



Visi Genie Magic: Basic I2C Application

DOCUMENT DATE: **7th March 2020**

DOCUMENT REVISION: **1.0**



Description

This Application note demonstrates how to interface MOTG-MP3 with 4D systems display.

Before getting started, the following are required:

Hardware

- Two of Any [4D Systems display module](#) powered by any of the following processors:
 - o DIABLO16
 - o PICASO
 - o PIXXY-28/44
- One [Programming Adaptor for target display module](#)
- Arduino Uno

Software

- [Workshop4](#)
- This requires the **PRO** version of Workshop4

This application note comes with one (1) Visi Genie projects:

- Visi-Genie-I2C.4DGenie

Note: Using a non-4D programming interface could damage the processor and void the warranty.

Content

Description2

Content.....2

Application Overview2

Setup Procedure3

Create a New Project.....3

Design the Project3

Add Meter3

Add Buttons4

Add Magic Code5

Add Magic Event.....6

Add the Magic Events to the Buttons.....8

Hardware Connections.....8

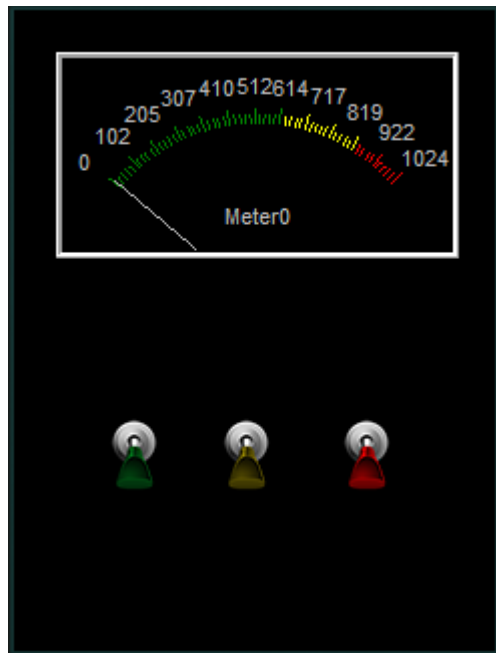
Run the Program10

Proprietary Information11

Disclaimer of Warranties & Limitation of Liability.....11

Application Overview

This application note demonstrates how to apply I2C communication between Arduino and 4D Systems display.



Setup Procedure

For instructions on how to launch Workshop4, how to open a ViSi-Genie project, and how to change the target display, kindly refer to the section “**Setup Procedure**” of the application note:

- [ViSi Genie Getting Started – First Project for Picaso Displays](#)
- [ViSi Genie Getting Started – First Project for Diablo16 Displays](#)

Create a New Project

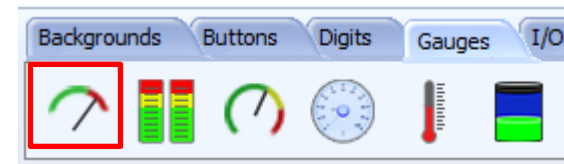
For instructions on how to create a new **ViSi Genie** project, please refer to:

- [ViSi Genie Getting Started – First Project for Picaso Displays](#)
- [ViSi Genie Getting Started – First Project for Diablo16 Displays](#)

Design the Project

Add Meter

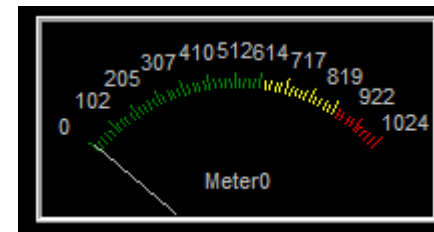
Go to Gauges and add meter.



From the currently add meter, go to its properties and adjust maximum to 1024 and minimum value to 0.

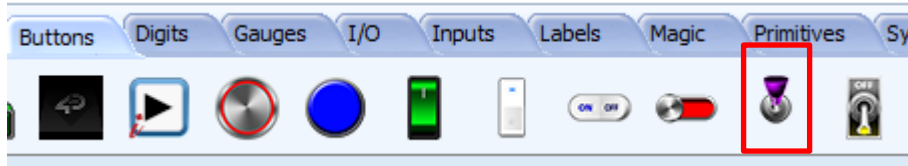
Left	20
Maxvalue	1024
Minvalue	0

Adjust meter image size and position.



Add Buttons

Go to Buttons add three toggle buttons.



Arrange the buttons.



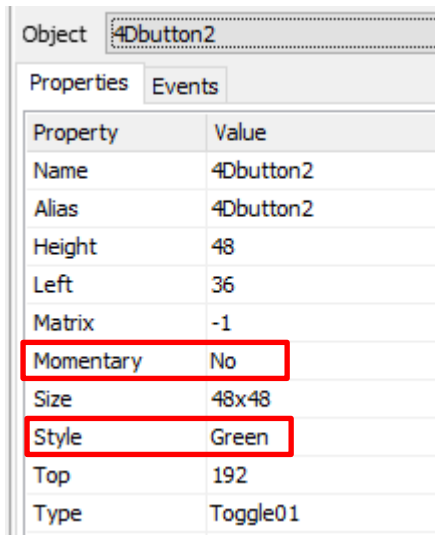
Go to 4Dbutton0 properties, set Momentary to NO and Style to Yellow.

Object	4Dbutton0
Properties Events	
Property	Value
Name	4Dbutton0
Alias	4Dbutton0
Height	48
Left	92
Matrix	-1
Momentary	No
Size	48x48
Style	Yellow
Top	192
Type	Toggle01

Go 4Dbutton1 properties, set Momentary to NO and Style to Red.

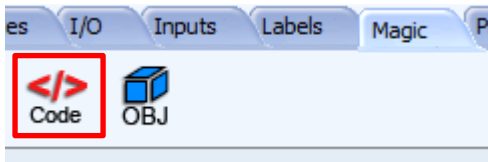
Object	4Dbutton1
Properties Events	
Property	Value
Name	4Dbutton1
Alias	4Dbutton1
Height	48
Left	152
Matrix	-1
Momentary	No
Size	48x48
Style	Red
Top	192
Type	Toggle01

Go to 4Dbutton2 properties, set Momentary to NO and Style to Green.

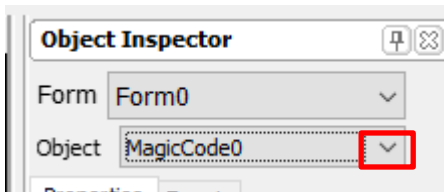


Add Magic Code

Add three magic codes by clicking it three times.



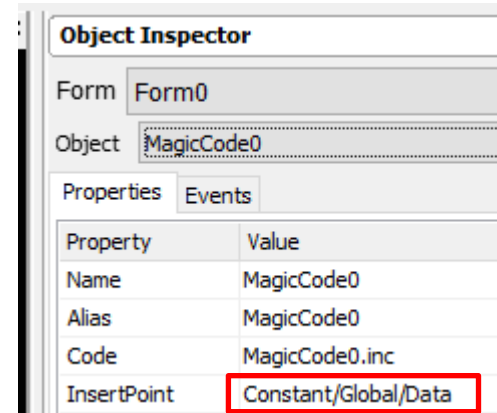
To see the added magic codes, go to object inspector and click the Object's drop down button.



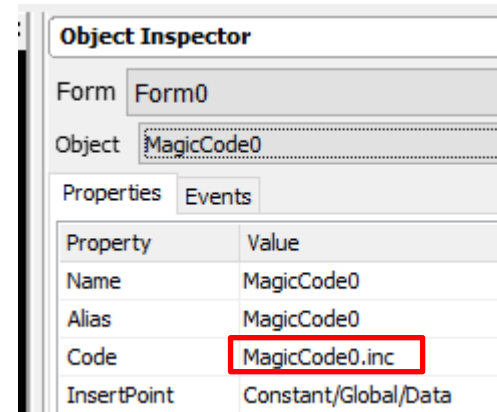
After clicking the drop down button you can see three magic codes.



Set MagicCode0 InsertPoint to Constant/Global/Data



Click MagicCode0.inc to open the code window.

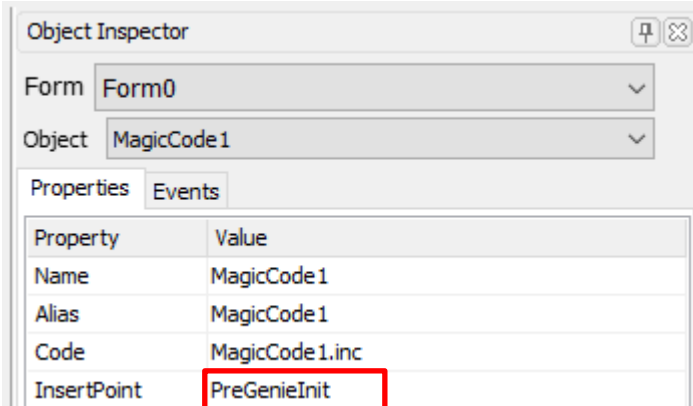


The codes inside MagicCode0. ARDUINO_ADDRESS is the I2C address of the Arduino which is 5, SDA pin is PA3, SCL pin is PA2, Buffer[2] is the storage of received data from the slave.

MagicCode0.inc

```
1 #constant ARDUINO_ADDRESS 5
2 #constant SDA_pin PA3 //GPIO
3 #constant SCL_pin PA2 //GPIO
4 var buffer[2]; //2bytes * 2 =
5 var mybuffer[1]; //2bytes * 2
6 var potentiometer_value; //
7 var ptr, myptr; // variable f
```

Set MagicCode1 InsertPoint to PreGenieInit.



Inside the MagicCode1 Code.

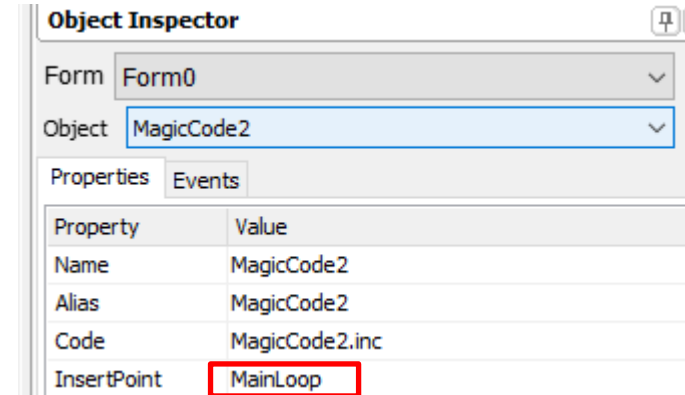
Create pointers for buffer and mybuffer. And

Setup I2C1 Speed, SCL and SDA pins.

MagicCode1.inc

```
1 ptr:=str_Ptr(buffer);
2 myptr:=str_Ptr(mybuffer);
3 I2C1_Open(I2C_SLOW, SCL_pin, SDA_pin);
```

Set MagicCode2 InsertPoint to MainLoop.



To request data from the slave shift ARDUINO_ADDRESS to left by one and Add One. The data is stored in the buffer which receive 3 bytes of data.

MagicCode2.inc

```
1 I2C1_Idle(); // wait until i2c is inactive
2 I2C1_Start(); //start i2c communication
3 I2C1_Write((ARDUINO_ADDRESS <<1) + 1); //s
4 I2C1_Getn(buffer, 3); //get three bytes of d
5 I2C1_Stop(); //stop i2c communication
```

Get the received data in the buffer using the pointer ptr. Shift the first byte to left by 8 and add the third byte. The first byte which is 48 is the header of the 2 bytes potentiometer value.

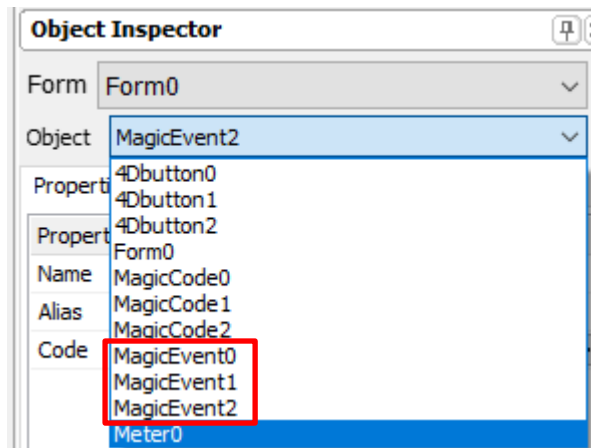
```
6 if(str_GetByte(ptr + 0) == 48) //check the header first
7   potentiometer_value := str_GetByte(ptr + 1) << 8 + str_GetByte(ptr + 2);
8   WriteObject(potmeter, 0, potentiometer_value); // put the potentiometer val
9 endif
```

Add Magic Event

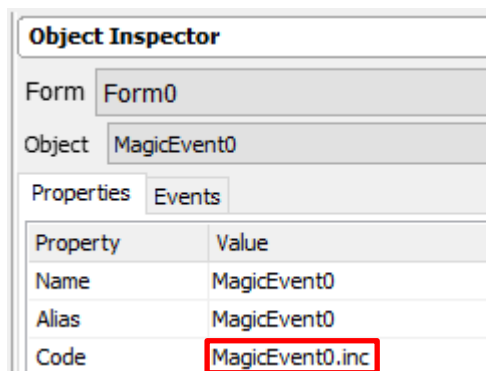
Add three Magic Events by clicking it three times.



Go to Object inspector and you must see three magic events shown by the image below.



To add or edit code of the MagicEvent function, go to their Properties and click the Code Value.



Codes Inside MagicEvent0 function. The first byte is 50, the command address for the Yellow LED, the second byte is the new value of the Button state which determines the LED state.

```

MagicEvent0.inc
1 func MagicEvent0(var newval)
2     str_PutByte(myPtr + 0, 48); // stores
3     str_PutByte(myPtr + 1, newval); // s
4     I2C1_Idle(); //wait until i2c is inac
5     I2C1_Start(); //start new communicati
6     I2C1_Write((ARDUINO_ADDRESS <<1)); //
7     I2C1_Putn(mybuffer, 2); //send the dat
8     I2C1_Stop(); //stop i2c communication
9 endfunc
  
```

Codes Inside MagicEvent1 function. The first byte is 49 command address for the Red LED, the second byte is the new value of the Button state which determines the LED state.

```

MagicEvent1.inc
1 func MagicEvent1(var newval)
2     str_PutByte(myPtr + 0, 49); // stores
3     str_PutByte(myPtr + 1, newval); // s
4     I2C1_Idle(); //wait until i2c is inac
5     I2C1_Start(); //start new communicati
6     I2C1_Write((ARDUINO_ADDRESS <<1)); //
7     I2C1_Putn(mybuffer, 2); //send 2 bytes
8     I2C1_Stop(); //stop communication
9 endfunc
  
```

Codes Inside MagicEvent2 function. The first byte is 50 command address for the Green LED, the second byte is the new value of the Button state which determines the LED state.

MagicEvent2.inc

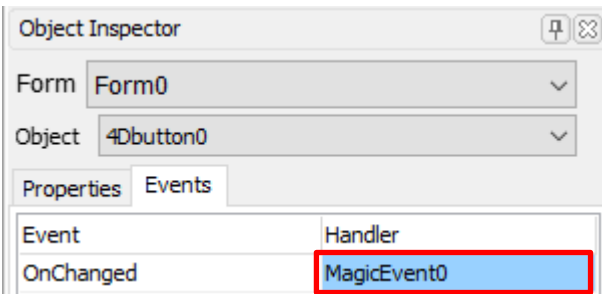
```

1 func MagicEvent2 (var newval)
2     str_PutByte (myptr + 0, 50); // stores
3     str_PutByte (myptr + 1, newval); // st
4     I2C1_Idle(); //wait until i2c is inac
5     I2C1_Start(); //start communication
6     I2C1_Write (ARDUINO_ADDRESS << 1);
7     I2C1_Putn (mybuffer, 2); //send the 2 b
8     I2C1_Stop(); //stop communication
9 endfunc

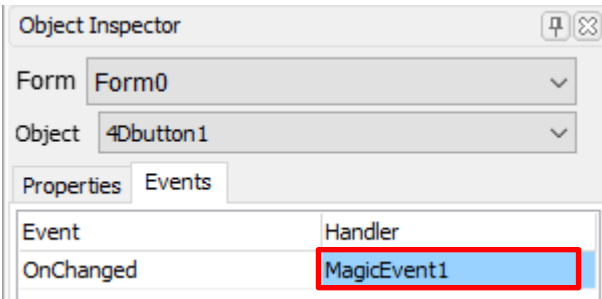
```

Add the Magic Events to the Buttons

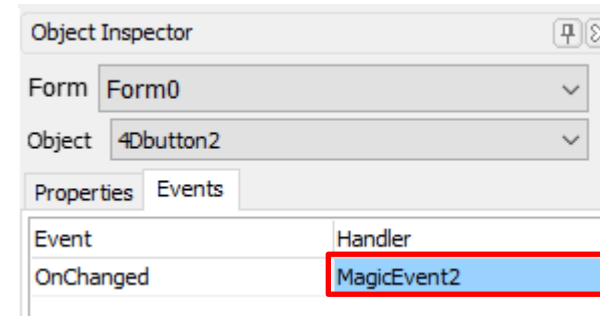
Go to 4Dbutton0 Events Tab and Set OnChanged to MagicEvent0.



Go to 4Dbutton1 Events Tab and Set OnChanged to MagicEvent1.



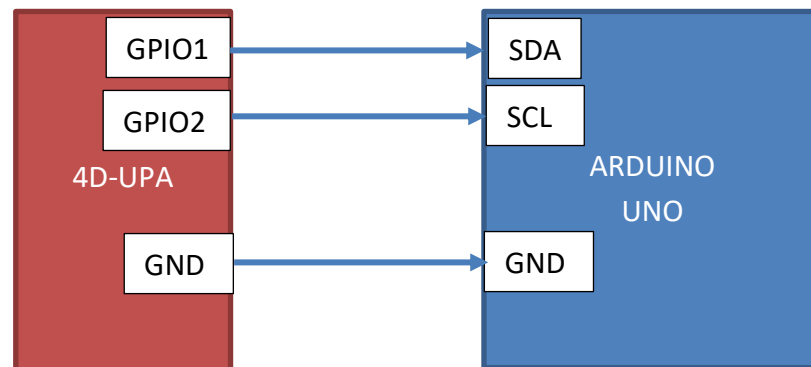
Go to 4Dbutton2 Events Tab and Set OnChanged to MagicEvent2.



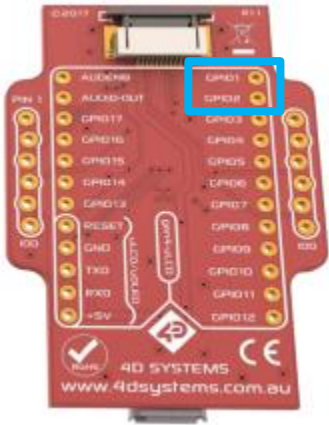
Hardware Connections

4D-UPA Breakout board to Arduino Uno I2C Connection Setup.

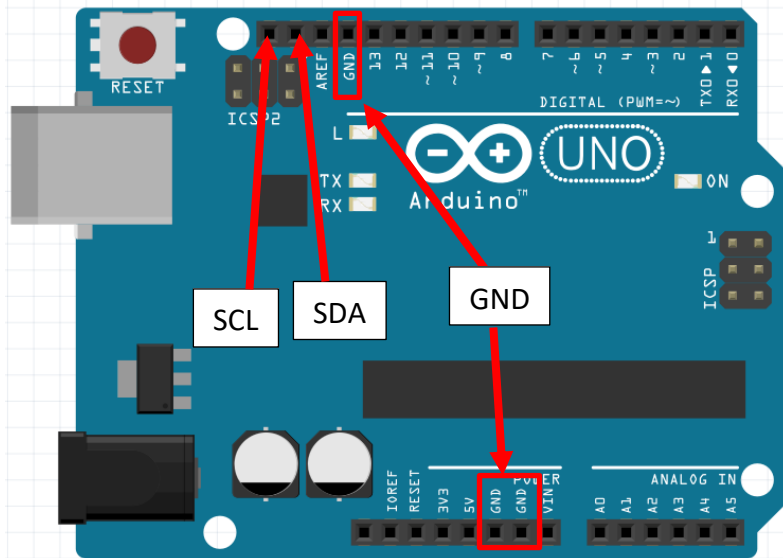
Connect GPIO1 with Arduino SDA pin, GPIO2 with Arduino SCL pin and GND with Arduino GND.



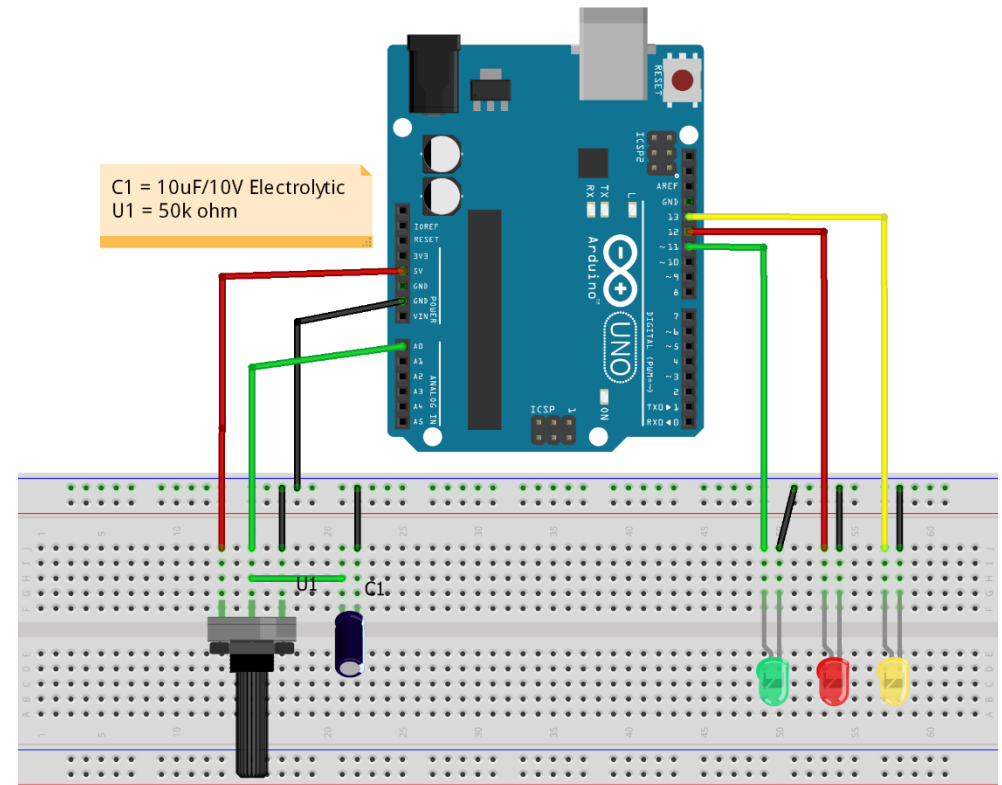
GPIO1 and GPIO2 location in the 4D-UPA board.



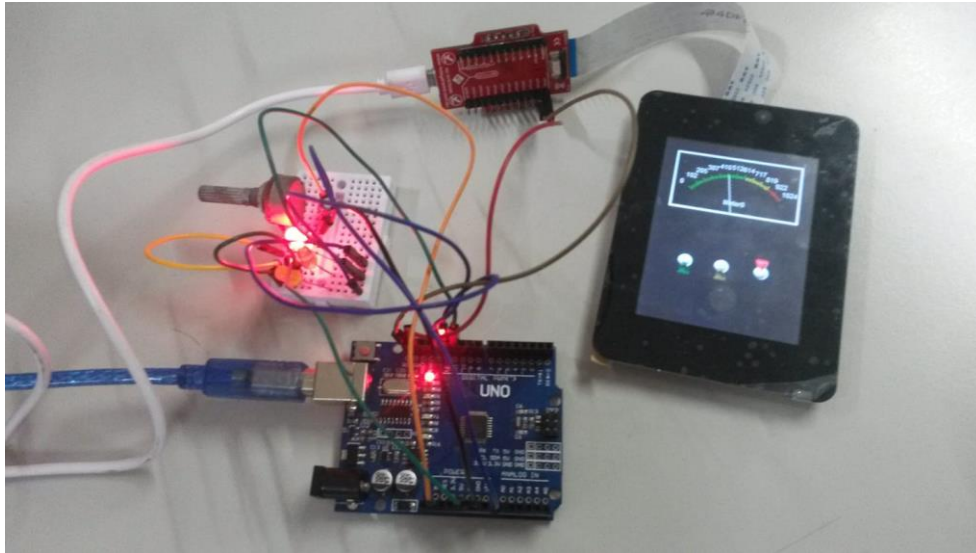
SDA SCL and GND pin in the Arduino Uno Board.



LEDs, Potentiometer and Capacitor connections in the Arduino Uno.
The purpose of the capacitor is to filter noise in the Analog input.



Before uploading the Arduino code please install the wire.h library to the Arduino. The library is included in the zip file of this Application note.



Run the Program

For instructions on how to save a **Visi-Genie** 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 any of the following application notes:

- [Visi-Genie Getting Started - First Project for Diablo16 Display Modules](#)

Proprietary Information

The information contained in this document is the property of 4D Systems Pty. Ltd. and may be the subject of patents pending or granted, and must not be copied or disclosed without prior written permission.

4D Systems endeavours to ensure that the information in this document is correct and fairly stated but does not accept liability for any error or omission. The development of 4D Systems products and services is continuous and published information may not be up to date. It is important to check the current position with 4D Systems.

All trademarks belong to their respective owners and are recognised and acknowledged.

Disclaimer of Warranties & Limitation of Liability

4D Systems makes no warranty, either expresses or implied with respect to any product, and specifically disclaims all other warranties, including, without limitation, warranties for merchantability, non-infringement and fitness for any particular purpose.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications.

In no event shall 4D Systems be liable to the buyer or to any third party for any indirect, incidental, special, consequential, punitive or exemplary damages (including without limitation lost profits, lost savings, or loss of business opportunity) arising out of or relating to any product or service provided or to be provided by 4D Systems, or the use or inability to use the same, even if 4D Systems has been advised of the possibility of such damages.

4D Systems products are not fault tolerant nor designed, manufactured or intended for use or resale as on line control equipment in hazardous environments requiring fail – safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines or weapons systems in which the failure of the product could lead directly to death, personal injury or severe physical or environmental damage ('High Risk Activities'). 4D Systems and its suppliers specifically disclaim any expressed or implied warranty of fitness for High Risk Activities.

Use of 4D Systems' products and devices in 'High Risk Activities' and in any other application is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless 4D Systems from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any 4D Systems intellectual property rights.