APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi 6 Channel PWM

DOCUMENT DATE: **7th MAY 2020**
DOCUMENT REVISION: **1.2**

Description

This application note is intended to demonstrating to the user the set-up, initialization and operation of the built-in pulse width modulation feature of the Diablo16 display module.

- Any of the following 4D Diablo 16 touch display 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/products 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)
- micro-SD (µSD) memory card
- Common Cathode R-G-B LED
- Workshop 4 IDE (installed according to the installation document)
- 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

This document is focused on the fundamental usage of the basic use of the built-in pulse width modulation feature of the DIABLO16 Graphics Processor. The projects includes 6 sliders that changes the width of the output pulse. The pulse width values are then applied to RGB LEDs connected on several PWM-capable IO-pins.

Setup Procedure

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

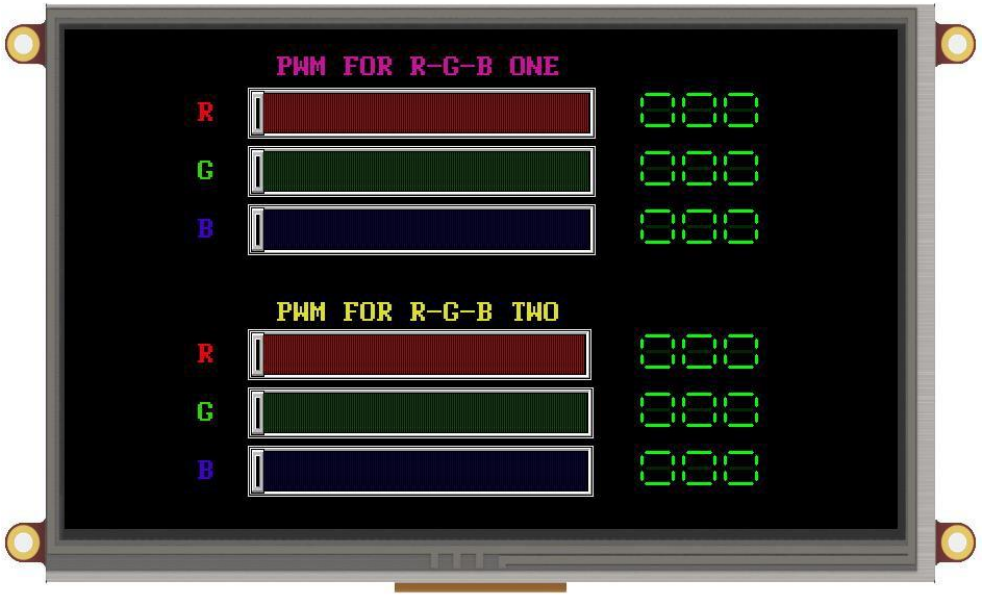
Create a New Project

For instructions on how to create a new **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

For this application project a couple of sliders and custom LED digits will be needed. Add objects by navigating to the Layout the objects similar to the one below.



After all the objects have been laid-out, let's continue with the other half which involves the coding of the project. This will be presented in a sectional manner so as not to create confusion with the project. For an in-depth detail of the functions used in this application note please refer to the Diablo16 Internal Functions Reference Manual.

The Include Section

This project starts with the identification of the platform being used as declared by the #platform function. For the program to be able to function properly files are included herein using the #inherit function.

quad_in

```
1  #platform "uLCD-70DT"
2  #inherit "4DGL_16bitColours.fnc"
3  #inherit "VisualConst.inc"
4  #inherit "quad_inConst.inc"
5  #inherit "leddigitsdisplay.inc"
```

In this application note, quad_inConst.inc, contains all the information about the objects that are used in the project. Meanwhile, the leddigitsdisplay.inc contains the function for the proper operation of the led digits objects.

The Main Program

The main program for this projects contains several sections: the mounting of the micro-SD card, the initial displaying and image touch setup for objects, the setup for quadrature and input-output control, the repeat-forever loops which contains the continuous registry reading and touch conditions. Also, the main program calls out sub-routine functions that perform a particular functions.

The micro-SD Card Initialization

Let's start with the initialization of the uSD card. The uSD card contains all the image information about the objects used in the project. The object information and data are saved under a *.DAT and a *.GCI filename extension which is copied to the uSD during project compilation. Mounting of the disk in this application note was done using the following set of program statements.

