APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

ViSi Data Communication Over Zigbee

DOCUMENT DATE: **13th April 2019**
DOCUMENT REVISION: **1.1**

Description

This application note is intended to demonstrating to the user the interconnection of the 4D Systems Diablo16 display module with a ZIGBEE personal area network 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.

- 4D Programming Cable / µUSB-PA5/µUSB-PA5-II
  for non-gen4 displays (uLCD-xxx)
- 4D Programming Cable & gen4-IB / gen4-PA / 4D-UPA,
  for gen-4 displays (gen4-uLCD-xxx)
- micro-SD (µSD) memory card
- 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 application note is intended to demonstrating to the user the interconnection of the 4D Systems Diablo16 display module with a ZIGBEE personal area network module.

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

Wireless network technology have paved the way to a better way of interconnecting devices. It has enabled reporting of data from a remote device to a central devices to other remote devices. Without the need of the physical interconnection between devices, this has brought about numerous applications. For this application project, a DRF1605H Zigbee was used. The module is based on TI's CC2530F256 chip running Zigbee.

The Zigbee module in this application project is based on a serial mode of communication. In particular, an asynchronous mode of serial communication is used. The DRF1605H is a module that has its connection pre-defined by the manufacturer. This means that connecting to it is simply straight forward.

The Diablo16 on the other hand is a very capable embedded graphics processor. It drives a touch capable display panel and is capable of communicating over several means of interconnectivity. For serial connection, the Diablo16 I/O pins can be configured and setup to a total of six connections. These serial connections are able to communicate over a wide range of pre-set baud rates but for this application project a 38400 baud rate was used. The baud rate was referenced to the default settings of the Zigbee module.

As mentioned from the previous statements, the interconnection between these two devices utilizes an asynchronous serial mode of communication. The data lines involved are the TRANSMIT and RECEIVE together with the positive and ground of connections of the power supply.

A good reminder at this point is that the power supply used for the Zigbee module is 3.3 volts. These conditions should be considered by the user in case other electronic parts are to be added to the system described in this application project.

Interconnections for the devices

For the Diablo16, the serial port one is used as the communication port for the Zigbee module. The later parts of this document will discuss the setup of the port used. There are several things to consider in using the Zigbee module with the display devices. For a perfectly working application the baud rate must first be considered, for this case a 38400 baud rate was used. Setting the baud rate for the Diablo16 would require functions and statements that are written in 4DGL language and DIABLO16 Internal functions. On the other hand, the baud rate for the Zigbee module must be configured using its own setup software. Secondly, consider the mode on which the system will operate.

The system may be configured to operate as a coordinator or as a router. Setting the mode of the Zigbee to a coordinator would result to the system being a master. Likewise, setting this module to a router would make the system a slave. For this application, the Diablo16 was setup to operate as a master and the slave device was made with another display module. This was done to demonstrate the flexibility of interconnectivity with the Zigbee modules of the 4D Systems displays.

Referring to the images, shown are the images of the DRF-1605H Zigbee module and its header pin-out.
(CLICK ON THE IMAGE FOR MORE INFORMATION)

J1		J2	
1	Debug_D	1	Reset_N
2	Debug_C	2	P0.0
3	P1.7	3	SW1
4	P2.0	4	RX
5	P1.5	5	TX
6	P1.6	6	CTS
7	P1.3	7	RTS
8	P1.4	8	P0.6
9	LED_2	9	P0.7
10	P1.2	10	LED_1
11	NC	11	GND
12	NC	12	3.3V In

The interconnection between the Diablo16 and the DRF1605H primarily involves the headers on J2 of the Zigbee module. Pins J2-4= RX, J2-3=TX, J2-11=GND and J2-12=3.3 Volts supply. The table below shows the pin match up for the MASTER display device and the COORDINATOR Zigbee module.

J2	
1	Reset_N
2	P0.0
3	SW1
4	RX
5	TX
6	CTS
7	RTS
8	P0.6
9	P0.7
10	LED_1
11	GND
12	3.3V In

DIABLO16	DRF-1605H
3.3 volt	J2 – 12
COM0 TX	J2 - 4
COM0 RX	J2 – 5
Gnd	J2 – 11

The connections for the slave is almost the same as the pin match up shown above except for the pins of the display module used. Since the slave device used is a uLCD-43PT, the pins to be used are shown below.

uLCD-43PT	DRF-1605H
3.3 volts	J2 – 12
COM0 TX	J2 - 4
COM0 RX	J2 – 5
Gnd	J2 – 11

Having been able to discuss the electrical connections between the devices, the next thing to consider is the data communication itself. Exchange of data in a Zigbee personal area network may follow several arrangements such that there would be several routers and a central coordinator. For this application project, the data communication was set to a transparent mode. In the transparent communication mode the data is simply interchanged between the master and a particular slave. The exchange of data between the two does not necessarily affect the communication of other routers to the coordinator.

The ViSi-based MASTER Application Project

For this project, the master display module was made to interact with the data sent by the slave. The master module is comprised of several forms which can be accessed by the slave module through strings that are sent over the wireless PAN.

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.

Three other files are included automatically during creation of the new project, namely: 4DGL_16bitColours.fnc, the VisualConst.inc, and ControlConst.inc. Notice that the last include file is almost the same as the project filename. This file is automatically generated as soon as project is saved.

CONTROL*

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

The Main Program

The main program for this projects contains several sections: the initialization and the mounting of the micro-SD card, the initial displaying of the image objects, and a continuous repeat-forever loop. The main loop detects data from the serial communications ports and processes the data received according to a set of if-conditions.

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.

Before the initialization of the microSD card, a pair of statements shown below sets the communication port zero to receive and transmit at a baud rate of 38400 with an initialized buffer named '_data'.

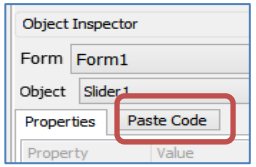
```
12  func main()
13
14      gfx_Set(SCREEN_MODE,LANDSCAPE) ;
15      com_SetBaud(COM0, 3840);
16      com_Init(_data,100,0);
17
18      putstr("Mounting...\n");
19      if (!(disk:=file_Mount()))
20          while(!(disk :=file_Mount()))
21              putstr("Drive not mounted...");
22              pause(200);
23              gfx_Cls();
24              pause(200);
25          wend
26      endif
27
28      hndl := file_LoadImageControl("CONTROL.dat", "CONTROL.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 these 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'. In addition, the filenames for the dat and gci files are automatically changed to the project filename as soon as the project is saved.

Added to the statements above are the screen orientation setup function and the touch enable function. The screen orientation is set to portrait thereby providing a 480x800 pixel dimension. This setup is attained using the gfx_Set() function.

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.

A special button from the Object Inspector can help reduce the time of coding of this part. The 'Paste Code" simply pastes object code.



Displaying the Objects

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 call out the images from their handlers and are displayed on screen. Observe the characters with the green font colours. This denotes the available object frame numbers available for the particular User Image object.

```
29  again:
30      gfx_Cls();
31      img_Show(hndl,iStatictext1) ;
32      img_Show(hndl,iStatictext9) ;
33      img_Show(hndl,iStatictext10) ;
```

Note that there is a statement that added before the series of image show statements. The 'again' is a label for a 'jump to location' inside the main program. The use of this shall be shown in the succeeding section.

The Repeat-forever Loop

This section of the main program contains statements that are continually run by the processor. For this project, the serial data is saved to the variable 'n'. The serial input serin() function is monitored using the if-condition statement. Whenever the

result value on the variable n is not equal to -1, the data received is saved into the n.

```
repeat

    n:= serin();
    if(n != -1)
            rec_buffer[m]:=n;
            m++;
    endif

    if((rec_buffer[0] == 0x48) && (rec_buffer[1] == 0x4F) && (rec_buffer[2] == 0x4D) && (rec_buffer[3] == 0x45))
            clear();
            goto again;
    else if((rec_buffer[0] == 0x46) && (rec_buffer[1] == 0x4F) && (rec_buffer[2] == 0x52) && (rec_buffer[3] == 0x4D) && (rec_buffer[4] == 0x20) && (rec_buffer[5] == 0x32))
            form2() ;
            clear();
    else if((rec_buffer[0] == 0x46) && (rec_buffer[1] == 0x4F) && (rec_buffer[2] == 0x52) && (rec_buffer[3] == 0x4D) && (rec_buffer[4] == 0x20) && (rec_buffer[5] == 0x34))
            form3();
            clear();
    else if((rec_buffer[0] == 0x46) && (rec_buffer[1] == 0x4F) && (rec_buffer[2] == 0x52) && (rec_buffer[3] == 0x4D) && (rec_buffer[4] == 0x20) && (rec_buffer[5] == 0x35))
            form4();
            clear();
    else if((rec_buffer[0] == 0x46) && (rec_buffer[1] == 0x4F) && (rec_buffer[2] == 0x52) && (rec_buffer[3] == 0x4D) && (rec_buffer[4] == 0x20) && (rec_buffer[5] == 0x33))
            form5();
            clear();
    endif

forever
endfunc
```

There is a series of if-else conditional statement shown above. These conditions tests the values received and saved on the variable array rec_buffer. The values received herein is a set of hexadecimal values that are the equivalent of a string being sent from the slave module.

The if-else condition needs several conditions to call and execute a particular sub-routine. Considering the first condition on the statements shown above, if the received bytes are equivalent to the values starting from buffer index zero up to three it will direct the processor to jump to the address given by the label ‘again’. For the other if-else conditions, the content of the rec_buffer is compared to the

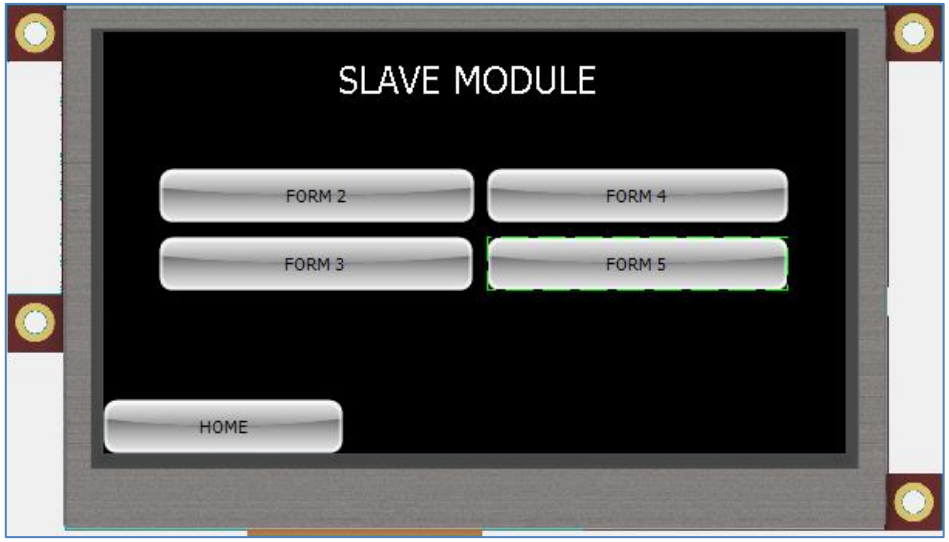
slave

```
#platform "uLCD-43PT"
#inherit "4DGL_16bitColours.fnc"
#inherit "VisualConst.inc"
#inherit "slaveConst.inc"
```

values and index position given in the program. If the values and index position are all the same as the reference then the subroutine statements is called and executed.

The ViSi-based SLAVE Application Project

The slave project includes a couple of winbutton that are setup to send out data over the serial communications port when pressed.



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.

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. Three other files are included automatically during creation of the new project, namely: 4DGL_16bitColours.fnc, the VisualConst.inc, and slaveConst.inc. Notice that the last include file is almost the same as the project filename. This file is automatically generated as soon as project is saved.

slave

```
#platform "uLCD-43PT"
#inherit "4DGL_16bitColours.fnc"
#inherit "VisualConst.inc"
#inherit "slaveConst.inc"
```

The Main Program

The main program for this projects contains several sections: the initialization and the mounting of the micro-SD card, the initial displaying of the image objects, and a continuous repeat-forever loop that is related to the touch status of the button objects.

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.

```
func main()
    touch_Set(TOUCH_ENABLE);
    gfx_ScreenMode(LANDSCAPE);
    com_SetBaud(COM0, 3840);

    putstr("Mounting...\n");
    if (!(disk:=file_Mount()))
        while(!(disk :=file_Mount()))
            putstr("Drive not mounted...");
            pause(200);
            gfx_Cls();
            pause(200);
        wend
    endif
    gfx_Cls();

    hndl := file_LoadImageControl("slave.dat", "slave.gci", 1);
```

Before the statements that detect and initialize the microSD, a statement using the touch_Set() function enables the touch function. Another statement sets the screen orientation and the last one configures the baud rate that will be used for the com0 communications port.

Displaying the Objects

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 call out the images from their handlers and are displayed on screen. Observe the characters with the green font colours. This denotes the available object frame numbers available for the particular User Image object.

```
    img_Show(hndl,iStatictext1) ;
    img_Show(hndl,iWinbutton1) ;
    img_Show(hndl,iWinbutton2) ;
    img_Show(hndl,iWinbutton3) ;
    img_Show(hndl,iWinbutton4) ;
    img_Show(hndl,iWinbutton5) ;
    img_ClearAttributes(hndl, iWinbutton1, I_TOUCH_DISABLE); // set to enable touch, only need to do this once
    img_ClearAttributes(hndl, iWinbutton2, I_TOUCH_DISABLE); // set to enable touch, only need to do this once
    img_ClearAttributes(hndl, iWinbutton3, I_TOUCH_DISABLE); // set to enable touch, only need to do this once
    img_ClearAttributes(hndl, iWinbutton4, I_TOUCH_DISABLE); // set to enable touch, only need to do this once
    img_ClearAttributes(hndl, iWinbutton5, I_TOUCH_DISABLE); // set to enable touch, only need to do this once
```

Setting of the touch feature for an image requires the function Img_Clearattributes(). This function allows the touch detection on the images. The result of the touch on the images is saved on a variable in the repeat-forever loop.

The Repeat-forever Loop

This section of the main program contains statements that are continually run by the processor. For this project, the serial data is saved to the variable 'n'. The serial input serin() function is monitored using the if-condition statement. Whenever the result value on the variable n is not equal to -1, the data received is saved into the n.

```
    repeat


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

Following the detection of a touch action on the image, a set of if-conditions process the status of the touch panel. The touch panel is checked for two conditions – a press over an image or a touch released on an image.

```
48          //------------------------------------------------------------
49          if(status == TOUCH_PRESSED)             // if there's a press
50              x:= touch_Get(TOUCH_GETX);
51              y:= touch_Get(TOUCH_GETY);
52
53              state:= 1;
54
55              if(n == iWinbutton1)
56                  pause(100) ;
57                  img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
58                  img_Show(hndl,iWinbutton1) ;
59                  to(COM0); print("HOME");
60
61              else if(n == iWinbutton2)
62                  pause(100) ;
63                  img_SetWord(hndl, iWinbutton2, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
64                  img_Show(hndl,iWinbutton2) ;
65                  to(COM0); print("FORM 2");
66
67              else if(n == iWinbutton3)
68                  pause(100) ;
69                  img_SetWord(hndl, iWinbutton3, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
70                  img_Show(hndl,iWinbutton3) ;
71                  to(COM0); print("FORM 3");
72
73              else if(n == iWinbutton4)
74                  pause(100) ;
75                  img_SetWord(hndl, iWinbutton4, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
76                  img_Show(hndl,iWinbutton4) ;
77                  to(COM0); print("FORM 4");
78
79              else if(n == iWinbutton5)
80                  pause(100) ;
81                  img_SetWord(hndl, iWinbutton5, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
82                  img_Show(hndl,iWinbutton5);
83                  to(COM0); print("FORM 5");
84              endif
```

If a touch pressed is detected, the next is to determine which image was touched. After both conditions are satisfied, a couple of statements are executed. The first statements that are executed are related to images simply changes the image index state of the button object. The last statements includes a to() and print() function pair, send out a string to the serial communications port. These strings are the data being received by the master module and processed.

Another touch related condition is the 'touch_released' condition. After a touch is detected on the screen, this conditional loop detects the release part of the process. The sole purpose of this if conditional loop is to return the button's image index to zero. The transitions that occur during the touch and the release results to a simple animation that shows images being pressed and released.

```
91          //------------------------------------------------------------
92          else if(status == TOUCH_RELEASED)           // if there's a release
93              state:= 0;
94
95              if(n == iWinbutton1)
96                  img_SetWord(hndl, iWinbutton1, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
97                  img_Show(hndl,iWinbutton1) ;
98
99              else if(n == iWinbutton2)
100                 img_SetWord(hndl, iWinbutton2, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
101                 img_Show(hndl,iWinbutton2) ;
102
103             else if(n == iWinbutton3)
104                 img_SetWord(hndl, iWinbutton3, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
105                 img_Show(hndl,iWinbutton3) ;
106
107             else if(n == iWinbutton4)
108                 img_SetWord(hndl, iWinbutton4, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
109                 img_Show(hndl,iWinbutton4) ;
110
111             else if(n == iWinbutton5)
112                 img_SetWord(hndl, iWinbutton5, IMAGE_INDEX, state); // where state is 0 for up and 1 for down
113                 img_Show(hndl,iWinbutton5) ;
114             endif
115         endif
116
117         //------------------------------------------------------------
118
119     forever
120 endfunc
```

Running the Program

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-32PTU and uLCD-35DT display modules are commonly used as examples, but the procedure is the same for other displays.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability