APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi GPIO Pins Picaso

DOCUMENT DATE: **12th APRIL 2019**
DOCUMENT REVISION: **1.1**

Description

This application note shows how to configure and use the GPIO pins of a Picaso display module. The 4DGL code of the Designer project can be copied and pasted to an empty ViSi project and it will compile normally. The code can also be integrated to that of an existing ViSi project.

Before getting started, the following are required:

- Any of the following 4D Picaso display modules:

gen4-uLCD-24PT gen4-uLCD-28PT gen4-uLCD-32PT
uLCD-24PTU uLCD-32PTU uVGA-III

and other superseded modules which support the Designer and/or ViSi environments.

- 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)
- 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

The Designer environment enables the user to write 4DGL code in its natural form to program the display module. 4DGL is a graphics oriented language allowing rapid application development, and the syntax structure was designed using elements of popular languages such as C, Basic, Pascal and others. Programmers familiar with these languages will feel right at home with 4DGL.

The purpose of this application note is, besides showing the user how to configure and use the GPIO pins, to introduce the basics of 4DGL through examples.

Setup Procedure

For instructions on how to launch Workshop 4, how to open a **Designer** project, and how to change the target display, kindly refer to the section "**Setup Procedure**" of the application note
Designer Getting Started - First Project

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 **Designer** project, please refer to the section "**Create a New Project**" of the application note
Designer Getting Started - First 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

The GPIO Pins

Picaso display modules have thirteen general purpose input/output (GPIO) pins available to the user. These are grouped as **IO1 to IO5** and **BUS0 to BUS7**. The five I/O pins (IO1 to IO5) provide flexibility of individual bit operations while the 8 pins (BUS0 to BUS7), known as GPIO BUS, serve collectively for byte wise operations. The IO4 and IO5 pins also act as strobing signals to control the GPIO Bus. The GPIO Bus can be read or written to by strobing IO4/BUS_RD or IO5/BUS_WR (respectively) a low pulse (with a 50 nsec duration or greater). The figure below shows the backside of a uLCD-32PTU and the location of the GPIO pins (IO1 to IO5 and BUS0 to BUS7) in the expansion header. Refer to the datasheet of your display module for the physical pin configuration.

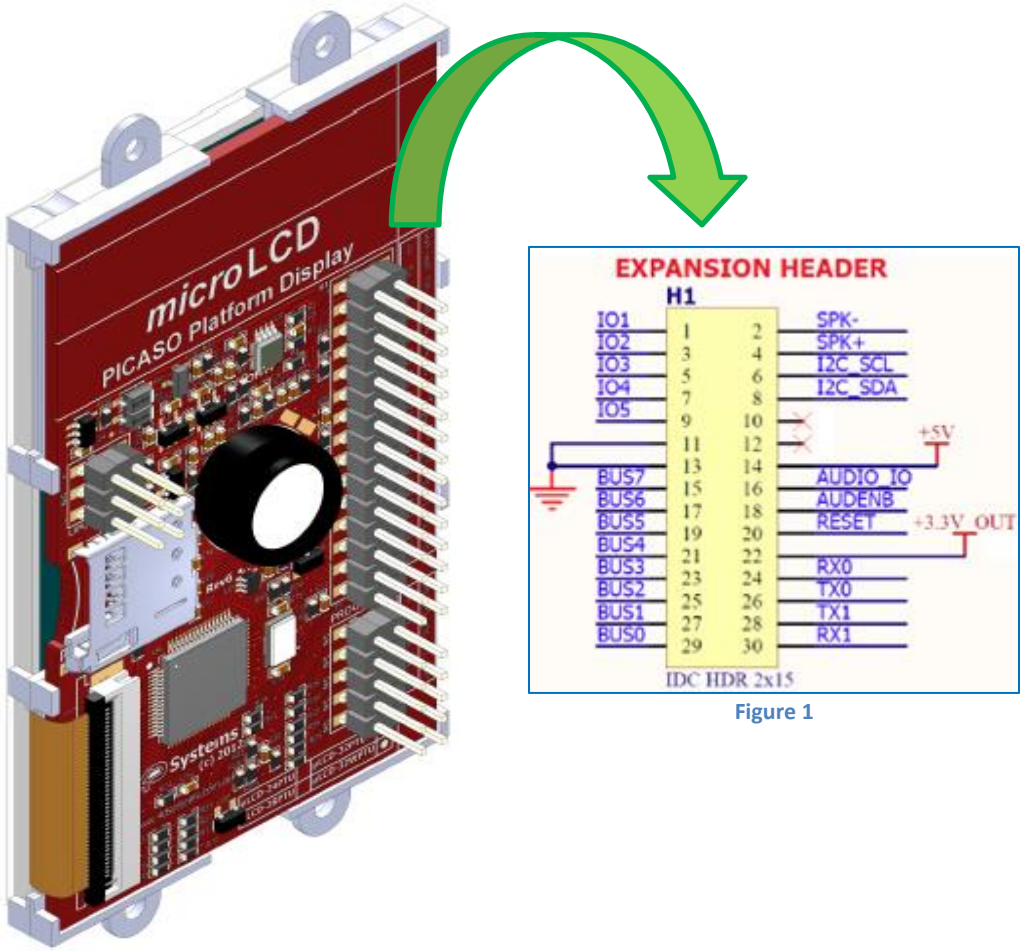


Figure 1

The following text describes the pins of the uLCD-32PTU and their purpose/s. Refer to your display module's datasheet for the appropriate pin description.

IO1-IO5 pins:
General purpose I/O pins. Each pin can be individually set as an INPUT or OUTPUT.

IO1 pin (Frame Mark pin):
The IO1 pin has two functions. It can be used as a GPIO pin, and it can also be used to detect the start of a Frame. When used as a GPIO pin, simply connect an Input/Output to the IO1 pin of the H1 header. When using IO1 for Frame Mark, simply leave the IO1 pin of the H1 header disconnected, and read the status of IO1 as an Input. ***Note: frame mark function is not available in uLCD-43P/PT/PCT modules. IO1 is used as a GPIO pin only.***

IO2 pin (Lithium Battery Status pin):
The IO2 pin has two functions. It can be used as a GPIO pin, and it can also be used to tell when the Lithium battery has reached a low level (3.7V) and needs to be charged. When used as a GPIO pin, simply connect an Input/Output to the IO2 pin on the H1 header. When using IO2 for Battery Status, simply leave the IO2 pin on the H1 header disconnected, and read the status of IO2 as an Input. ***Note: battery status detection function is not available in uLCD-43P/PT/PCT modules. IO2 is used as a GPIO pin only.***

IO3 pin (Peripheral Supply pin):
IO3 is controllable via the processor, or via the H2 Interface pin driven by an external circuit. If IO3 is set as OUTPUT and driven HIGH the µSD and Display are enabled, and disabled when driven LO. Set as INPUT to use an external circuit to drive this pin. ***Note: peripheral supply function is not available in uLCD-43P/PT/PCT modules. IO3 is used as a GPIO pin only.***

IO4/BUS_RD pin (GPIO IO4 or BUS_RD pin):
General Purpose IO4 pin. Also used for BUS_RD signal to read and latch the data in to the parallel GPIO BUS0 to BUS7.

IO5/BUS_WR pin (GPIO IO5 or BUS_WR pin):
General Purpose IO5 pin. Also used for BUS_WR signal to write and latch the data to the parallel GPIO BUS0 to BUS7.

BUS0-BUS7 pins (GPIO 8-Bit Bus):
8-bit parallel General purpose I/O Bus.
Note: All GPIO pins are 5.0V tolerant.

The GPIO Pin Functions

The following sections will discuss how to configure the GPIO pins either as input or output, how to control the logic level of a pin (when set as an output), and how to read the logic level of a pin (when set as an input). Again, the 5 I/O pins (IO1 to IO5) provide flexibility of individual bit operations while the 8 pins (BUS0 to BUS7), known as GPIO bus, serve collectively for byte wise operations. The term bus here, therefore, is used to collectively refer to the pins BUS0 to BUS7, while the term pin refers to any GPIO pin.

A GPIO function, when prepended with **pin_**, is used for configuring, setting, or reading logic states from a pin. On the other hand, a GPIO function prepended with **bus_** is used for configuring, controlling, or reading logic levels from a group of pins - in this case the GPIO bus.

Bus Mode

Bus mode defines how the bus will be used – as an input or as an output. To set the GPIO bus mode, use the function

```
bus_Set(arg1);
```

The lower 8 bits of arg1 are placed in the BUS direction register. The upper 8 bits of arg1 are ignored.

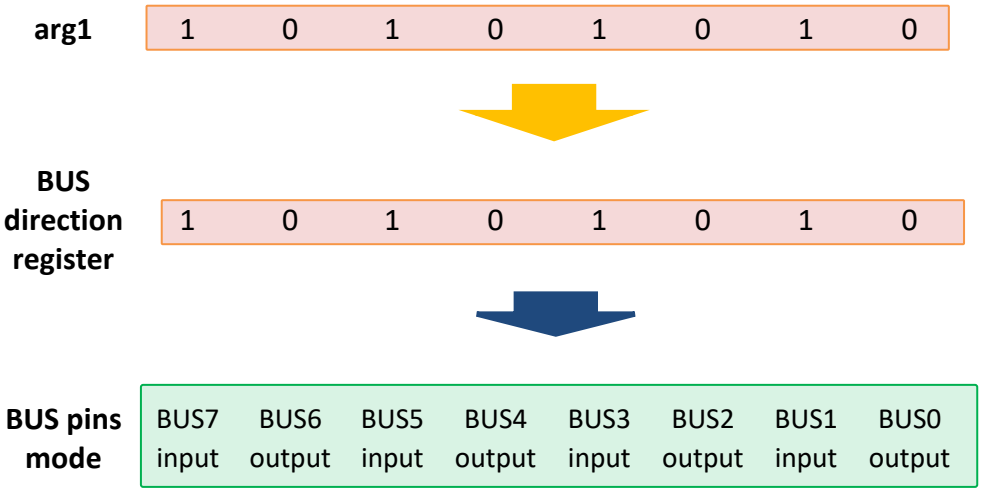
bit	pin mode
0	output
1	input

Example:

ignored

```
bus_Set(0x00AA);
```

In binary, 0xAA (hexadecimal) is equal to 10101010.



To set all of the bus pins as inputs:

```
bus_Set(0xFF);
```

To set all of the bus pins as outputs:

```
bus_Set(0x00);
```

Pin Mode

To individually set the mode of any of the IO1 to IO5 pins use the function:

```
pin_Set(mode, pin);
```

For instance, to set IO1 as an output:

```
pin_Set(OUTPUT, IO1_PIN);
```

To set I03 as an input:

```
pin_Set(INPUT, IO3_PIN);
```

Control the GPIO Bus

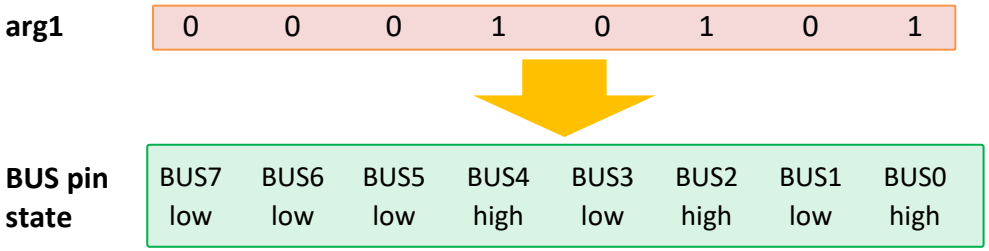
After having configured the bus as an output, the pins can now be set to high or low using the function

```
bus_Out(arg1);
```

The lower byte of arg1 is placed on the 8-bit-wide bus. The upper byte is ignored. Note that any bus pin set as an input is not affected.
To illustrate:

```
bus_Out(0x0015);
```

In binary, 0x15 is equal to 00010101.



Control the GPIO Pins

The pins can also be set to high or low individually, without affecting the state of the other pins, using the function

```
pin_HI(pin);
```

This function outputs a high level (logic 1) on the specified pin previously set as an output. If the pin is not already set as an output, it is automatically set as such. Possible values for pin are shown below:

```
pin = 1  : set IO1_PIN to "High" level (pin 2 of J1)
pin = 2  : set IO2_PIN to "High" level (pin 1 of J1)
pin = 3  : set IO3_PIN to "High" level (pin 3 of J1)
pin = 4  : set IO4_PIN to "High" level (pin 5 of J1, also used for BUS_RD)
pin = 5  : set IO5_PIN to "High" level (pin 9 of J2, also used for BUS_WR)
pin = 6  : set DCENB pin to "High" level (BACKLITE ON)
pin = 7  : set AUDIO_ENABL pin to "High" level (AMP OFF)
pin = 8  : set BUS_0   to "High" level (pin 27 of J1)
pin = 9  : set BUS_1   to "High" level (pin 25 of J1)
pin = 10 : set BUS_2   to "High" level (pin 23 of J1)
pin = 11 : set BUS_3   to "High" level (pin 21 of J1)
pin = 12 : set BUS_4   to "High" level (pin 19 of J1)
pin = 13 : set BUS_5   to "High" level (pin 17 of J1)
pin = 14 : set BUS_6   to "High" level (pin 13 of J2)
pin = 15 : set BUS_7   to "High" level (pin 11 of J2)
```

Example:

```
pin_HI(BUS_3); //same as pin_HI(11);
```

To individually set the bus pins to low, use the function

```
pin_LO(pin);
```

This function outputs a low level (logic 0) on the specified pin previously set as an output. If the pin is not already set as an output, it is automatically set as such. Possible values for pin are shown below:

```
pin = 1 : set IO1_PIN to "Low" level (pin 2 of J1)
pin = 2 : set IO2_PIN to "Low" level (pin 1 of J1)
pin = 3 : set IO3_PIN to "Low" level (pin 3 of J1)
pin = 4 : set IO4_PIN to "Low" level (pin 5 of J1, also used for BUS_RD)
pin = 5 : set IO5_PIN to "Low" level (pin 9 of J2, also used for BUS_WR)
pin = 6 : set DCENB pin to "Low" level (BACKLITE)
pin = 7 : set AUDIO_ENABL pin to "Low" level (AMP ON)
pin = 8  : set BUS_0  to "Low" level (pin 27 of J1)
pin = 9  : set BUS_1  to "Low" level (pin 25 of J1)
pin = 10 : set BUS_2  to "Low" level (pin 23 of J1)
pin = 11 : set BUS_3  to "Low" level (pin 21 of J1)
pin = 12 : set BUS_4  to "Low" level (pin 19 of J1)
pin = 13 : set BUS_5  to "Low" level (pin 17 of J1)
pin = 14 : set BUS_6  to "Low" level (pin 13 of J2)
pin = 15 : set BUS_7  to "Low" level (pin 11 of J2)
```

Example:

```
pin_LO(IO1_PIN);
```

Read the State of a Bus

After having configured the bus as an input, its state can be read with the function:

```
bus_In( );
```

The function returns the state of the bus as an eight bit value and assigns it to the lower byte of the assigned variable.

Example:

```
var1 := bus_In( );
```

The lower byte of var1 will get loaded with the state of the bus.

Read the State of a Pin

After having configured a pin as an input, its state can be read with the function:

```
pin_Read(pin);
```

This function reads the logic state of the pin. It returns a logic 0 (0x0000) or a logic 1 (0x0001).

Example:

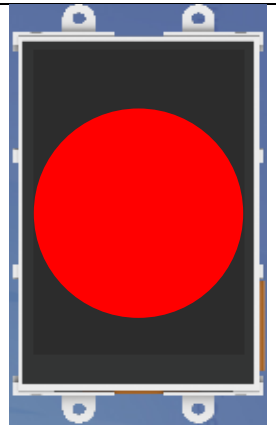
```
var1 := pin_Read(IO1);
```

The variable **var1** will have a value of either 0x0000 or 0x0001, depending on the logic state of **IO1**.

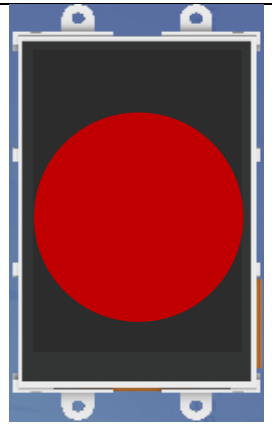
A Simple Project – IO1 as an Input

Description

This simple project reads the logic state of the IO1 pin which was configured as an input and connected to a tact switch (see schematic in Figure 17). If the logic state of IO1 is 1, the screen displays a dark reddish circle. If the logic state of IO1 is 0, the screen displays a red circle. The screen (uLCD-32PTU, portrait orientation in this example) will look like as shown below.



ON Red circle – IO1 is low



OFF Dark red circle – IO1 is high

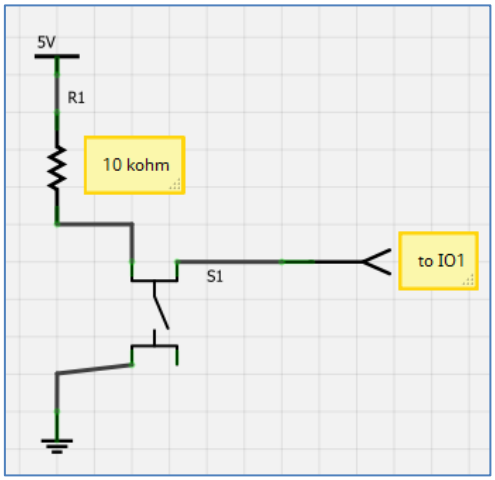


Figure 2 Switch Schematic

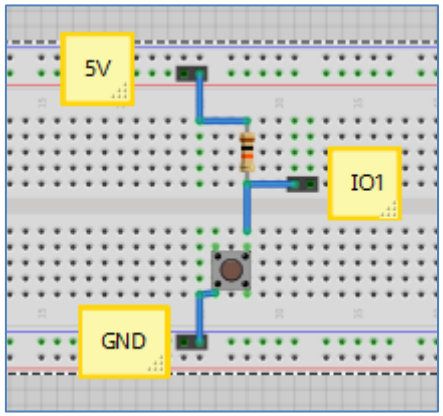


Figure 3 Switch (Breadboard)

Program Code

Below is a code for the project. Comments are added to guide the user.

```
1   #platform "uLCD-32WPTU"
2
3   #inherit "4DGL_16bitColours.fnc"
4
5   func main()
6
7    gfx_ScreenMode(PORTRAIT) ; // change manually if orientation
8    pin_Set(INPUT, IO1_PIN); //setup pinIO1 as an input
9
10   repeat
11     if(pin_Read(IO1_PIN) == 0)  //if IO1_PIN is low
12       print("ON \r");           //display circle at on state
13       gfx_CircleFilled(120, 200, 100, RED);
14
15     else                       //if IO1_PIN is high
16       print("OFF\r");           //display circle at off state
17       gfx_CircleFilled(120, 200, 100, DARKRED);
18     endif
19   forever
20  endfunc
```

Figure 4

Attached is a zipped file of the code in Figure 19. A brief explanation now follows. Lines previously explained are bypassed.

Lines 10 to 19 make up the repeat…forever loop. It begins with

```
10    repeat
```

and ends with

```
19    forever
```

Instructions in between these lines are executed forever.

The if-else-endif Statement

Lines 11 to 18 now make up the **if-else-endif** statement. The program makes use of the **if-else-endif** statement to make a decision. Below is the syntax or format:

```
Syntax:

        if(condition)
            [statements]
        else
            [statements]
        endif
```

First the program tests a condition. If the condition is found to be true, the program executes a set of instructions. If not (else), the program executes another set of instructions. In Figure 19, the condition being tested is the logic state of IO1_PIN. If it is at logic 0

```
11      if(pin_Read(IO1_PIN) == 0)  //if IO1_PIN is low
```

The program prints “ON” and draws a red solid circle on the screen.

```
12        print("ON \r");          //display circle at on state
13        gfx_CircleFilled(120, 200, 100, RED);
```

Else, if IO1_PIN is at logic 1 or for any other condition,

```
15      else                      //if IO1_PIN is high
```

the program prints “OFF” and draws a **dark red** solid circle on the screen.

```
16        print("OFF\r");    //display circle at off state
17        gfx_CircleFilled(120, 200, 100, DARKRED);
```

The if-else-endif statement ends with the line

```
18    endif
```

Now play with the tact switch and the colour of the circle should change accordingly. To learn more about the if-else-endif statement and loops, consult the 4DGL Programmers Reference Manual .

Run the Program

For instructions on how to save a **Designer** 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 the application note

Designer Getting Started - First Project

For instructions on how to save a **ViSi** 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 the application note

ViSi Getting Started - First Project for Picaso and Diablo16

The uLCD-32PTU display module is commonly used as an example, but the procedure is the same for other displays.

Proprietary Information

Disclaimer of Warranties & Limitation of Liability