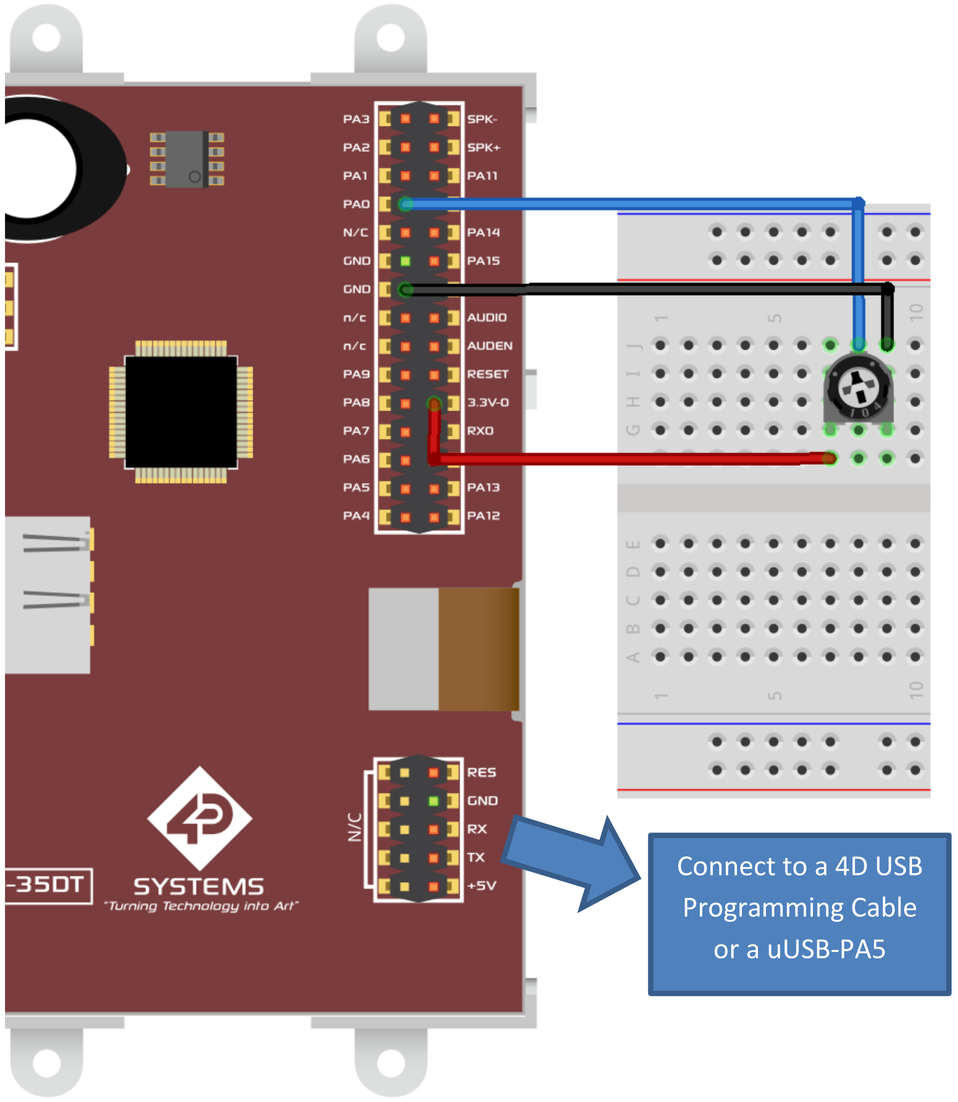
Complete Setup



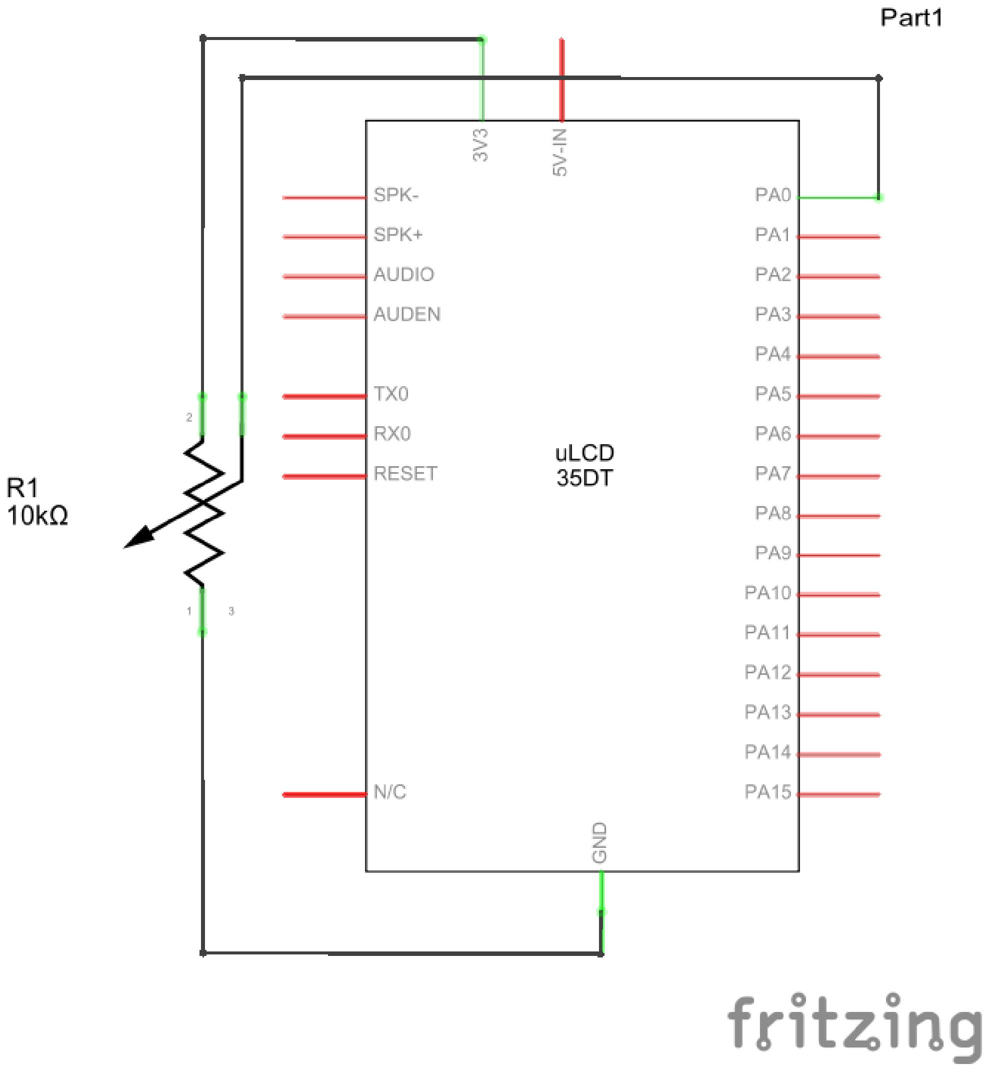
The Trimmer Potentiometer



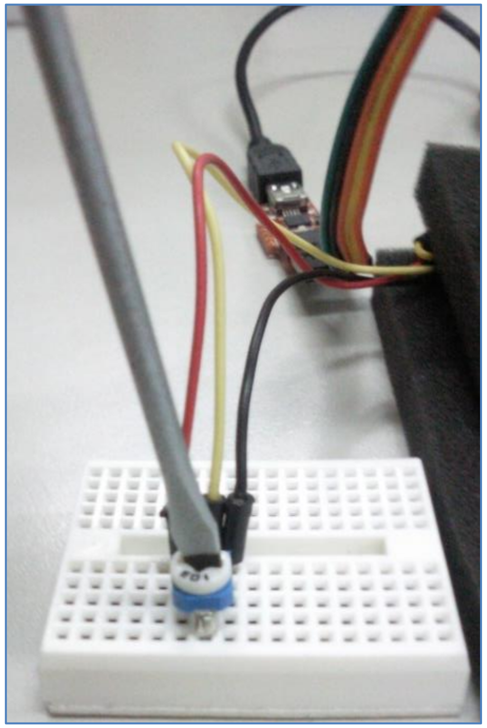
Connection Diagram



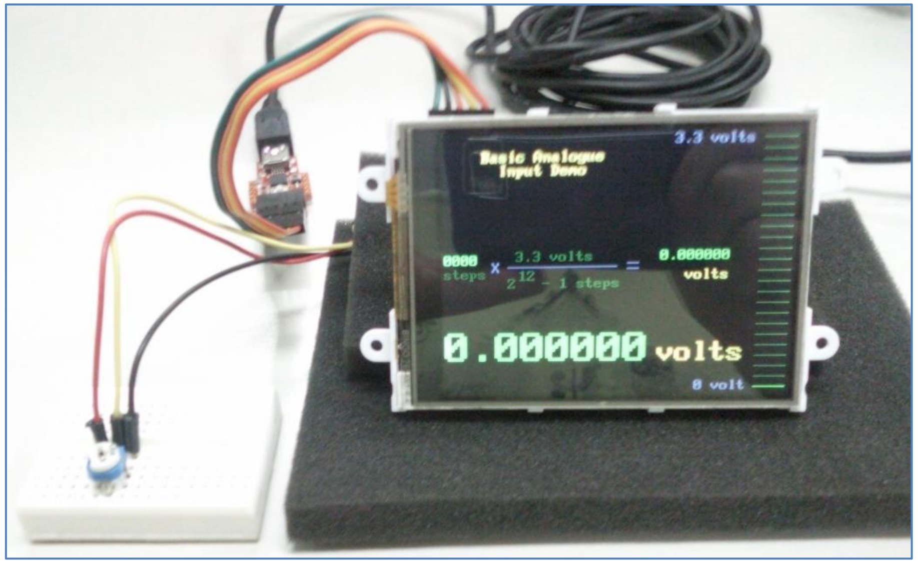
Schematic Diagram



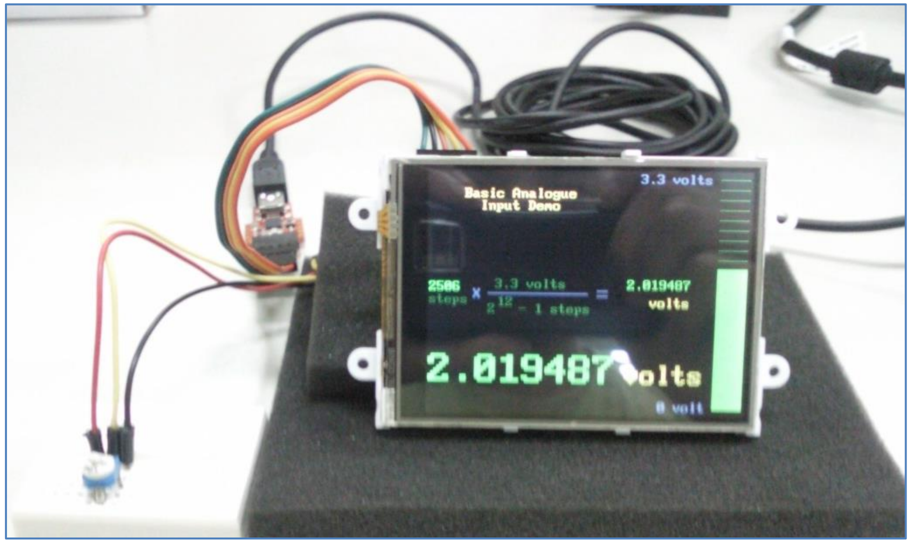
Adjusting the Trimmer Potentiometer



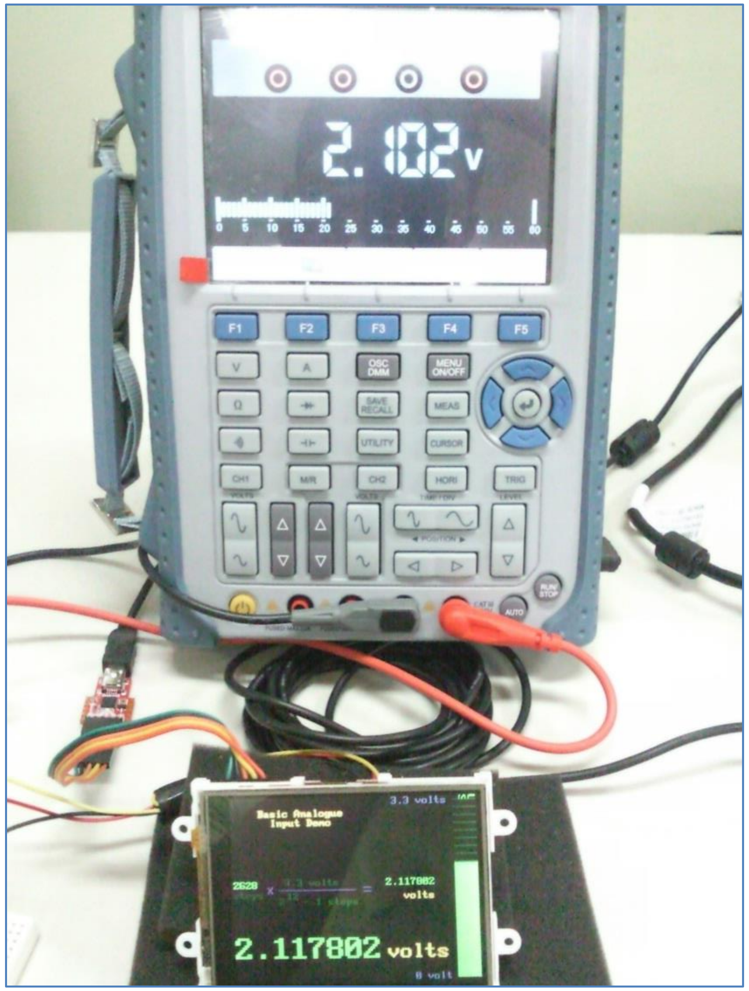
Voltage at Minimum



At around 2.0 Volts



With a Digital Multimeter



Proprietary Information

Disclaimer of Warranties & Limitation of Liability