APPLICATION NOTES

4D SYSTEMS
TURNING TECHNOLOGY INTO ART

Designer or ViSi I2C Connection to Wii Nunchuk

DOCUMENT DATE: **15th May 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 I2C communications port of the PICASO display modules. 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 uLCD-32PTU uLCD-43(P/PT/PCT)
  uLCD-28PTU uLCD-32WPTU uVGA-III

  and other superseded modules which support the Designer and/or ViSi environments.

- 4D Programming Cable or µUSB-PA5
- 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

Description **2**
Content **2**
Application Overview **3**
Setup Procedure **3**
Create a New Project **3**
Design the Project **4**
The Include Section ***4***
The main program ***4***
The Display Screen Setup ***4***
The repeat-forever detect loop ***4***
The sub-routines ***5***
Communication initialization sub-routine - Init() 5
The request for data sub-routine - request() 7
The receive data sub-routine – receive() 7
Editing the analogue and accelerometer data –edit() 8
The IF conditional loops ***9***
Drawing a colour filled circle using the analogue data 9
Drawing a colour filled circle using the accelerometer x/y data 9
When no buttons and both buttons are pressed 9
Running the Program **10**
Proprietary Information **12**

Disclaimer of Warranties & Limitation of Liability..... 12

Application Overview

This Application note is intended to demonstrating to the user the set-up, initialization and operation of the built-in I2C communications port of the PICASO Embedded Graphics Processor. For this project a Wii Nunchuk was utilized as a slave device. Data output that contains the slave device's analogue output and accelerometer output were read using I2C connection.

This application note also intends to explain the functionality and working of I2C, as well as some sample code that explains how I2C is implemented. Remember that the PICASO Inter-Integrated Circuit can only function as a master device.

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

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 PICASO Internal Functions Reference Manual and 4D Graphics Language Programmer's Reference Manual.

The Include Section

This project starts with the identification of the platform being used as declared by the #platform function. Also a colour related include file is added using the # inherit statement. It is followed by the declaration of constants and global variable which will be used in this application.

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


#constant device    0xA4
#constant speed     I2C_MED
var _data[6]:= [0x00,0x00,0x00,0x00,0x00,0x00];
var n,m;
```

The main program

The main program for this projects contains two sections: the initialization of the screen and the repeat-forever loop that shall be continuously run by the processor. In the sample program, several sub-routines are called in to perform a particular function. These functions include initialization of the I2C communication of the PICASO to the Wii Nunchuk I2C port, the reception of data sub-routine and an additional sub-routine to convert the data received to fit a certain segment of the screen for demonstration purposes. Details on the sub-routines will be presented in the succeeding sections of this document.

The Display Screen Setup

Aside from the assisted manner of setting up the screen mode of the display module, one of the PICASO internal function is also capable of doing so. The gfx_ScreenMode() function can set the screen to either a landscape or a portrait orientation or their reversed orientation.

```
func main()

    gfx_ScreenMode(LANDSCAPE);
    gfx_Cls();
    pause(100);
```

As part of the presentation in this application two other functions are added before the repeat-forever loop. The gfx_Cls() statement is used to clear the screen and subsequently, the delay of 100 milliseconds is inserted using the function pause().

The repeat-forever detect loop

The next segment of the program is the repeat-forever loop. This loop generally contains all the routines that would be run by the processor endlessly. For this program, the repeat-forever repeatedly calls on several sub-routines that were executed. Sub-routines related to I2C slave device initialization, the data acquisition from the slave device, and conditional loops are contained in the repeat forever.

```
repeat
    init();                     // call initialization function
    request();                  // call nunchuk data update
    receive();                  // call receive data from nunchuck
    edit();                     // edit values function to fit screen


if(_data[5]== 0)                // if both the C/Z button are pressed
    gfx_Cls();
endif
```

```
96  if(_data[5] == 3)                   // if C/Z are not pressed
97      txt_MoveCursor(3,38);           // start printing at (line,column)
98          print(" Wii Nunchuk Data");
99      txt_MoveCursor(7,38);           // start printing at (line,column)
100         print("JOYSTICK X ::");
101         print("\nJOYSTICK Y ::");
102         print("\nACCELERO X ::");
103         print("\nACCELERO Y ::");
104         print("\nACCELERO Z ::");
105         print("\nBUTTON C/Z ::");
106     txt_MoveCursor(6,52);
107     for(n:=0; n<=5; n++)
108         print("\n ", _data[n]) ;
109     next
110
111     for(n:=0; n <=272; n+=10)       // loop for drawing graph
112         gfx_Line(0,n,270,n, YELLOW);
113         gfx_Line(n,0,n,270, YELLOW);
114     next
115 endif
```

```
118 if(_data[5] == 1)                   // if the C button is pressed
119     txt_MoveCursor(6,50);           // start printing at (line,column)
120     for(n:=0; n<=5; n++)
121         print("\n   ", _data[n]) ;  // print the value in index n of _data
122     next
123     for(n:=0; n <=272; n+=10)       // loop for drawing graph
124         gfx_Line(0,n,270,n, YELLOW);
125         gfx_Line(n,0,n,270, YELLOW);
126     next
127     gfx_CircleFilled(_data[0] + 8,264-_data[1],3,BLUE); // draw circle with using
128                                                         // accelero x and y values
129 endif
130
131 if(_data[5] == 2)                   // if the Z button is pressed
132     txt_MoveCursor(6,50);           // start printing at (line,column)
133     for(n:=0; n<=5; n++)
134         print("\n   ", _data[n]) ;  // print the value in index n of _data
135     next
136
137     for(n:=0; n <=272; n+=10)       // loop for drawing graph
138         gfx_Line(0,n,270,n, YELLOW);
139         gfx_Line(n,0,n,270, YELLOW);
140     next
141     gfx_CircleFilled(_data[2],_data[3],3,RED); // draw circle with using
142                                                // accelero x and y values
143 endif
144 forever
```

This group of statements presents the repeat-forever segment of the application program. A brief explanation of the statements included shall be presented in the next sections.

The sub-routines

A good practice in avoiding confusion and reducing the codes placed under the main program is to use sub-routines. These sub-routines can be called upon directly and executed.

```
85  repeat
86      init();                         // call initialization function
87      request();                      // call nunchuk data update
88      receive();                      // call receive data from nunchuck
89      edit();                         // edit values function to fit screen
90
```

Referring to the statements above, we would notice that four sub-routines are called in succession. Each time a sub-routine is called in the processor executes the statements included therein. Sub-routines can be written to return particular global variable values.

Communication initialization sub-routine - Init()

Establishing communications over the I2C port requires several important information. These information includes the initialization procedure required by the slave device.

Initializing the PICASO I2C port to work with a slave device would need the slave address ID, register addresses and the data that is needed to be written to the slave device's register address.

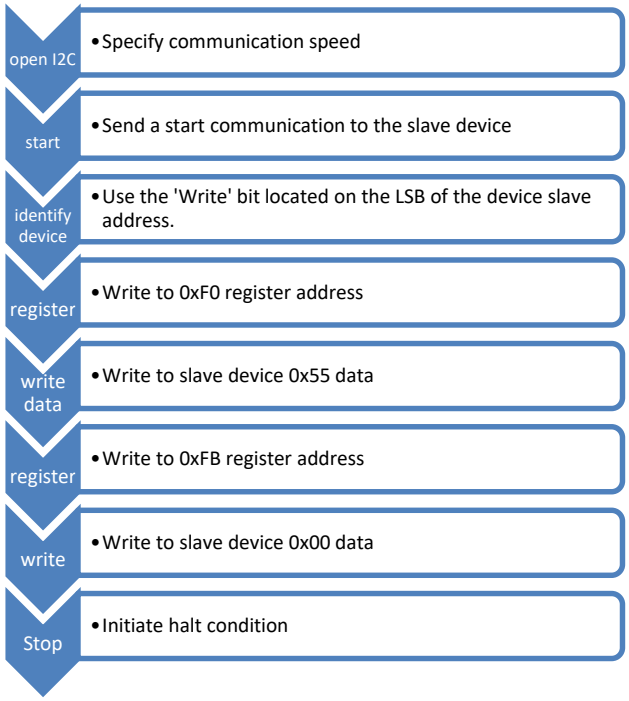
```
func init()
// ** note to the for initialization
// The data used herein is for third party Nunchuks
    I2C_Open(speed);
    I2C_Start();
    I2C_Write(device);
    I2C_Write(0xF0);
    I2C_Write(0x55);
    I2C_Write(0xFB);
    I2C_Write(0x00);
    I2C_Stop();

// The data used in this segment is for genuine Nunchuks
// Comment out the above initialization data and use the
// following statements
/*    I2C_Open(speed);
    I2C_Start();
    I2C_Write(device);
    I2C_Write(0x40);
    I2C_Write(0x00);
    I2C_Stop();
*/
endfunc
```

Looking at the statements above, the initialization start with opening the I2C port to a particular speed. The available speed speeds of transfer for the I2C are 100 KHz, 400 KHz and 1MHz. The PICASO embedded graphics processor can only operate as an I2C master. Hence, all other device which will connect to the I2C must be configured to be slave devices.

The statements displayed above means that the first step to perform is opening the I2C port using a particular communication speed and then initiate a 'start' condition. After the initiation of the start condition, send to the slave device its write address. The slave write address is 0xa4 while the read address is 0xA5 in hexadecimal.

Wii Nunchuk I2C device address							
1	0	1	0	0	1	0	Read : 1 Write : 0

Sending a 'write device address' to the slave would result to a hexadecimal value equivalent to 0xA4 and subsequently, a 'read device' will be equal to 0xA5. Slave devices have their own register addresses that are inherent to themselves. These registers, when written with a particular data shall perform a particular task. From the statements pasted on the previous column, we can see that for the initialization of the Wii Nunchuk a couple of hexadecimal are needed to complete the handshaking process. Data are written in right after the register address is identified. Referring to the statements on the left column the register 0xF0 is written with the 0x55 data, likewise the 0xFB register is written with the 0x00. Figuratively, we can use the flow chart below to summarize the handshake initiation for the PICASO and the Wii Nunchuk.

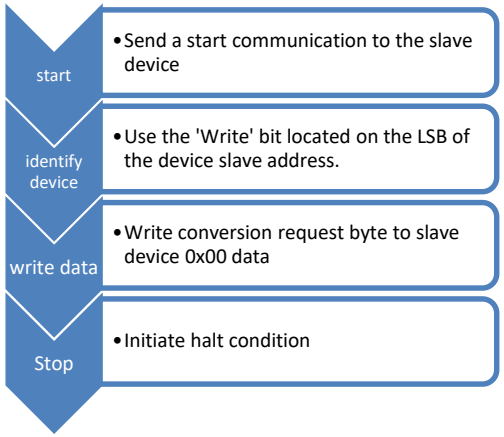


Handshaking of master and slave device is the most important amongst I2C communication setup procedure. Without fulfilling this requirement the transferring of data to and from the slave device will not be possible.

The request for data sub-routine - request()

After setting up the master-slave I2C handshake between the devices the next step is to send the request for the internal preparation of data within the Wii Nunchuk. The slave device awaits a request byte to be written following a device 'write' address byte. The statements below shows this request procedure. The slave device write address, as given from the previous section of this document, is 0xA5. After writing this to the slave, it is followed by a 0x00 byte. This byte directs the slave device to prepare the output data of the accelerometer, analogue output and the button status.

```
35  func request()
36      I2C_Start();
37      I2C_Write(device);
38      I2C_Write(0x00);
39      I2C_Stop();
40  endfunc
```



Again, putting this into a flow diagram will result to the following. Note that this sub-routine is primarily used to tell the Wii Nunchuk

The receive data sub-routine – receive()

Next sub-routine to understand is the receive routine. This section simply sends out the slave device address, with the read bit high, to the slave device. The slave read request is a bit set at the LSB of the slave address. Reading the information from the slave device requires that the LSB of the slave address will be set to a high bit to represent – 'reading mode of the slave address'.

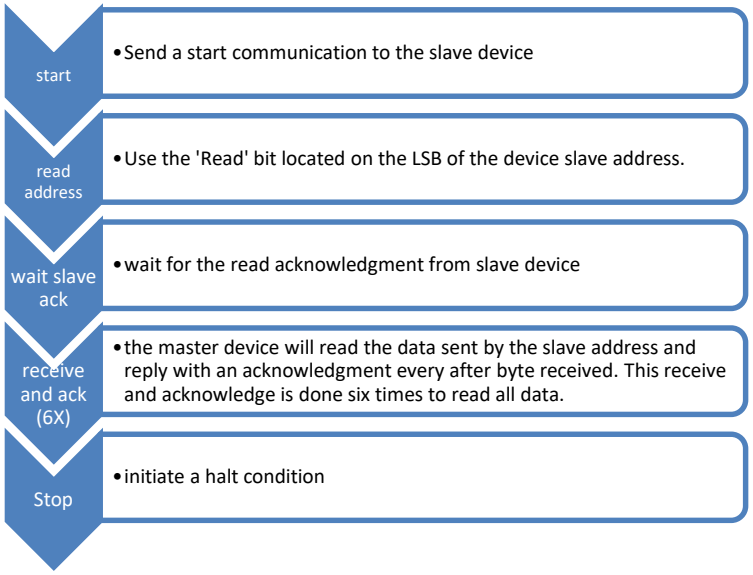
Wii Nunchuk I2C device address							
1	0	1	0	0	1	0	Read : 1

Hence, the resulting byte to be written over the I2C would be 0xA5. Below are the statements contained in the receive() sub-routine. Note that that the constant device has a value of (0xA4);

```
41
42  func receive()
43      var count:=0;
44  again:
45      I2C_Start();
46      I2C_Write(device + 0x01);
47      I2C_AckStatus();                // wait for data from Nunchuk
48      _data[0]:=I2C_Read();           // analog X
49      I2C_Ack();
50      _data[1]:=I2C_Read();           // analog y
51      I2C_Ack();
52      _data[2]:=I2C_Read();           // accelerometer x
53      I2C_Ack();
54      _data[3]:=I2C_Read();           // accelerometer y
55      I2C_Ack();
56      _data[4]:=I2C_Read();           // accelerometer z
57      I2C_Ack();
58      _data[5]:=I2C_Read();           // lower bits of acc and c/z button
59      I2C_Nack();
60      I2C_Stop();
61
62      // data correction using axis data 2 bit component from _data[5]
63      _data[2]:= ((_data[2] << 2) + ((_data[5] & 0x0F) >> 2));
64      _data[3]:= ((_data[3] << 2) + ((_data[5] & 0x30) >> 4));
65      _data[4]:= ((_data[4] << 2) + ((_data[5] & 0xC0) >> 6));
66      _data[5]:= _data[5] & 0x03;
67
68  endfunc
```

Before a 6-byte data is sent from the slave device to the master, an acknowledgment from the Wii Nunchuk is needed. The I2C_AckStatus() function is used to receive the acknowledgment from the slave device. Following receipt of the acknowledgment from the slave device a succession of data receive and acknowledgment is done by the master device, in this case the PICASO.

The flow diagram is the process in which the 6-bytes from the Wii Nunchuk is received by the PICASO.



The data received by the PICASO needs to be arranged at this point. The statements at the end of this sub-routine re-arranges the receive data to show the real data for the analogue, accelerometer and the buttons.

Data byte receive								Address
Joystick X								0x00
Joystick Y								0x01
Accelerometer X (bit 9 to bit 2 for 10-bit resolution)								0x02
Accelerometer Y (bit 9 to bit 2 for 10-bit resolution)								0x03
Accelerometer Z (bit 9 to bit 2 for 10-bit resolution)								0x04
Accel. Z bit 1	Accel. Z bit 0	Accel. Y bit 1	Accel. Y bit 0	Accel. X bit 1	Accel. X bit 0	C-button	Z-button	0x05

Byte 0x00 : X-axis data of the joystick

Byte 0x01 : Y-axis data of the joystick

Byte 0x02 : X-axis data of the accellerometer sensor

Byte 0x03 : Y-axis data of the accellerometer sensor

Byte 0x04 : Z-axis data of the accellerometer sensor

Byte 0x05 : bit 0 as Z button status - 0 = pressed and 1 = release

bit 1 as C button status - 0 = pressed and 1 = release

bit 2 and 3 as 2 lower bit of X-axis data of the accellerometer sensor

bit 4 and 5 as 2 lower bit of Y-axis data of the accellerometer sensor

bit 6 and 7 as 2 lower bit of Z-axis data of the accellerometer sensor

The end result of the receive() sub-routine returns the real values to the _data buffer array.

Editing the analogue and accelerometer data –edit()

This subroutine is intended to format the received accelerometer data to fit the dimensions of the display module used in this application project. This sub-routine was written to provide a better visual presentation of the accelerometer data.

```
func edit()
   // editting of data to fit size of screen for output
  _data[2]:=400 - (_data[2] >> 1);// edit the output to screen position
  _data[3]:=400-(_data[3] >> 1);  // edit the output to screen position
endfunc
```

Here, the _data index 2 and 3 are shifted to the right by a single bit and then subtracted from a value of 400. The resulting value of this subroutine replaces the real value in the same index positions of _data array.

The IF conditional loops

Being able to communicate with the slave device Wii Nunchuk, simple IF-condition loops that displays the output value are included in this application. This IF-conditional loop are made in reference with the buttons pressed. The Wii Nunchuk has two buttons available. Using these buttons, a total of four states can be made.

Drawing a colour filled circle using the analogue data

The analogue data described from the previous sections of this document is used to provide a centre point for a colour filled circle. This conditional loop is dependent on the C button of the Wii Nunchuk. Referring to the program statements below, we could see that whenever the button pressed is pressed and results to a 0x01 value in _data[5].

```
114
115 if(_data[5] == 1)                    // if the C button is pressed
116     txt_MoveCursor(6,50);           // start printing at (line,column)
117     for(n:=0; n<=5; n++)
118         print("\n    ", _data[n]) ;  // print the value in index n of _data
119     next
120     for(n:=0; n <=272; n+=10)        // loop for drawing graph
121         gfx_Line(0,n,270,n, YELLOW);
122         gfx_Line(n,0,n,270, YELLOW);
123     next
124     gfx_CircleFilled(_data[0] + 8,264-_data[1],3,BLUE); // draw circle with using
125                                                         // accelero x and y values
126 endif
127
```

With the value of _data[5] being able to comply with the condition then the drawing of the horizontal and vertical lines together with the other statements inside the conditional loop are executed.

Drawing a colour filled circle using the accelerometer x/y data

Likewise the accelerometer x and y, is used to draw a colour filled circle. The centre point of the circles denotes the current position of the Wii Nunchuk. Again, the values used herein are for presentation purposes. Referring to the statements on the next column, we can see horizontal and vertical line drawn to the screen display. The numerical data is also displayed on the side. All these statements are executed only if the Z button is pressed.

```
128 if(_data[5] == 2)                    // if the Z button is pressed
129     txt_MoveCursor(6,50);           // start printing at (line,column)
130     for(n:=0; n<=5; n++)
131         print("\n    ", _data[n]) ;  // print the value in index n of _data
132     next
133
134     for(n:=0; n <=272; n+=10)        // loop for drawing graph
135         gfx_Line(0,n,270,n, YELLOW);
136         gfx_Line(n,0,n,270, YELLOW);
137     next
138     gfx_CircleFilled(_data[2],_data[3],3,RED); // draw circle with using
139                                                // accelero x and y values
140 endif
```

A press on the Z button will result to a value equal to 0x02. If the condition is fulfilled, then the statements include therein are executed.

When no buttons and both buttons are pressed

In the event that there are no buttons pressed, this will result to a value equal to 0x03. This is the result of the received data over I2C. When this condition is fulfilled, the drawing of horizontal and vertical lines are continually executed.

```
 93 if(_data[5] == 3)                    // if C/Z are not pressed
 94     txt_MoveCursor(3,38);           // start printing at (line,column)
 95         print(" Wii Nunchuk Data");
 96     txt_MoveCursor(7,38);           // start printing at (line,column)
 97         print("JOYSTICK X ::");
 98         print("\nJOYSTICK Y ::");
 99         print("\nACCELERO X ::");
100         print("\nACCELERO Y ::");
101         print("\nACCELERO Z ::");
102         print("\nBUTTON C/Z ::");
103     txt_MoveCursor(6,52);
104     for(n:=0; n<=5; n++)
105         print("\n ", _data[n]) ;
106     next
107
108     for(n:=0; n <=272; n+=10)        // loop for drawing graph
109         gfx_Line(0,n,270,n, YELLOW);
110         gfx_Line(n,0,n,270, YELLOW);
111     next
112 endif
113
```

The set of conditions included in this conditional loop clears the screen using the gfx_Cls() function. The screen is cleared only if a result of value in _data[5] is equal to the 0x00. This is the event that both of the C and Z buttons are pressed.

```
89  if(_data[5]== 0)                    // if both the C/Z button are pressed
90      gfx_Cls();
91  endif
```

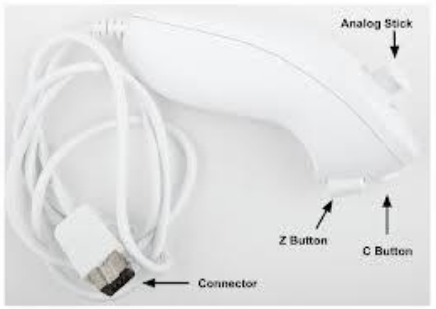
Running the Program

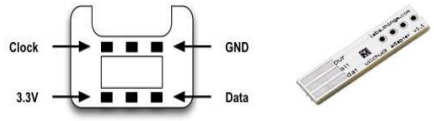
Making the program work is fairly simple. Download the program into the display module and connect the slave device to the master. The images shows the correct connection of the I2C to the display device. Note that the SCLs and SDAs must be coupled together for both devices.

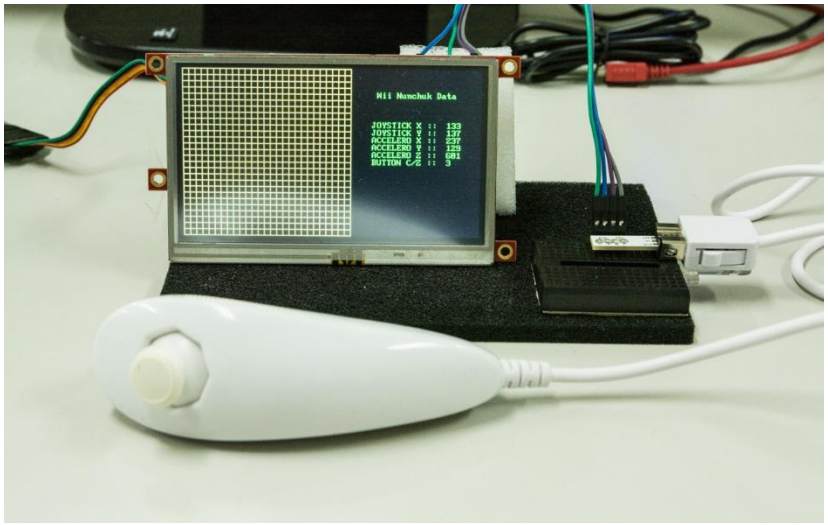
Wiring Connections of Wii Nunchuk to PICASO

Using the Wii Nunchuk terminal adapter, seen in the images here, the connection to the Wii Nunchuk can be easily connected to the PICASO display module. A set of header pins and jumper wire will be very useful for temporary connections.

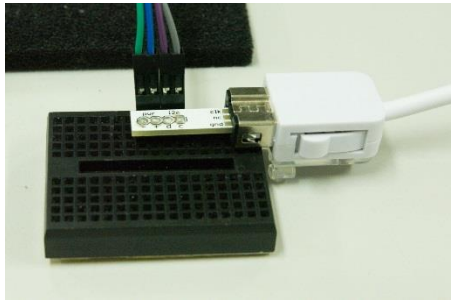
For the succeeding parts of this section, a simplified wiring connection discussion will be presented. Also, included in the next sections are the axial reference for the Wii Nunchuk and connection pin-outs. Below is the terminal pin-out for the WII Nunchuk.





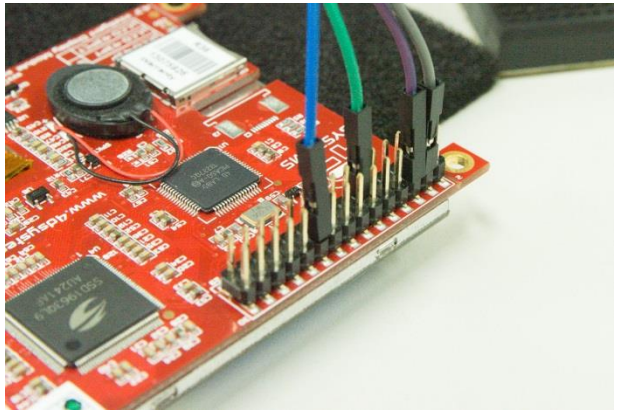


Above is the complete display and Wii Nunchuk project. Notice that the interface between the two devices is quite simple. There are simply four connections needed in this application project, namely: SCL, SDA, VCC, and GND. Below we can see how the Wii Nunchuk port is connected to the adapter.

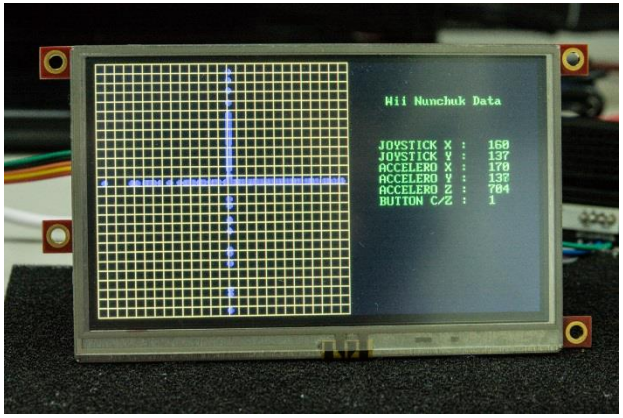


Wii Adapter	PICASO (30 way header)	Wire colour
+	3.3 volts	Blue
-	GND	Green
SDA	PIN 3	Violet
SDC	PIN 2	Grey

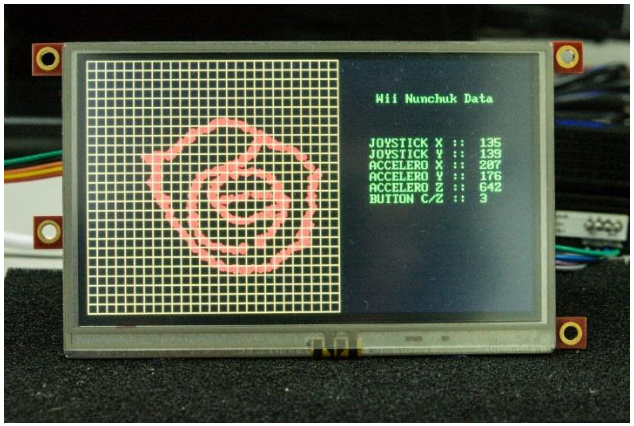
Following the pin match-up table for the Wii Adapter and the PICASO 30 way header pins connections, the connections must be similar to the one below.



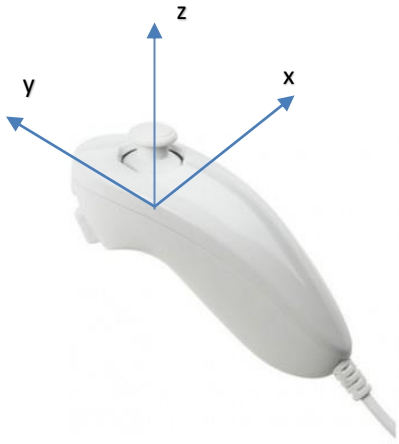
Here are some photos that show the result of the project. When the analogue signal from the joystick is used, we need to press the C button to activate the drawing of circles.



Likewise, pressing the Z button enables drawing of the circle using the accelerometer as reference for the X and Y value.



Below is the axial reference for the Wii Nunchuk.



Proprietary Information

Disclaimer of Warranties & Limitation of Liability