APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi Loop Back Testing with a PIC MCU

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

Description

This Application Note is intended to demonstrating and teaching the user how to test the interface between the 4D PICASO display modules with the MICROCHIP PIC microcontrollers. This application is intended for use in the 4D Workshop 4 – Designer environment. The tools needed includes the following;

Before getting started, the following are required:

- Any of the following 4D Picaso display modules:

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

  and other superseded modules which support the Designer and/or ViSi environments.

- 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 document is about basic asynchronous serial interfacing of the 4D display module with the PIC18F45K20. It contains a simple but helpful information on how to test the data information being sent by the PICASO serial port to a host controller in particular the MICROCHIP PIC18F34K20. The system is arranged such that the information data is sent by the PICASO display. This information is then received by the PIC18F45K20 and sends it back for the screen to display.

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

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

In particular, we will be using the Serial (UART) Communications Functions. The PICASO serial communications is relatively easy to use. Since the PICASO display module only uses 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 PICASO display module has two 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.

Below is a program that will initialize and set-up the serial communication port of the 4D display module.

```
#platform "uLCD-28PTU"
#inherit "4DGL_16bitColours.fnc"
    var dat;
    var buf[5];

func main()
    var n:=0;
    gfx_ScreenMode(LANDSCAPE_R) ;    // set screen mode in reverse landscape
    com_SetBaud(COM0,960);           // set COM port 0 with 9600 baud rate
    com_Init(buf, 5, 0x00);          // set buffer with size of 5 and no qualifier
repeat
    while(n<=0xFF)
        serout(n);                   // send 0XF0 value to TX 0.
        txt_MoveCursor(3,8) ;
        print("Data Transmitted:   ", [HEX2Z] n); // print value of transmit data
        pause(75);                   // wait 75 milliseconds
        dat := serin();              // dat is temporarily saved in this variable
             if(dat <= -1)           // check if received data is less than -1
                txt_MoveCursor(5,10);
                print("No data received");  // print this is value is -1
             else
                txt_MoveCursor(5,8) ;
                print("Data received:   ", [HEX2Z] dat); // print value of received
             endif
        n++;                         // increment the value of n by one
    wend                             // end while loop
    n:=0;                            // return value of n to zero
forever

endfunc
```

From the program above it is clearly seen that during the start of the program main(), the serial communications port 0 is being setup to communicate with the PIC host controller. It is set to transmit and receive at a baud rate of 9600.

```
6  func main()
7      var n:=0;
8      gfx_ScreenMode(LANDSCAPE_R) ;   // set screen mode in reverse landscape
9      com_SetBaud(COM0,960);          // set COM port 0 with 9600 baud rate
10     com_Init(buf, 5, 0x00);         // set buffer with size of 5 and no qualifier
```

In this program, see the statement com_SetBaud(COM0, 960). Notice that with this function the serial communication port 0 is set with a baud rate of 9600. In this 4DGL function the actual desired baud rate is written but with the real baud /10.

Moving to the next part of the program, notice how simple the initialization of the serial com port is done. The initialization statement com_Init() includes a variable array buf, buffer size and a qualifier. The qualifier is the part of the initialization that if specified, the PICASO serial communication will not start unless the qualifier is received. If the qualifier is set to '0', this implies that no start bit qualifier is specified.

Next, we have the while loop. The while generates the data that is to be sent to the serial lines.

```
11 repeat
12     while(n<=0xFF)
13         serout(n);                  // send 0XF0 value to TX 0.
14         txt_MoveCursor(3,8) ;
15         print("Data Transmitted:   ", [HEX2Z] n); // print value of transmit data
16         pause(75);                  // wait 75 milliseconds
17         dat := serin();             // dat is temporarily saved in this variable
18              if(dat <= -1)          // check if received data is less than -1
19                 txt_MoveCursor(5,10);
20                 print("No data received");  // print this is value is -1
21              else
22                 txt_MoveCursor(5,8) ;
23                 print("Data received:   ", [HEX2Z] dat); // print value of received
24              endif
25         n++;                        // increment the value of n by one
26     wend                            // end while loop
27     n:=0;                           // return value of n to zero
```

The variable n is incremented starting from zero until it reaches the limit of 0xFF hex limit. Each time the value of n is incremented starting from zero, this are sent to the serial communication port zero TX. The sending of data is done with the serout(n) function in line number 13.

```
13         serout(n);                  // send 0XF0 value to TX 0.
14         txt_MoveCursor(3,8) ;
15         print("Data Transmitted:   ", [HEX2Z] n); // print value of transmit data
```

After sending this value of n, it is also displayed in the screen to show what value is currently being sent over the serial TX. Following the sending routine is the receiving of data bytes coming from the host controller. With the aid of the serin() function we are able to get the buffered RX data. This is being read and displayed into the screen.

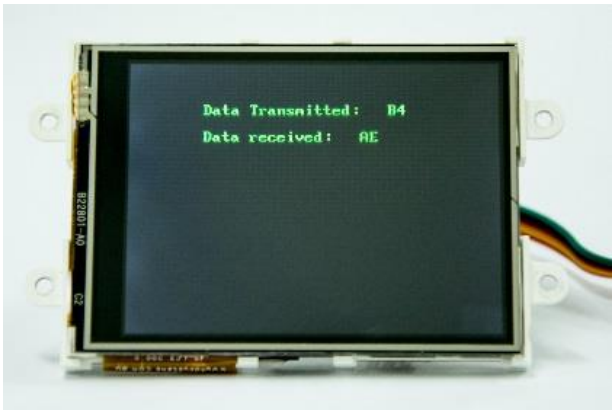
An if-else routine is used to determine if there are real data being received. If the PICASO display module serial RX is left hanging, it results to a -1 value which means that there are no data at the moment. If the serin() results to a value of -1 – the program displays a message indicating that there are no data in the serial RX line.

```
17         dat := serin();             // dat is temporarily saved in this variable
18              if(dat <= -1)          // check if received data is less than -1
19                 txt_MoveCursor(5,10);
20                 print("No data received");  // print this is value is -1
21              else
22                 txt_MoveCursor(5,8) ;
23                 print("Data received:   ", [HEX2Z] dat); // print value of received
24              endif
```

On the other hand if a data is receive then this data is printed in the display with a message "Data Received" preceding it.

After compiling and download of the program, the PICASO display module will show the following output. If the MICROCHIP host controller is not connected during test:

This means that the PIC18F is not connected and that the serial connections are left hanging or disconnected. On the other hand, if we connect the PIC18F to the display module, we will have a result in the display similar to the one below.



During the process of communication the delay caused by the pause(500) results to a lag in the retransmitted value. This is also caused by the operation manner of running a program in microcontrollers wherein the statements of a program is run one step at a time. Although there is a lag in data it does not mean that it is incorrect. The series of data are being buffered by the display module and are being displayed on the screen. On the other hand the PIC18F controller continuously receive and re-transmits all data being received from the display module.

The PIC 18F45K20 Loopback program

The 18F45k20 host controller loopback program starts with the declaration of its configuration files. The following configurations were used:

```
// PIC18F45k20 configuration
#pragma config FOSC = INTIO67, FCMEN = OFF, IESO = OFF              // CONFIG1H
#pragma config PWRT = OFF, BOREN = OFF, BORV = 30                   // CONFIG2L
#pragma config WDTEN = OFF, WDTPS = 32768                           // CONFIG2H
#pragma config MCLRE = ON, LPT1OSC = OFF, PBADEN = ON, CCP2MX = PORTC     //
CONFIG3H
#pragma config STVREN = ON, LVP = OFF, XINST = OFF                  // CONFIG4L
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF           // CONFIG5L
#pragma config CPB = OFF, CPD = OFF                                 // CONFIG5H
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF     // CONFIG6L
#pragma config WRTB = OFF, WRTC = OFF, WRTD = OFF                   // CONFIG6H
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF  // CONFIG7L
#pragma config EBTRB = OFF
```

The main body of the PIC18F45K20 loopback program is relatively easy to understand since it only uses the custom-defined function sendbyte() to transmit a single byte data and the receive() function which continuously receives the input data to the PIC18F45k20. The port D of the demo board was used to display the received value and as an indicator of data reception.

```
/** I N C L U D E S **************************************************/
#include <p18f45k20.h>
#include <delays.h>
#include "UART.h"
/** V A R I A B L E S ************************************************/
void main (void)
{
   uart();
   TRISD = 0x00;
while(1)
{
   sendbyte(receive());
   PORTD = receive();
}
```

A header file is written to initialize the host controller's serial communication port. This initialization configuration is included in the UART.h header file. This header has the definition for the functions sendbyte () and receive ().

Referring to the UART.h contents on the opposite column. We could see a function uart(). This function can be called in the main program to configure the serial communication port.

```
// UART.h
void uart(void)
{     TRISCbits.TRISC7=1;                  //Make UART RX pin input
      TRISCbits.TRISC6=0;                  //Make UART TX pin output
      SPBRGH  = 0    ;                      //9600bps 16MHz Osc
      SPBRG   = 25;
      RCSTAbits.CREN=1;                    //Enables receiver
       RCSTAbits.SPEN=1;                   //Serial port enable
       BAUDCONbits.BRG16=1;// enable 16-bit Baud Rate Generator
       BAUDCONbits.CKTXP = 1;
      TXSTAbits.SYNC=0;                    //0 = Asynchronous mode
      TXSTAbits.BRGH=1;                    //1 = High speed
      TXSTAbits.SENDB = 0;
      BAUDCONbits.DTRXP =1;
      PIE1bits.TXIE = 1;
}
unsigned char receive(void)
{   unsigned char data;
   if(RCSTAbits.FERR==1 && RCSTAbits.OERR==1 )
   {   while(PIR1bits.RCIF == 1)
      {  RCSTAbits.CREN =0;    //Overrun error (can be cleared by clearing bit CREN)
        data = RCREG;
        RCSTAbits.CREN =1;
      }
   }
   else
   {   while(PIR1bits.RCIF == 1)
      {   RCSTAbits.CREN =0;    //Overrun error (can be cleared by clearing bit CREN)
         data = RCREG;
         RCSTAbits.CREN =1;
      }
   } return data; }
```

```
void sendbyte(unsigned char data)
{
      TXREG = data;
      TXSTAbits.TXEN=0;
        while(TXSTAbits.TRMT == 0)
        {
              TXSTAbits.TXEN=1;    // enable transmission
              while(TXSTAbits.TRMT == 0 ) // wait here till transmit complete
              { NOP(); }
        }
      TXSTAbits.TXEN=0;
}
```

The routines to receive the data from the display is already included in the header file. This receive routine readily accepts data from the display module and return this as a value ‘data‘. Furthermore, the sendbyte() function is able to send a single byte at every function call.

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

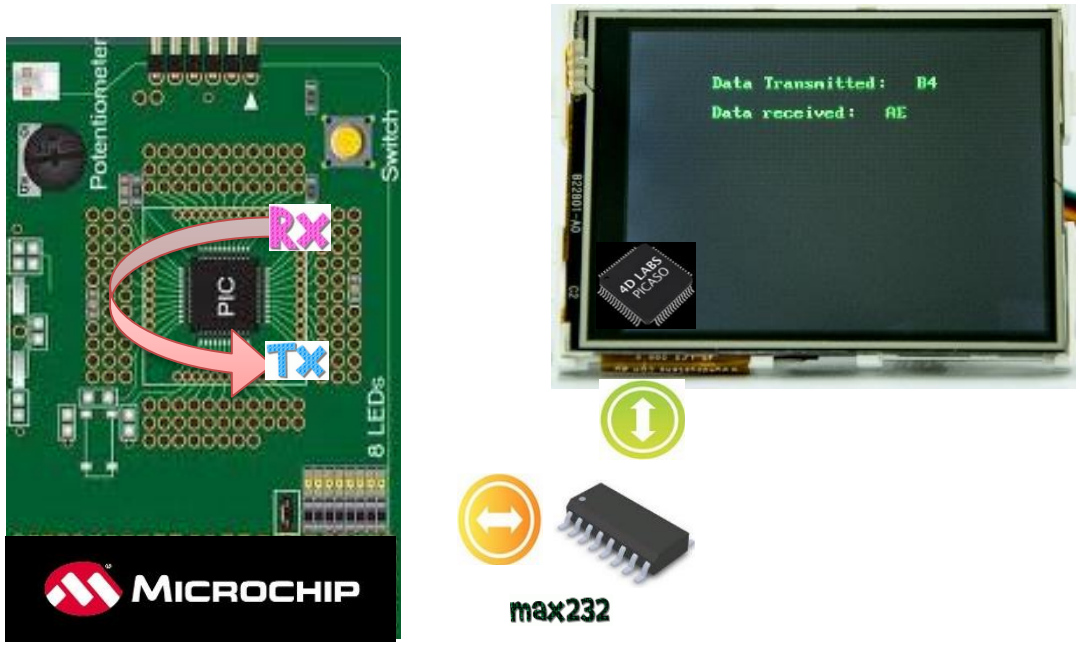
Designer Getting Started - First Project

For instructions on how to save a **ViSi** project, how to connect the target display to the PC, how to select the program destination (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 display module is commonly used as the example, but the procedure is the same for other displays.

PICASO Serial interface with the Microchip PIC18F45K20 Block Diagram



The flow of data in this setup begins with the PICASO display. The PICASO display generates a counting number that begins at zero and passes this to the PIC microcontroller. As the controller receives this stream of data is also sends it back to the display thereby creating a LOOPBACK of data. It will be noticeable during the test that there a significant delay in the data streaming since both of the controllers operate in its own time-base set by the oscillator. Although the microcontrollers have different sets of oscillator frequency they can be set to a particular baud rate that enable them to have correct data transmitted and received.

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.