



ViSi-Genie Arduino: Switching Banks

DOCUMENT DATE: **9th MAY 2020**
DOCUMENT REVISION: **1.0**

WWW.4DSYSTEMS.COM.AU



Description

This application note shows how to switch between banks of the Diablo16 processor using an Arduino host.

Before getting started, the following are required:

Hardware

- Any [4D Systems display module](#) powered by the Diablo16 processor
- [Programming Adaptor for target display module](#)
- [uSD Card](#)
- [USB Card Reader](#)
- An Arduino board with at least two UART ports (one of the ports will be used for debugging)

Software

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

This application note comes with two (2) ViSi-Genie projects.

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

Content

Description	2
Content	2
Application Overview.....	3
Setup Procedure	4
Create a New Project	4
Design the Project.....	4
<i>Open the Project Files.....</i>	<i>4</i>
<i>First Project Objects.....</i>	<i>5</i>
<i>Second Project Objects</i>	<i>7</i>
<i>Upload the ViSi-Genie Programs.....</i>	<i>7</i>
Design the Arduino Sketch	8
<i>Open the Arduino Sketch</i>	<i>8</i>
<i>Install the ViSi-Genie-Arduino-Library-BETA Library</i>	<i>8</i>
Code Discussion.....	8
Bank-Specific Constants and Variables	8
Variables for knowing which Bank is Active	9
Writing to Bank-Specific Objects	9
Switching Banks	10
Recovering from a Disconnection	10
Recovering from a Bank Switch	11

Restoring States of Objects	11
Set up the Host and the Display Module	12
Proprietary Information	13
Disclaimer of Warranties & Limitation of Liability	13

Application Overview

The Diablo16 processor has six flash banks (Bank 0 to Bank 5), each of which has a capacity of 32 kB. As of Workshop4 version 4.5.0.8, it is now possible for the user to specify the destination flash bank of a ViSi-Genie program. This was not possible in previous versions of Worskhop4. Prior to version 4.5.0.8, bank 0 was the only possible flash memory destination of a ViSi-Genie program.

The purpose of this application note is to show how to switch between banks using an Arduino as a host controller. This application note uses the ViSi-Genie environment, together with Genie-Magic. Thus, the PRO version of Workshop4 is required.

For more information on the basics of the multiple flash bank feature in ViSi-Genie, refer to the application note ViSi-Genie Flash Banks.

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 Diablo16 Display Modules](#)

Create a New Project

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

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

Design the Project

For this application note, a gen4-uLCD-35DCT-CLB will be used for the project. The same procedure is applicable for any Diablo16 displays. Also, this application note comes with zip files which contain demo projects needed for the discussions.

Open the Project Files

Open the project files inside the zip files “first.zip” and “second.zip”.

Note that the projects contain magic objects, so Workshop4 PRO is needed to open them. The project “first” should contain the objects shown below. The target bank for project “first” is bank 0.



Note that these objects does not occupy the whole screen area. This is in consideration of easily testing the project with smaller displays. You may resize the project if you desire.

The project “second”, on the other hand, is shown below. The target bank for this project is bank 1.

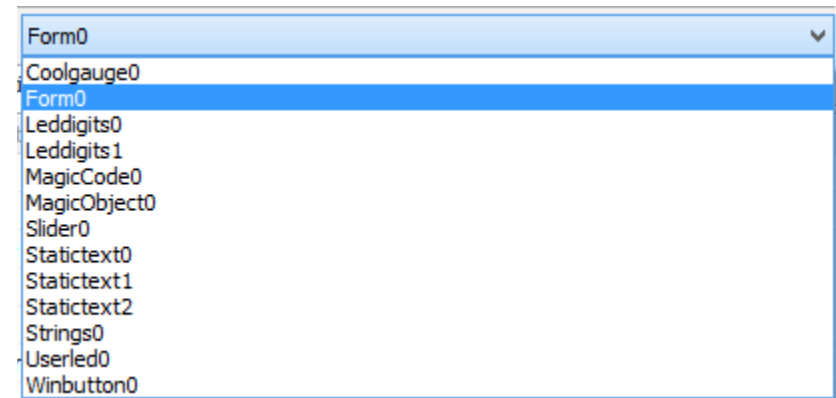


Again, please note that these objects does not occupy the whole screen area in consideration of easily testing the project with smaller displays. You may resize the project if you desire.

Both of these projects will communicate with the host at 200000 baud.

First Project Objects

The first project contains the following objects:



The input objects present in the project are Winbutton0 and Slider0.

Winbutton0 sends a REPORT_EVENT message to the host controller indicating that the user wants to switch to the second bank.

Note: It is important for applications which utilize multiple flash banks with different ViSi Genie projects that the host controller knows what flash bank the display is currently at.

If Winbutton0's REPORT_EVENT message is successfully received, the host will reply by issuing a WRITE_OBJ message to MagicObject0. MagicObject0 will then run the appropriate flash bank stated in the host's WRITE_OBJ message if valid. Otherwise, the display will reset.

```
func rMagicObject0(var action, var object, var newVal, var *ptr)

var returnFB := 0;
if(action == WRITE_OBJ)
    seroutX(ACK);
    pause(5);
    returnFB := flash_Run(newVal);

    if(returnFB)
        gfx_MoveTo(0,0);
        if(returnFB == -1)
            print("invalid bank number");
        else if(returnFB == -2)
            print("no valid program in the selected bank");
        else
            print("run bank not successful, unknown error");
        endif

        print("\nrestarting...");
        pause(2000);
        SystemReset();
    endif
else if(action == READ_OBJ)
    SendReport(REPORT_OBJ, tMagicObject, 0,flash_Bank()) ;
    // let the host know the current bank
endif
endfunc
```

As seen in the code above, MagicObject0 can also receive a READ_OBJ message, and it will send a REPORT_OBJ message stating the current flash bank that is running.

```
else if(action == READ_OBJ)
    SendReport(REPORT_OBJ, tMagicObject, 0,flash_Bank()) ;
    // let the host know the current bank
```

The object MagicCode0 contains the line below.

```
SendReport(REPORT_EVENT, tMagicObject, 0, flash_Bank()) ;
```

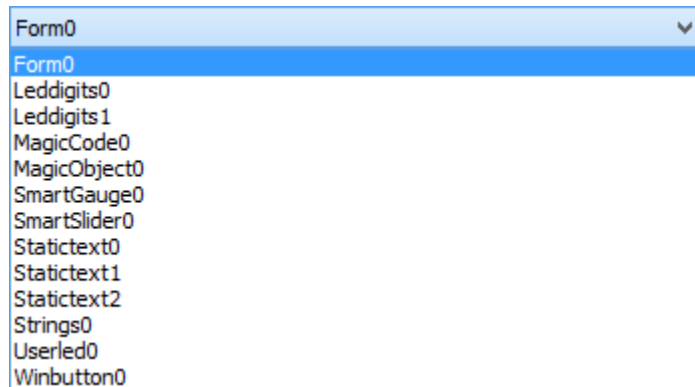
MagicCode0 is inserted to the Post Genie Initialize section of the ViSi-Genie program. This sends a REPORT_EVENT message to the host that the display is ready and what bank it is currently running.

Slider0 simply sends a REPORT_EVENT message to the host. Once the host receives this message, it will update Leddigits0 accordingly.

The other remaining objects act as outputs that will be controlled by the host.

Second Project Objects

The second project contains the following objects:



It can be noticed that this project is very similar to the previous one. SmartSlider0 and SmartGauge0 simply replaced Slider0 and Coolgauge0 of the previous project, respectively.

SmartSlider0 in the second project and Slider0 in the first project have the same function. The same applies to SmartGauge0 and Coolgauge0.

Winbutton0's function is identical to that of Winbutton0 in the first project.

MagicCode0 and MagicObject0 are also present and have the same functions as described in the previous section.

The other remaining objects, similar to the previous project, will act as outputs that will be controlled by the host.

Upload the ViSi-Genie Programs

As discussed in the application note [ViSi-Genie Flash Banks](#), there are two ways to load a program to its target bank. The first one is to directly load the program to the target flash bank. The second option is to copy the program file to the uSD card which is directly mounted to the PC. The uSD card can then be unmounted from the PC and mounted to the display module. Another program is then needed to copy the program from the uSD card to the destination flash bank. In this example, the first approach is used. It is also possible to use the second approach. For more information, refer to the application note linked above.

Compile the projects and upload the programs to their respective flash banks. Also, the GCI and DAT files need to be copied to the uSD Card. The first project should occupy Bank0 while the second project should occupy Bank1. Again, it is also possible to utilize the other banks. For more information, refer to the application note [ViSi-Genie Flash Banks](#).

Design the Arduino Sketch

Open the Arduino Sketch

The sketch for this application note is `switchBankDemo_R_x_yy.ino`, where “**x_yy**” is the revision number. Look for the sketch file in the attached zip file.

Install the ViSi-Genie-Arduino-Library-BETA Library

The library used in the sketch is found [here](http://arduino.cc/en/Guide/Libraries). For more information on how to properly install libraries in the Arduino IDE, visit this link:

<http://arduino.cc/en/Guide/Libraries>

First-time users of the above library are encouraged to study the default examples in the library. The following discussions assume that the reader has a working knowledge of the Arduino platform and has an understanding of the examples in the library.

Code Discussion

Bank-Specific Constants and Variables

As emphasized in the application note ViSi-Genie Switching Banks, it is important for the host to be aware which bank is currently running on the display module. This is so that the host writes only to the objects that exist on the program that is currently running. In the source code of the host's

program therefore, it is useful to organize bank-specific variables. For this example, the *struct bank* is defined as follows.

```
struct bank {
    uint8_t gauge;
    uint16_t gaugeVal;
    int8_t gaugeDir;
    uint8_t input;
    uint16_t inputVal;
};
```

The variables *Project1* and *Project2*, each of the data type *bank*, are then declared.

```
bank Project1;
bank Project2;
```

In *setup()*, the variables exclusive to each ViSi-Genie project are assigned as members of *bank Project1* and *Project2*, accordingly.

```
setBank(&Project1, GENIE_OBJ_COOL_GAUGE, 0, 1, GENIE_OBJ_SLIDER, 0);
setBank(&Project2, GENIE_OBJ_ISMARTGAUGE, 0, 1, GENIE_OBJ_ISMARTSLIDER, 0);
```

The function *setBank()* is defined before the setup routine.

```
void setBank(bank* _bank, uint8_t gauge, uint16_t gaugeVal, int8_t gaugeDir, uint8_t input, uint16_t inputVal) {
    _bank->gauge = gauge;
    _bank->gaugeVal = gaugeVal;
    _bank->gaugeDir = gaugeDir;
    _bank->input = input;
    _bank->inputVal = inputVal;
}
```


Variables for knowing which Bank is Active

For monitoring the current bank, a global *bank* pointer and a global signed 8-bit integer are declared and initialized to invalid values.

```
bank* currentProject = NULL;
int8_t currentBank   = -1;
```

After receiving either a REPORT_EVENT or REPORT_OBJ message from the display module, the host will be able to assign the proper values for *currentProject* and *currentBank*. To illustrate, refer to the code blocks shown below. These blocks are inside the *myGenieEventHandler()* routine.

```
if (Event.reportObject.cmd == GENIE_REPORT_EVENT) {
    ...
    else if (Event.reportObject.object == GENIE_OBJ_MAGIC) {
        if (Event.reportObject.index == 0) {
            Serial.println("Received REPORT_EVENT message from
MagicObject0");
            receivedReportEventFlag = true;
            // Received when the program is ready (Post Genie Init)
            currentBank = genie.GetEventData(&Event);
            currentProject = (currentBank == PROJECT1) ? &Project1 :
&Project2;
        }
    }
}
```

```
else if (Event.reportObject.cmd == GENIE_REPORT_OBJ) {
    if (Event.reportObject.object == GENIE_OBJ_MAGIC) {
        if (Event.reportObject.index == 0) {
            Serial.println("Received REPORT_OBJ message from
MagicObject0");
            receivedReportObjFlag = true;
            // Received as a reply to a READ_OBJ request, when the program
is not ready
            currentBank = genie.GetEventData(&Event);
            currentProject = (currentBank == PROJECT1) ? &Project1 :
&Project2;
        }
    }
}
```

Writing to Bank-Specific Objects

In the main loop it is shown how the host writes to the gauge object of the current bank.

```
if (currentBank != -1) {
    currentProject->gaugeVal += currentProject->gaugeDir;
    if (currentProject->gaugeVal == 99 || !currentProject->gaugeVal)
currentProject->gaugeDir *= -1;
    genie.WriteObject(currentProject->gauge,0,currentProject-
>gaugeVal);
    genie.WriteObject(GENIE_OBJ_USER_LED,0,currentProject->gaugeVal
% 2);
}
```

Note that the host writes to the display module only if the value of *currentBank* is valid.

```
if (currentBank != -1) {
    currentProject->gaugeVal += currentProject->gaugeDir;
    if (currentProject->gaugeVal == 99 || !currentProject->gaugeVal)
currentProject->gaugeDir *= -1;
    genie.WriteObject(currentProject->gauge,0,currentProject-
>gaugeVal);
    genie.WriteObject(GENIE_OBJ_USER_LED,0,currentProject->gaugeVal
% 2);
}
```

Otherwise, the host sends a READ_OBJ message to the display module to poll the number of the bank that is currently running.

```
else {
    genie.ReadObject(GENIE_OBJ_MAGIC, 0);
    Serial.println("polling bank of the display now!");
}
```

The display module replies to the READ_OBJ message with a REPORT_OBJ message, which will be handled by the function *myGenieEventHandler()*, as shown earlier.

Switching Banks

When Winbutton0 of either bank 0 or 1 is pressed, the display module sends a REPORT_EVENT message to the host. This REPORT_EVENT message originating from Winbutton0 is then processed in the function *myGenieEventHandler()*.

```
if (Event.reportObject.cmd == GENIE_REPORT_EVENT) {  
    ...  
    else if (Event.reportObject.object == GENIE_OBJ_WINBUTTON) {  
        if (Event.reportObject.index == 0) {  
            Serial.println("switching bank now");  
            genie.WriteObject(GENIE_OBJ_MAGIC, 0, (currentBank ==  
PROJECT1) ? PROJECT2 : PROJECT1);  
            currentBank = -1;  
            currentProject = NULL;  
            switchBankFlag = true;  
        }  
    }  
}
```

As can be seen in the code snippet, the REPORT_EVENT message from Winbutton0 causes the host to send back a WRITE_OBJ message to the display module, causing it to switch to the other bank. Also, the variables for monitoring the currently active bank are equated to invalid values. This means that the host must have to poll the display module again for the number of the currently active bank in the next iteration of the main loop. The host can also use the content of the REPORT_EVENT message coming from the display every time it restarts.

Recovering from a Disconnection

The library used in this example has a facility for knowing if the display module is disconnected from the host. A disconnection can be caused by the display module being power cycled, by the UART lines being cut off, etc. Every time that the display module is disconnected from the host, a GENIE_DISCONNECTED event is raised and is processed by the function *myGenieEventHandler()*.

```
else if (Event.reportObject.cmd == GENIE_PING) {  
    switch (Event.reportObject.object) {  
        case GENIE_DISCONNECTED:  
            digitalWrite(13, HIGH);  
            currentBank = -1;  
            Serial.println("\nDisplay disconnected!\n");  
            break;  
        ...  
        default:  
            break;  
    }  
}
```

As can be seen in the code snippet above, the variable *currentBank* is assigned an invalid value when a disconnection happens. Again, this causes the host to query the display module for the number of the current bank in the next iterations of the main loop, assuming that the display module is connected back again to the host.

Recovering from a Bank Switch

Note that when switching banks, the display actually resets and boots up. During this reset-and-boot-up period, the display module is disconnected from the host. Hence, a bank switch can be thought of as another cause of a disconnection. To differentiate a bank switch from other causes of a disconnection (such as the display module being inadvertently power cycled or the UART lines being cut off), the variable *switchBankFlag* is used. When sending a WRITE_OBJ message to the display module to cause it to switch banks, the host also sets the value of the flag *switchBankFlag* to *true*.

```
if (Event.reportObject.cmd == GENIE_REPORT_EVENT) {  
    ...  
    else if (Event.reportObject.object == GENIE_OBJ_WINBUTTON) {  
        if (Event.reportObject.index == 0) {  
            Serial.println("switching bank now");  
            genie.WriteObject(GENIE_OBJ_MAGIC, 0, (currentBank ==  
PROJECT1) ? PROJECT2 : PROJECT1);  
            currentBank = -1;  
            currentProject = NULL;  
            switchBankFlag = true;  
        }  
    }  
}
```

Restoring States of Objects

After a bank switch or a disconnection, the display module might send a REPORT_EVENT message containing the number of the active bank (if power cycled). In the event that the REPORT_EVENT message is not sent or if the host is not able to receive and process it, the host polls the display module until it receives a REPORT_OBJ message containing the number of the active bank. After the host is able to determine the number of the current bank, it now restores the objects to their last known states accordingly. The restoration process is performed by the function *updateBankObjects()*, which is called from the main loop.

```
void loop() {  
    static long waitPeriod = millis(); // timer to repeat task  
    genie.DoEvents();  
  
    if (millis() >= waitPeriod) {  
        ...  
    }  
  
    updateBankObjects(); //update bank objects and states of bank  
    switch flags  
}
```

The definition of this function is found at the end of the sketch. Note that this function is able to differentiate between disconnections caused by a bank switch from those caused by other cases.

Set up the Host and the Display Module

Refer to the section *“Connect the Display Module to the Arduino Host”* of the application note [ViSi-Genie Connecting a 4D Display to an Arduino Host](#) for more information on how to connect a display module to an Arduino host.

It is highly recommended to use an Arduino host with at least two hardware UART ports during development. Serial0 can be used for debugging purposes while Serial1 can be used for communicating with the display module. Users can modify the attached sketch accordingly to remove the debugging lines.

Note: The library used in this application note, at the time of writing, does not support software serial.

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.