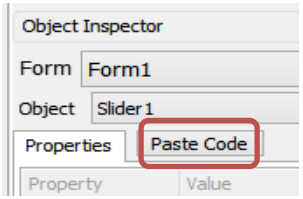
```
52      putstr("Mounting...\n");
53      if (!(disk:=file_Mount()))
54          while(!(disk :=file_Mount()))
55              putstr("Drive not mounted...");
56              pause(200);
57              gfx_Cls();
58              pause(200);
59          wend
60      endif
61
62      gfx_Set(SCREEN_MODE,LANDSCAPE) ;
63      gfx_Cls();
64
65      hndl := file_LoadImageControl("quad_in.dat", "quad_in.gci", 1);
66      indicate := file_LoadImageControl("direct.dat", "direct.gci", 1);
67      indicate1:= file_LoadImageControl("direct1.dat", "direct1.gci",1);
68      stop := file_LoadImageControl("stop.dat","stop.gci",1);
```

When starting a new project in the ViSi environment these set of statements are already included in the coding area. The last part of this set of statements uses a function file_LoadImageControl() to call on the object data/information files on the uSD drive. This initializes the data to be called in using the variable 'hndl'. Likewise, additional variable were also assigned as image control for some graphics composer generated GCI and DAT files.

Having been able to load and initialize the uSD drive, the processor is now able to access the information stored therein. As mentioned from the

previous section, the filenames with an extension of DAT and GCI has the image data and information.

Therefore, the next part of the main program is to display all the objects that were placed on the Workshop IDE form viewer. To do so, a special button from the Object Inspector can help reduce the time of coding of this part. The 'Paste Code" simply pastes object code into the coding area.



Initial Image Display and Image Touch Setup

In this part of the program, the img_Show() function calls out the object image and information found in the microSD drive. This set of statements displays every object that were included in the application project. The displaying of the images is directly done using the img_Show() function.

```
288     img_Show(hndl, iLeddigits1);
289     img_Show(hndl, iLeddigits2);
290     img_Show(hndl, iLeddigits3);
291     img_Show(hndl, iLeddigits4);
292     img_Show(hndl, iLeddigits5);
293     img_Show(hndl, iLeddigits6);
294     img_Show(hndl,iSlider1) ;
295     img_Show(hndl,iSlider2) ;
296     img_Show(hndl,iSlider3) ;
297     img_Show(hndl,iSlider4) ;
298     img_Show(hndl,iSlider5) ;
299     img_Show(hndl,iSlider6) ;
```

In this segment of the program statements. We have displayed all the slider and LED digit images that were used to provide the value for the width of the modulated pulse. Also included in this segment of the main program is the displaying of the labels for our objects.

```
301     name1();                    // LABEL FOR PWM ONE
302     name2();                    // LABEL FOR PWM TWO
303     name3();                    // PWM ONE RED
304     name4();                    // PWM ONE GREEN
305     name5();                    // PWM ONE BLUE
306     name6();                    // PWM TWO RED
307     name7();                    // PWM TWO GREEN
308     name8();                    // PWM TWO BLUE
```

Moving to the next part of the main program, this segment is all related to the image touch detection setup.

```
img_SetWord(hndl, iSlider1, IMAGE_FLAGS, (img_GetWord(hndl, iSlider1, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);
img_SetWord(hndl, iSlider2, IMAGE_FLAGS, (img_GetWord(hndl, iSlider2, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);
img_SetWord(hndl, iSlider3, IMAGE_FLAGS, (img_GetWord(hndl, iSlider3, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);
img_SetWord(hndl, iSlider4, IMAGE_FLAGS, (img_GetWord(hndl, iSlider4, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);
img_SetWord(hndl, iSlider5, IMAGE_FLAGS, (img_GetWord(hndl, iSlider4, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);
img_SetWord(hndl, iSlider6, IMAGE_FLAGS, (img_GetWord(hndl, iSlider4, IMAGE_FLAGS) | I_STAYONTOP) & ~I_TOUCH_DISABLE);

    touch_Set(TOUCH_ENABLE);                    // enable the touch screen
```

This statements uses the img_SetWord() function. The primary objective of this set of statement is to enable the touch detection for the slider image. The slider images on this project serves as an input objects which utilizes the touch feature of the device. At the end, of this segment we would notice that the touch feature of the device was enabled using the touch_Set(TOUCH_ENABLE) statement.

The Repeat-forever Touch Detection Loop

At this end part of the main program, the routine was to detect any activity on the touch screen. Please refer to the image on the next page. Three touch states were included in the repetitive routine: the detection for a pressed

state, a released state, and a moving state. Prior to the touch detection, a variable ‘n’ is assigned to store temporary image touch detection result. The img_Touched() function checks the object being touched and return the name of the object enlisted in the variable ‘hndl

```
320         state := touch_Get(TOUCH_STATUS);       // get touchscreen status
321         n := img_Touched(hndl,-1) ;
```

Moving to the touch detection routines, when a touch status of ‘pressed’ is detected the value of the coordinates are saved on the variables x and y.

```
324         if(state == TOUCH_PRESSED)              // if there's a press
325             x := touch_Get(TOUCH_GETX);
326             y := touch_Get(TOUCH_GETY);
327         endif
```

The most significant segment of this routine is the moving touch state, it is in this conditional loop that the image touch detection is made use. If a touch was detected over the slider image, a sub-routine or a function is called upon and executed by the processor.

```
        state := touch_Get(TOUCH_STATUS);       // get touchscreen status
        n := img_Touched(hndl,-1) ;
```

Let us take the above statement as an example. From the start of the repeat-forever loop, the img_Touched() function saves the result of an image touch to a variable ‘n’, this is then checked in the touch moving conditional statements. If it proves to be equal to one of the conditions then the sub-routine will be executed. For this statement a moveSlider1() sub-routine is being called and executed. Referring to the image following this section, we

The moveSlider Sub-routine

In the earlier part of this application note it was mentioned that the PWM function will be used in the sub-routines of the main program. This sub-routines are named moveSlider and appended with their respective numbers. This would mean that when an image Slider1 object is touched then this will result to a call and execute of a moveSlider1() sub-routine.

```
11  func moveSlider1()
12      var posn;
13      // Slider1 1.0 generated 9/1/2013 10:07:20 AM
14      img_Show(hndl,iSlider1) ;                                   //
15      img_ClearAttributes(hndl, iSlider1, I_TOUCH_DISABLE);       //
16      posn := x - 168 ;                                           //
17      if (posn < 0)
18          posn := 0 ;
19      else if (posn > 317)                                        //
20          posn := 100 ;                                           //
21      else
22          posn := 100 * posn / 317 ;                              //
23      endif
24      img_SetWord(hndl, iSlider1, IMAGE_INDEX, posn);
25      img_Show(hndl, iSlider1);
26
27      // Leddigits1 1.0 generated 9/1/2013 9:33:48 AM
28      img_Show(hndl, iLeddigits1);                                //
29      ledDigitsDisplay(posn, iLeddigits1+1, 552, 3, 1, 40, 0) ;
30
31      pwm_Init(PA4,PWM_PLAIN,posn * 10);
32
33  endfunc
```

The first part of the sub-routine includes the slider codes that are automatically generated using the ‘Paste Code’ tab on the object inspector window. This is also true with the LED digits objects. The paste code function is a very good tool to minimize the coding of the objects.

```
13      // Slider1 1.0 generated 9/1/2013 10:07:20 AM
14      img_Show(hndl,iSlider1) ;                              // show initialy,
15      img_ClearAttributes(hndl, iSlider1, I_TOUCH_DISABLE);  // set to enable t
16      posn := x - 168 ;                                      // x - left - 8
17      if (posn < 0)
18          posn := 0 ;
19      else if (posn > 317)                                   // width - 17)
20          posn := 100 ;                                      // maxvalue-minval
21      else
22          posn := 100 * posn / 317 ;                         // max-min - (max-
23      endif
24      img_SetWord(hndl, iSlider1, IMAGE_INDEX, posn);
25      img_Show(hndl, iSlider1);
```

The last line of the sub-routine includes the pwm_Init(). There are no particular assignment or hierarchy for this function.

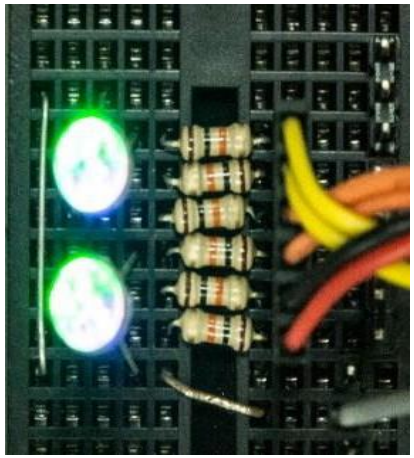
```
31      pwm_Init(PA4,PWM_PLAIN,posn * 10);
```

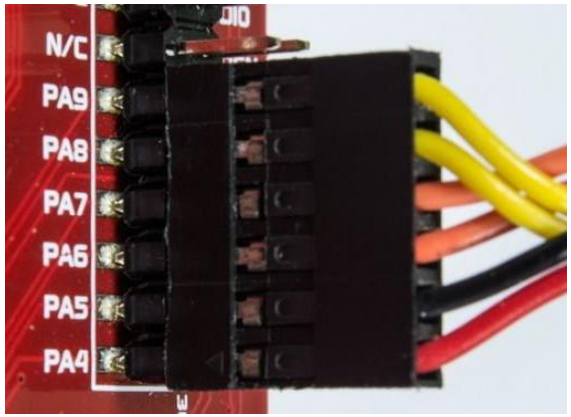
At any time, this function will be able to initialize the GPIO with that has a PWM output function. The initialization and value are contained in the arguments section of the function. The first argument is the GPIO pin number, followed by a mode of PWM and lastly the value of the PWM duty cycle or percent width modulation. The width of the modulation operates on a 0 % to 100 %. But the numerical value that is acceptable in the function ranges from 0 up to 1000. This was done so as to enable precision of pulse width modulation. For this reason, the output of the slider object was multiplied with a factor of ten during pwm_Init() function call.

Run the Project

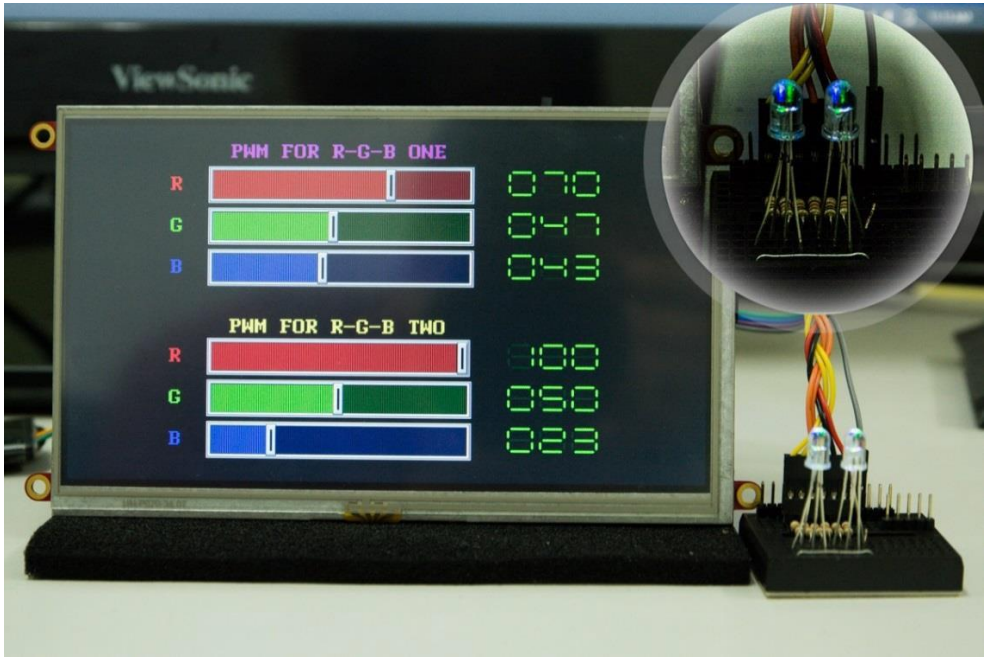
After building and downloading to the display module the R-G-B LED can be directly wired into the 30-way header pins. The connection to the display module can be reference from the figure on the images below. The ground pin of the common cathode RGB LED must be connected to the ground pin of the display module to complete the circuit. Also, understand that the GPIO in the display device are a set of 3.3 volt TTL pins.

The pulse width modulation feature of these devices can be utilized in various range of application. However, the requirement of the output load must be considered. If the output load shall require a higher current and a higher voltage rating then additional components – such as transistors or opto-couplers for switching. These PWM signal can be fed into a transistor's base to turn off/on the transistor.





Referring to the image below. This image shows the output of the application project. As the sliders are moved the value of the modulation for each colour of the RGB LED is shown on the right. Hence, changing the slider value would mean a change in the intensity of the colour.



The PWM feature of the DIABLO16 can be used to provide a signal for mini DC servo motors, and to normal DC motor with additional circuitry.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability