APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Data Loop Back Testing of Serial Hardware

DOCUMENT DATE: **15th APRIL 2019**
DOCUMENT REVISION: **1.1**

Description

This Application note is intended to demonstrating to the user the set-up, initialization and operation of the built-in serial communications port of the Diablo16 display module.

Before getting started, the following are required:

- The target module can also be a Diablo16 display

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. The display module used in this application note is the uLCD-32PTU, which is a Picaso display. This application note is applicable to Diablo16 display modules as well.

- 4D Programming Cable / uUSB-PA5/uUSB-PA5-II
  for non-gen4 displays(uLCD-xxx)
- 4D Programming Cable & gen4-PA, / gen4-IB / 4D-UPA
  for non-gen4 displays(uLCD-xxx)
- 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.
- This application note requires that the reader has a basic knowledge of any programming language such as C.

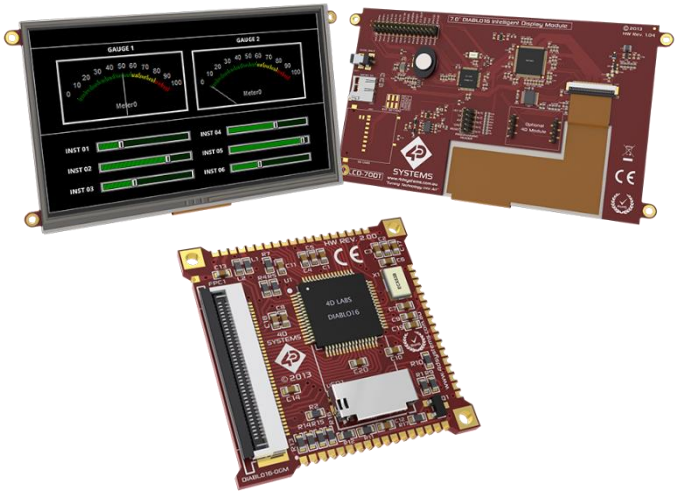
Content

Application Overview

This document is about basic asynchronous serial hardware loopback. It uses the four serial communication ports that are available to the Diablo16 OGM and Diablo16 display module. The primary objective of this application note is to demonstrate to the user the ease of serial hardware setup, initialization, and usage with the Diablo16 display module. Using the available serial port of this display module, the user will be able to interconnect with several other devices.

The availability of several data rate speeds for these serial communication ports allows the user to choose their desired transfer speed according to their need. The range of baud rate transfer is from a minimum of 300 baud up to a 600K baud.

The DIABLO16 Embedded Graphics Processor



Driving the display and peripherals is DIABLO16 embedded graphics processor, a very capable and powerful chip which enables stand-alone functionality, programmed using the 4D Systems Workshop 4 IDE Software. The Workshop IDE enables graphic solutions to be constructed rapidly and with ease due to its design being solely for 4D's graphics processors.

The DIABLO16 Processor offers considerable FLASH and RAM upgrades over the PICASO processor, and also provides map-able functions such as I2C, SPI, Serial, PWM, Pulse Out, and Quadrature Input, to various GPIO, and also provide up to 4 Analogue Input channels.

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

The DIABLO16 Serial Com Ports

To create a simple program that will be able activate and initialize the DIABLO16 serial ports, we will need to use some commands enlisted in the DIABLO 4DGL Internal Functions.

In particular, we will be using the Serial (UART) Communications Functions. The DIABLO16 serial communication ports uses only the Transmit(TX), Receive(RX) and ground pins of the standard serial communication port, it is relative easy to use. Before going further it is good to remember that the DIABLO16 display module has four serial communications port which can be simultaneously used to communicate with other serial capable controllers or devices.

There are several important settings of a serial communications that are very important to have an effective and correct data communications. First thing to consider is the BAUD RATE, the baud rate is simply the rate of transfer from and to the device. If there exist a mismatch in the baud rate of two serially coupled devices – the serial communication will be erroneous.

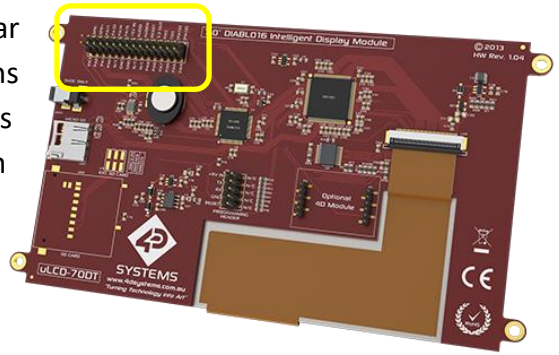
Secondly, it is also good to consider the number of bytes that shall be transferred at the instance of communication. This is important so that we display device will be programmed to have a particular buffer which is capable to save a series of bytes continuously. This size of the temporary data storage, which is termed as a buffer, will determine the size of the data that can be stored. For this application, we will be utilizing these communication ports to update the value of a Workshop IDE object built under the ViSi environment. The objects include an input object (i.e. slider) will provide the data that for the output objects (i.e. cool gauge and custom LED digits).

For this loopback test, we will have to attain the following: setup and initialization of the map-able serial communication ports. Without being able to achieve the aforementioned objective, the application will not be able achieve its purpose.

The map-able Serial communication ports

The DIABLO16 embedded graphics processor allows the user to assign the general I/O pin to be used for each ports. In the former sections of this application note, it was mentioned that the DIABLO16 has a total of four serial communication ports. Of this four serial com ports, Com0 is the only one used to programming the DIABLO16 processor. Also, the Com0 is assigned on fixed pin assignment while the others are capable of being mapped into a field of I/O pins.

These I/O pins are located similar to the image. This group of pins includes the 14 general I/O pins which are configurable to perform certain functions depending on the user preferences.



The serial communication port can be assigned to a pin according to the specifications of the DIABLO16 Datasheet. Referring to the image below, we can see that a particular serial transmit or receive pin can only be assigned to a particular pin number. This pin may be common to all the other three serial com ports or exclusive to itself.

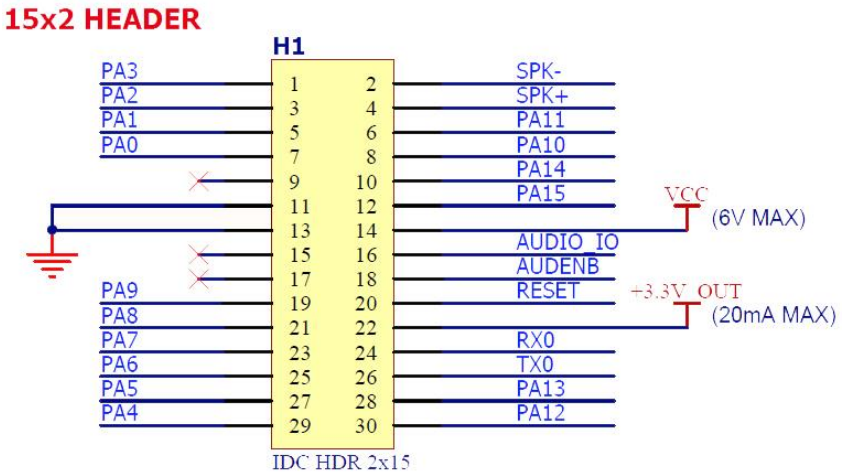
DIABLO16 Serial TTL Comm Port Configuration Options						
	TX1	RX1	TX2	RX2	TX3	RX3
PA0		✓		✓		✓
PA1	✓	✓	✓	✓	✓	✓
PA2		✓		✓		✓
PA3	✓	✓	✓	✓	✓	✓
PA4	✓	✓	✓	✓	✓	✓
PA5	✓	✓	✓	✓	✓	✓
PA6	✓	✓	✓	✓	✓	✓
PA7	✓	✓	✓	✓	✓	✓
PA8	✓	✓	✓	✓	✓	✓
PA9	✓	✓	✓	✓	✓	✓
PA10		✓		✓		✓
PA11		✓		✓		✓
PA12	✓	✓	✓	✓	✓	✓
PA13	✓	✓	✓	✓	✓	✓
PA14						
PA15						

A serial port can only be assigned with a single transmit or receive terminal. This means that a particular serial port cannot have two receive or transmit terminal. This port is restricted to follow the normal convention for serial communication port.

A general I/O pin can only be configured to a particular function. Overlapping assignment of pin functions can cause erroneous data reception and transmission.

The DIABLO16 configurable I/O are a group of 3.3 volts TTL level terminal but these are tolerant to a maximum of 5 volts. Anything greater than or less than the specified operating voltage may prohibit proper communication or even damage the embedded graphics processor.

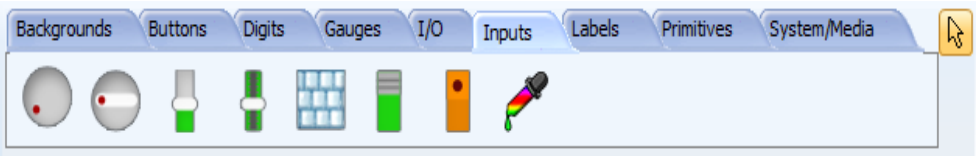
The 2x15 male header pin assignment of the DIABLO16 70DT. The previous table on serial com port configuration option are lumped together with several other specific purpose pins.



Design the Project

This application projects involves the use of an input slider together with a custom LED digits to display its current value. On the other hand to display the received value at the receiving end of the serial com port, a cool gauge as a display.

Navigate through the widgets menu of the Workshop 4 IDE and add a slider, a custom LED digit or a simple LED DIGIT and a cool gauge object to the project. Create the project to be 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.

```
1  #platform "uLCD-70DT"
2  #inherit "4DGL_16bitColours.fnc"
3  #inherit "D3001_Serial_Hardware_LoopbackConst.inc"
4  #inherit "leddigitsdisplay.inc"
```

In this application note, the D3001_Serial_Hardware_LoopbackConst.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 initialization and setup of serial communication ports, the mounting of the micro-SD card, the initial displaying and image touch setup for objects, and lastly, the touch detection loop.

The Serial communications port setup and initialization

The following statement is the pin assignment for the serial communication ports. It can be noticed that no assignment involving the COM0 was done.

The COM0 of this display device is fixed. The only setting that can apply to this communication port is the buffer, qualifier identification and the baud rate setting. This port has its own designated pin on the H1 header pin group. In this group of statements the communication ports pin assignment were made adjacent to the ports TX and RX so that putting up a jumper will be easy.

```
138     COM1_RX_pin(PA4);       // select the hardware pin for COM1 receive line
139     COM1_TX_pin(PA5);       // select the hardware pin for COM1 transmit line
140
141     COM2_RX_pin(PA6);       // select the hardware pin for COM1 receive line
142     COM2_TX_pin(PA7);       // select the hardware pin for COM1 transmit line
143
144     COM3_RX_pin(PA8);       // select the hardware pin for COM1 receive line
145     COM3_TX_pin(PA9);       // select the hardware pin for COM1 transmit line
146
147     com_Init(buffer0,5,0);
148     com1_Init(buffer0,5,0);
149     com2_Init(buffer0,5,0);
150     com3_Init(buffer0,5,0);
151     com_SetBaud(COM0, 11500);
152     com_SetBaud(COM1, 11500);
153     com_SetBaud(COM2, 11500);
154     com_SetBaud(COM3, 11500);
```

The micro-SD initialization

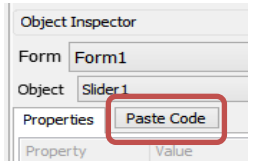
Let's start with the initialization of the uSD card. The uSD card contains all the image information about the objects used in the project. The object information and data are saved under a *.DAT and a *.GCI filename extension which is copied to the uSD during project compilation. Mounting of the disk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

```
123     var state, n ;
124
125     putstr("Mounting...\n");
126     if (!(disk:=file_Mount()))
127         while(!(disk :=file_Mount()))
128             putstr("Drive not mounted...");
129             pause(200);
130             gfx_Cls();
131             pause(200);
132         wend
133     endif
134     gfx_TransparentColour(0x0020);
135     gfx_Transparency(ON);
136
137     hndl := file_LoadImageControl("D3001_Se.dat", "D3001_Se.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'.

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.



The initial image display and image touch setup segment

In this part of the program, the img_Show() function simply 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.

```
139     img_Show(hndl, iCustomdigits1);
140     img_Show(hndl, iCustomdigits2);
141     img_Show(hndl, iCustomdigits3);
142     img_Show(hndl, iCustomdigits4);
143     img_Show(hndl,iCoolgauge1) ;
144     img_Show(hndl,iCoolgauge2) ;
145     img_Show(hndl,iCoolgauge3) ;
146     img_Show(hndl,iCoolgauge4) ;
147     img_Show(hndl,iSlider1) ;
148     img_Show(hndl,iSlider2) ;
149     img_Show(hndl,iSlider3) ;
150     img_Show(hndl,iSlider4) ;
```

The DIABLO16 has a total of four serial com ports. In line with this, this project has a group of three objects with four indices each. As seen in the previous part of this application note, all of these objects are used for each serial port to perform a particular function.

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);

    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 image touch detect 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’.

```
161         state := touch_Get(TOUCH_STATUS);       // get touchscreen status
162         n := img_Touched(hndl,-1) ;
163
```

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.

```
164         //------------------------------------------------------------
165         if(state == TOUCH_PRESSED)              // if there's a press
166             x := touch_Get(TOUCH_GETX);
167             y := touch_Get(TOUCH_GETY);
168         endif
169         //------------------------------------------------------------
170         if(state == TOUCH_RELEASED)             // if there's a release
171         endif
172         //------------------------------------------------------------
173         if(state == TOUCH_MOVING)               // if it's moving
174             x := touch_Get(TOUCH_GETX);
175             y := touch_Get(TOUCH_GETY);
176             if (n == iSlider1) moveSlider1() ;
177             if (n == iSlider2) moveSlider2() ;
178             if (n == iSlider3) moveSlider3() ;
179             if (n == iSlider4) moveSlider4() ;
180         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.

```
176             if (n == iSlider1) moveSlider1() ;
```

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.

The moveSlider sub-routine

This application projects contained four sub-routines that are identical to each other but are different with regards to the serial communication port that they utilize. The function names were assigned to ease up the identification of the sub-routine. The touch in the Slider1 object will result to a call and execute of a moveSlider1() sub-routine. Also a touch in the Slider2 will call and execute a moveSlider2() sub-routine.

```
13  func moveSlider1()
14      var posn, frame ;
15
16          // Slider1 1.0 generated 8/26/2013 8:59:20 PM
17      img_Show(hndl,iSlider1) ; // show initialy, if required
18      img_ClearAttributes(hndl, iSlider1, I_TOUCH_DISABLE); // set to enable touch, onl
19      posn := x - 44 ;                        // x - left - 8
20      if (posn < 0)
21          posn := 0 ;
22      else if (posn > 245)                    // width - 17)
23          posn := 100 ;                       // maxvalue-minvalue
24      else
25          posn := 100 * posn / 245 ;    // max-min - (max-min) * posn / (width-17)
26      endif
27      img_SetWord(hndl, iSlider1, IMAGE_INDEX, posn);
28      img_Show(hndl, iSlider1);
```

The first part of the sub-routine is the declaration of the local variables used. one the moveSlider1() is called it first shows the slider image. This set of statements is automatically copied to the coding area under the mainSlider1() function with the use of the ‘Paste Code’ button on the Object Inspector window of the Workshop 4 IDE. In this set of statements, the slider1 image is being set to the frame equivalent to the content of the variable ‘posn’. Furthermore, to have a better visual of the slider value this ‘posn’ result will be displayed by the custom LED digit object. When the slider value is changed by sliding the bevel, the resulting value will be stored in the local variable ‘posn’ and then be sent over to the serial

communication port's TX. One the data is already sent, this data will be used to update the value of an object – a cool gauge.

```
30        // Customdigits1 1.0 generated 8/26/2013 9:18:54 PM
31      img_Show(hndl, iCustomdigits1);  // show all digits at 0, only do this once
32      ledDigitsDisplay(posn, iiCustomdigits1, 32, 4, 4, 32, 0) ;
33
34      serout(posn);             // COM0 TX
35      frame:=serin();           // COM0 RX
36
```

This will also be the operation for all the other communication ports. These com ports will serve as the medium in which the data is made to communicate. The serin() function immediately retrieves the data that was sent over to itself. The data is temporarily saved to a variable 'frame'. This returned value is now set as the current value for the cool gauge object.

```
32      serout(posn);             // COM0 TX
33      frame:=serin();           // COM0 RX
34
35          // Coolgauge1 1.0 generated 8/26/2013 9:08:08 PM
36      img_SetWord(hndl, iCoolgauge1, IMAGE_INDEX, frame) ;
37      img_Show(hndl,iCoolgauge1) ;
```

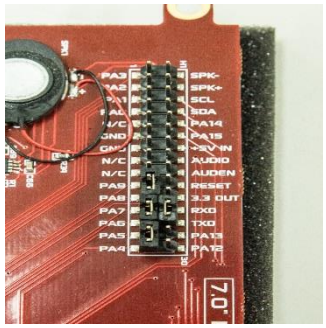
Running the Project

After being able to download the project into your DIABLO16 display module, a couple of steps are need to run the project properly. First thing to do is to check if the slider and the LED digits are working properly. Adjust the position of the slider. These change in position should be equal to the number being displayed in the LED digits. After being able to make sure that the slider object is already working properly, let us focus now out attention on the serial hardware test.

During initial power up of the display device, we would notice that all of our objects are being display in the module. On the contrary, at the instant of moving the slider object we would notice that the cool gauge object will not be displayed. This means that our serial receive port is not able to detect any transmitted value. Referring to the image on the next page, we would see here the result of the missing data.

Due to an open connection between the TX and RX of the communication ports the reception of data is not possible. We will need to have a couple of jumpers.

These jumpers will be connected to the pins relative to the pin assignment made in the serial port initialization segment of the main program. Following the pin assignment made will yield to a set of jumpers positioned similar to the image.



Having been able to set the jumpers to create a connection between the TX and RX of each serial communication ports. The correct data being sent over must be detected already and hence, lead to the cool gauge being displayed.

The cool gauges must match the values displayed on the LED digits and must be able to change its value each time the slider is moved.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability