UM1064
User manual

Software guide for Unico Lite

Introduction

The Unico Lite graphical user interface (GUI) is a complete demonstration software which provides the source code to show how to manage sensor data-flow from generic MEMS sensors (such as accelerometers and gyroscopes).

This user manual describes all the components of the Unico Lite GUI. For details regarding the commands or data format please refer to the UM0979 user manual.

Contents

1 PC system requirements

The Unico Lite software has been designed to operate with Microsoft® Windows platforms and is written with Microsoft® Visual Studio 2008.

2 Unico Lite graphical user interface

The Unico Lite graphical user interface is a simple Windows form application written in C# (.NET Framework 3.5) designed to show basic operation such as board connection, read/write register sensors and how to acquire continuous data from the eMotion board (data-flow).

The basic concepts described below are suitable for different sensors. In this case, the source code can be used directly with two different STEVAL boards:

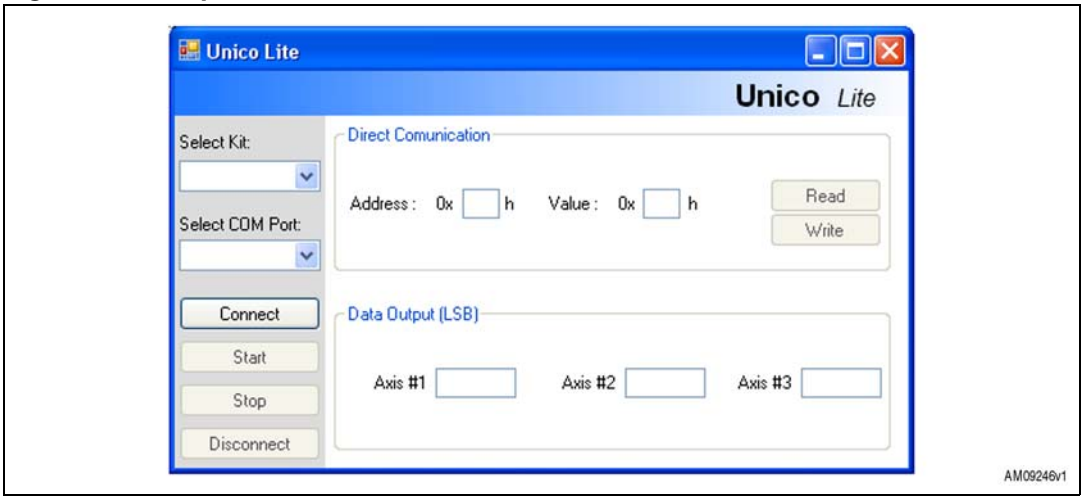
1. STEVAL-MKI105V1 for LIS3DH 3 axes digital accelerometer
2. STEVAL-MKI107V1 for L3G4200D 3 axes digital gyroscope

However, it is extremely easy to add new STEVAL boards for other gyroscopes, accelerometers, and modules.

To execute the Unico Lite software GUI:

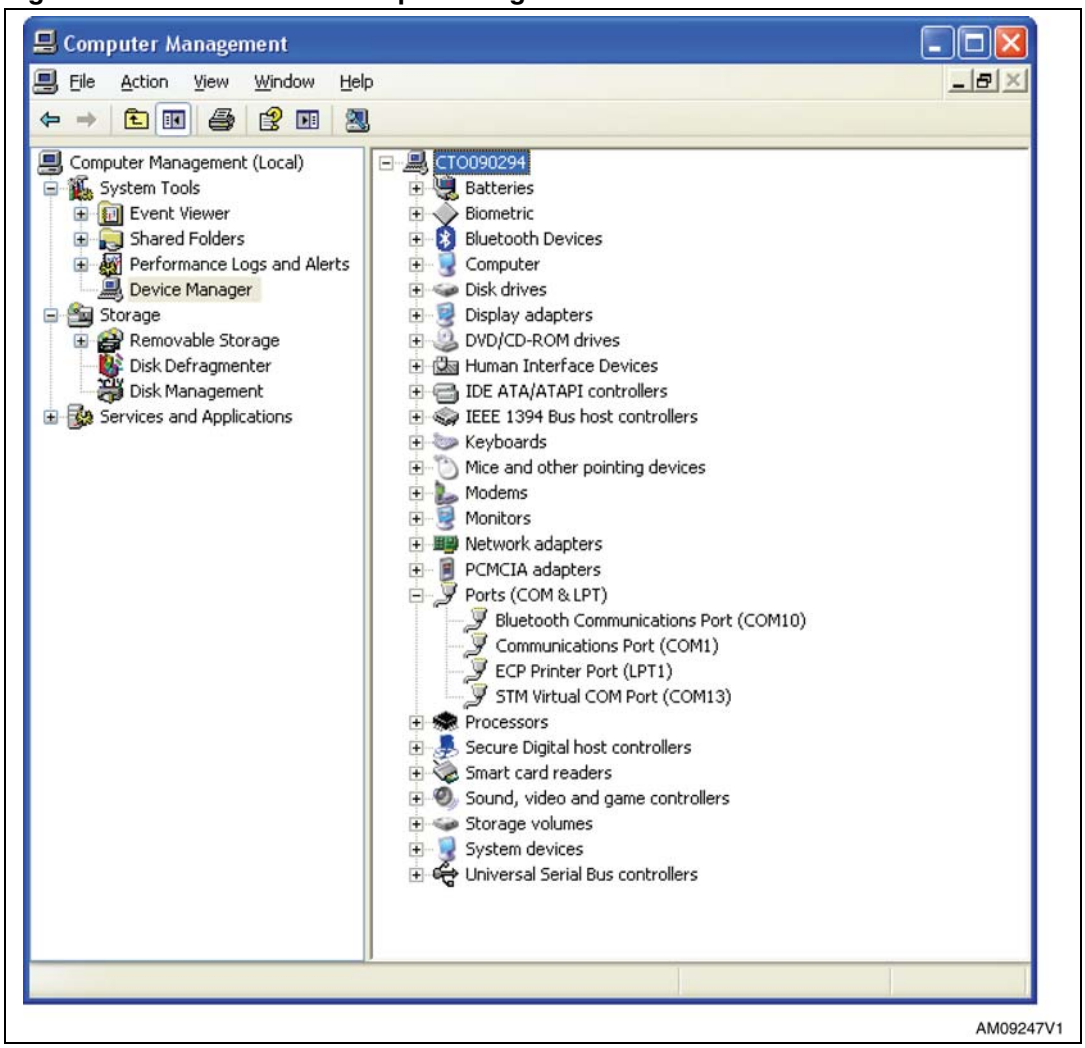
1. Plug the board into the PC through a USB port
2. Click on Unico_Lite
3. The GUI window appears as shown in *Figure 1*

Figure 1. Graphical user interface



4. Select the COM port currently in use from the list. In order to check which COM port has been assigned to the board, right click on “My Computer” and select “Manage”, select “Device Manager”, scroll through the list until “Ports(COM & LPT)” and look for “STM Virtual COM Port”. In the following example (*Figure 2*), COM13 has been assigned to the board.

Figure 2. Virtual COM driver port assignment



5. Select the demonstration board currently in use
6. Click “Connect”

Now it is possible to use the GUI, in order to:

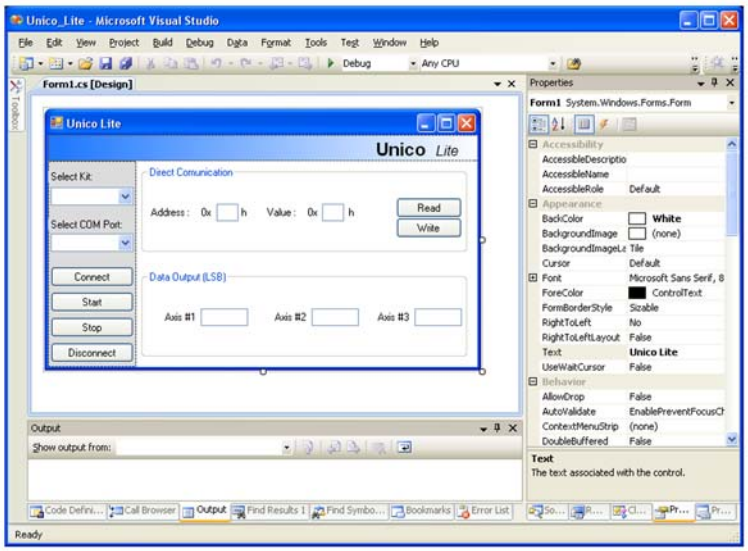
1. Read the register: insert the register address in the address box (hexadecimal value) and click the “Read” button. The register content is shown in the value box.
2. Write the register: insert the register address in the address box and insert the register value in the value box (hexadecimal value), and click the “Write” button.
3. Get continuous data: click the “Start” button and check sensor data.

Note: For the continuos data, please check the register settings just to have the sensor in normal mode or low power mode. In the case of power-down configuration, no data is shown in the boxes because no data comes from the sensor.

3 Windows form design

The first step is to design the form application through Microsoft Visual Studio 2008 (*Figure 4*).

Figure 3. Main screen



To design the Unico Lite GUI (*Figure 4*) the following controls have been used:

Figure 4. Unico Lite controls

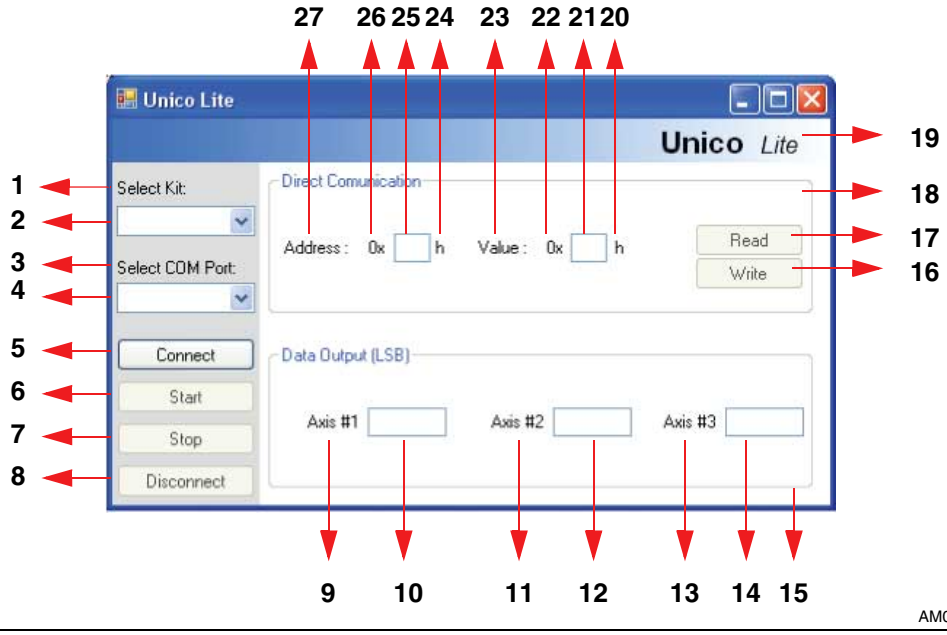


Table 1. ????

Ref.	Control type	Name	Text	Initial status
1	Label		Select COM port	-
2	ComboBox	CBX_Kit	-	Enabled
3	Label		Select COM port	-
4	ComboBox	CBX_ComPort	-	Enabled
5	Button	BTN_Connect	Connect	Enabled
6	Button	BTN_Start	Start	Disabled
7	Button	BTN_Stop	Stop	Disabled
8	Button	BTN_Disconnect	Disconnect	Disabled
9	Label		Axis #1	-
10	TextBox	TB_Val1	-	Enabled
11	Label		Axis #2	-
12	TextBox	TB_Val2	-	Enabled
13	Label		Axis #3	-
14	TextBox	TB_Val3	-	Enabled
15	GroupBox	GroupBox2	Data output (LSB)	-
16	Button	BTN_Write	Write	Disabled
17	Button	BTN_Read	Read	Disabled
18	GroupBox	GroupBox1	Direct communication	-
19	PictureBox	PictureBox1		-
20	Label	-	h	-
21	TextBox	TB_Value		Enabled
22	Label	-	0x	-
23	Label	-	Value:	-
24	Label	-	h	-
25	TextBox	TB_Address		Enabled
26	Label	-	0x	-
27	Label	-	Address:	-

4 Code

4.1 Constants

`const int MAX_COM_PORTS`

defines the max. COM ports shown in the CBX_ComPort. In this case the standard value is 30.

`const int MAX_KITS`

defines the max. number of kits supported by the code. In this version the software supports 2 kits.

```
public struct Kits
{
    public string Name;
    public int words;
    public string setdb;
    public string read;
    public string write;
}
```

defines a single kit that can be described by:

a) name: STEVAL kit code

b) words: number of data bytes coming back after *start command (please refer to Table 4 in Section 4 of the UM0979 user manual)

c) setdb: selects the part of the firmware able to handle the adapter board (e.g. for STEVAL-MKI105V1 the command is *setdb105v1)

d) read: for different types of sensor (accelerometer, gyroscope, magnetometer, etc.) the board performs different kinds of commands

e) write: for different types of sensors (accelerometer, gyroscope, magnetometer, etc.) the board performs different kinds of command

4.2 Variables

`Kits[] MKI = new Kits[MAX_KITS];`

creates a struct array for kit description.

`int Port_Index`

global value for the ComboBox COM port index (default value = -1).

`int Kit_Index`

global value for the ComboBox kit index (default value = -1).

`string Val_Txt_1,Val_Txt_2, Val_Txt_3`

temporary string for data value.

```
string DataReadFromSerialPort
```

temporary string for data read.

```
bool Start_Done
```

flag used to monitor whether the “Start” button has been pressed or not.

4.3 Classes

```
public SerialPort c_Serial;
```

this class is used to control a serial port file resource. This class provides synchronous and event-driven I/O, access to pin and break states, and access to serial driver properties.

4.4 Initialization

Is the code in the constructor function wrote to initialize controls and classes.

```
public Form1()
{
    InitializeComponent();
```

standard C# declaration to initialize controls

```
    c_Serial = new SerialPort();
```

standard C# initialization for a new instance of the SerialPort class

```
    for(int i=0; i<MAX_COM_PORTS; i++)
    {
        CBX_ComPort.Items.Add("COM" + Convert.ToString(i+1));
    }
```

Fill CBX_ComPort ComboBox with COM ports available (in this example from COM1 to COM30)

```
    MKI[0].Name = "MKI105V1";
    MKI[0].words = 9;
    MKI[0].setdb = "*setdb105v1";
    MKI[0].read = "*r";
    MKI[0].write = "*w";

    MKI[1].Name = "MKI107V1";
```

```
MKI[1].words = 8;
MKI[1].setdb = "*setdb107v1";
MKI[1].read = "*gr";
MKI[1].write = "*gw";
```

Initialize struct kit with two kits, STEVAL-MKI105V1 and STEVAL-MKI107V1

```
for (int i = 0; i < MAX_KITS; i++)
{
    CBX_Kit.Items.Add(MKI[i].Name);
}
```

Fill CBX_Kit ComboBox

```
}
```

4.5 Connection

This is the piece of code in charge of managing serial port operation (like connect, disconnect, open and close) and button iterations.

4.5.1 Connect

The target of this function is to configure the serial port object and open the connection to the demonstration board. In detail, the port settings are:

1. PortName: the one selected with CBX_ComPort
2. BaudRate: 115200
3. DataBits: 8
4. ReadTimeout: 500 [ms]
5. WriteTimeout: 500 [ms]
6. DataReceived: connection between Event and Method. Each time a new data is sent on the serial port from the STEVAL-board the main thread calls the Method (in this case *USB_DataReceived()*).

After configuration the software tries to open the Com port selected, if no exception occurs the software writes on the serial port:

1. “*setdb*xxx*v1” (*xxx* is the STEVAL code, in this case 105 for LIS3DH and 107 for L3G4200D)
2. “*zoff”

Note: These two commands are mandatory for the connection.

At the end of the function, the software performs Enable/Disable controls.

4.5.2 Start

Enable/disable controls and sends the *start* command. The *Start_Done* flag is set to true, which means that the user clicks the “Start” button.

4.5.3 Stop

Enable/disable controls and sends the *stop* command. The *Start_Done* flag is set to false, which means that the user clicks the “Stop” button.

4.5.4 Disconnect

Sends the “*zon” command, closes the serial port and enables/disables controls.

4.6 Event

Each time a new data is written on the serial port the function *USB_DataReceived()* is called automatically from the main thread. The target of this function is to read the character on the serial port and call *String_Vector_Decode()* only if the character read is equal to “s” (START CHAR, please refer to UM0979).

4.7 Decode

The main purpose is to get sensor data from the data stream and visualize it. The data stream is a sequence of characters such as:

... s t xh xl yh yi zh zl i1 i2 s \r \n s t xh xl yh yi zh zl i1 i2 s \r \n s t xh xl yh yi zh zl i1 i2 s \r \n.

and the first step is to organize data in strings without START CHARS (“s” and “t”) and END CHARS (“\r” and “\n”, that are carriage return and new line).

...

(string #1) xh xl yh yi zh zl i1 i2 s

(string #2) xh xl yh yi zh zl i1 i2 s

(string #3) xh xl yh yi zh zl i1 i2 s

...

Before starting to decode the stream, it is necessary to make a second check on the character “t”. Prior to launching this feature, character “s” is firstly checked followed by the “t” character: if both characters have been received in sequence it means that the incoming string is correct and it is then possible to start the decoding.

Decode, as already mentioned, is done by the *Manage_Input_Buffer()* function after reading the input buffer; the function returns with an ordered 2-dimensional byte array and the number of complete strings.

With this kind of array and the number of the complete string, it is easy to reconstruct sensor data and convert it from 2’s complement to magnitude and sign; after this, data is ready to be shown in the right TextBox.

4.8 Buttons

4.8.1 Read

This performs the read register function. In detail, a stop command is sent just to stop, eventually, the data stream, after this, the software gets the address register from TB_Address TextBox and composes the command. MKI read is used because different kinds of sensors use different kinds of read/write commands, in this case there are two kits (accelerometer and gyroscope) and the complete commands are:

- Accelerometer: *rAA
- Gyroscope: *grAA

After the command is sent, the board replies with the register content, an example of the string format is: RAAhCCh

where AA is the address and CC is the content. The code is:

```
int pos1 = DataReadFromSerialPort.IndexOf('h');
int pos2 = DataReadFromSerialPort.IndexOf('h', pos1 + 1);
int pos3 = DataReadFromSerialPort.IndexOf('\r');
int pos4 = DataReadFromSerialPort.IndexOf('\r', pos3 + 1);
string MSB = DataReadFromSerialPort.Substring(0, 1);
string LSB = DataReadFromSerialPort.Substring(1, 1);

if ((pos2 - pos1) == 3) /*New firmware case */
{
    MSB = DataReadFromSerialPort.Substring((pos2 - 2), 1);
    LSB = DataReadFromSerialPort.Substring((pos2 - 1), 1);
}

TB_Value.Text = MSB + LSB;
```

is able to get the CC value and compose it in order to show in the TB_Value TextBox.

4.8.2 Write

This is similar to the read function, but in this case the software has only to send a write command, composed by a prefix, address, and new register value.

5 Revision history

Table 2. Document revision history

Date	Revision	Changes
21-Apr-2011	1	Initial release.