Micromega Corporation

Application Note 42

Drawing Graphs on a Serial Graphic Display

Introduction

This application note describes how the uM-FPU V3.1 floating point coprocessor can be used to drive a serial graphic display. It offloads all of the graphics overhead from the microcontroller, so that virtually any microcontroller can display complex graphs on a serial graphic display – even microcontrollers with limited available resources. The serial graphic display is connected directly to the FPU which can drive the display at up to 115,200 baud. The microcontroller communicates with the FPU using an SPI or I^2C interface. The user-defined functions described in this application note take care of all of the graphics overhead, including automatic scaling, layout and display of both line graphs and histograms.

Background

Drawing a graph with a desktop application like Microsoft Excel can seem very simple. A list of data points is entered, the chart type is selected, then Excel automatically scales and displays the data. Drawing a graph using a microcontroller is much more difficult. Serial graphic displays generally provide a set of commands for drawing simple objects such as points, lines, boxes, circles and text strings, but it's up to the user to figure out how to use these commands to draw more complicated objects such as line charts or histograms. The user needs to write a program to determine the scaling factors for the data, draw and annotate each axis, add grid lines or tick marks, then scale and display the data. This can require a large amount of code space, data space, and execution time. The microcontroller may have limited available resources or only be capable of driving the serial graphic display at a low baud rate.

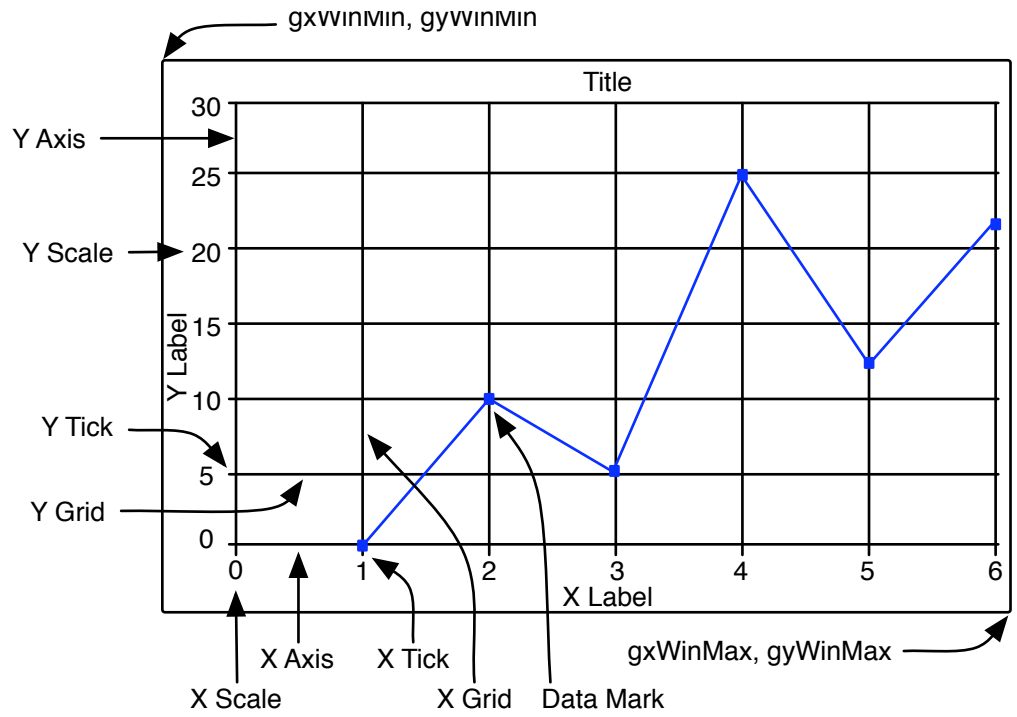
The uM-FPU V3.1 chip can solve these problems by offloading all of the graphics overhead. The microcontroller sends the data points to the FPU using a standard SPI or I^2C interface. The serial graphic display is connected directly to the FPU, which can drive the serial graphic display at speeds up to 115,200 baud. Using the user-defined functions described in this application note, the task of graphing data from a microcontroller can be as simple as the Excel example. The data points are sent to the FPU chip, then the FPU handles all of the tasks to automatically scale the data, and display the graph on a serial graphic display.

Displaying Graphs using the uM-FPU V3.1

A graph is a diagram that shows the relationship between a series of data points. For a two-dimensional graph, each data point has an *X* and *Y* value. For the graphs described in this application note, the *X* values are assumed to be sequential and occur at a fixed interval. The *X* value for each data point is calculated based on the sequential number of the data point and the values specified by the *xUnits*, *xOffset*, and *xScale* registers. The units on the *X* axis can be integer numbers, floating point numbers, time values, frequency values, or units of pi. The *Y* values can be any floating point value, and are stored in an array on the FPU by calling the *storePoint* function. Once the data points have been stored, the *drawLineGraph* function is then used to display a diagram with a line drawn between each data point on the graph, or the *drawHistogram* function is used to display a diagram with a rectangle drawn from the the *X* axis to the *Y* value of each data point on the graph.

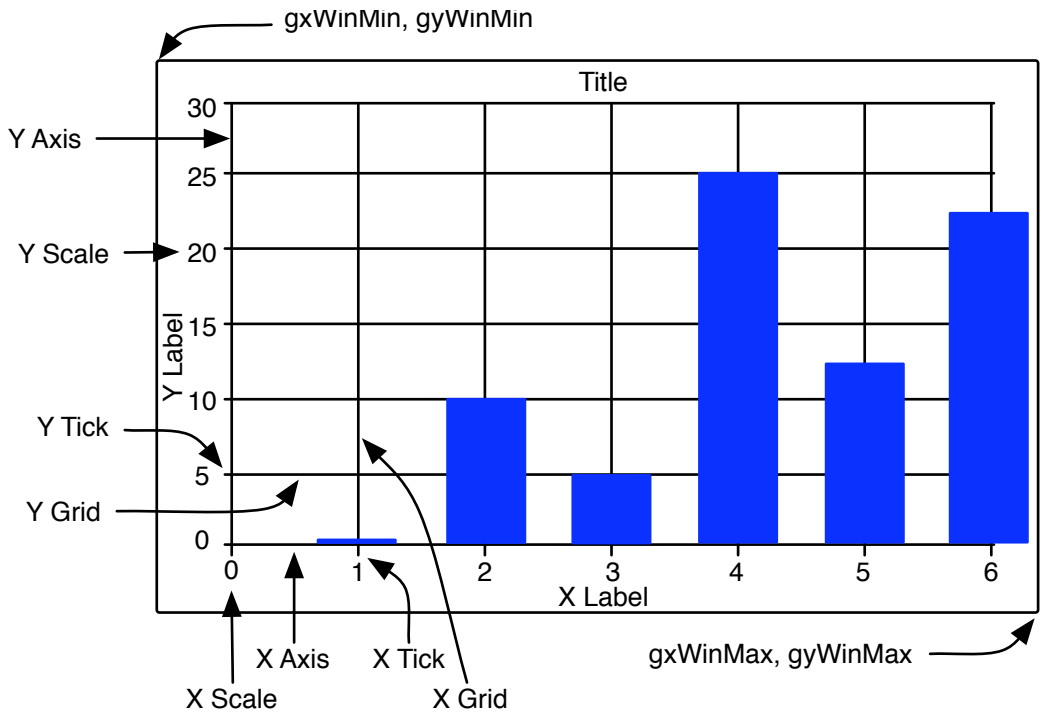
Components of a Line Graph

The following diagram show the various components of a line graph.



Components of a Histogram

The following diagram show the various components of a histogram.



Specifying the Format of the Graph

The type of graph displayed is specified by setting various register that determine the characteristics of the graph. The *initGraph* routine sets default values for all of these registers, but you can customize the graph by setting the registers to alternate values. The *DrawLineGraph* and *DrawHistogram* routines use the same registers. The following registers are used:

formatX, formatY	basic format of the graph
gxWinMin, gyWinMin, gxWinMax, gyWinMax	define the size of the graph window
fillColor, axisColor, dataColor	define the display colors
xUnits, xOffset, xScale, xTick	define the X axis units and scale
yMin, yMax	define the Y value range

formatX

This register determines the format of the X axis and data point markers. The action is enabled if the bit is 1, or disabled if the bit is 0.

Bit	7	6	5	4	3	2	1	0
	Mark	Zero	Label	Manual	Scale	Grid	Tick	Axis

Bit 0	draw X axis
Bit 1	draw X tick mark
Bit 2	draw X grid
Bit 3	draw X scale annotations
Bit 4	manually scale X axis
Bit 5	reserve space for X axis label
Bit 6	start X scale at zero
Bit 7	display a mark at each data point on line graph
Bits 8-31	unused

formatY

This registers determine the format of the Y axis, title and border. The action is enabled if the bit is 1, or disabled if the bit is 0.

Bit	7	6	5	4	3	2	1	0
	Border	Title	Label	Manual	Scale	Grid	Tick	Axis

Bit 0	draw Y axis
Bit 1	draw Y tick mark
Bit 2	draw Y grid
Bit 3	draw Y scale annotations
Bit 4	manually scale Y axis
Bit 5	reserve space for Y axis label
Bit 6	reserve space for Title
Bit 7	display border around the display window
Bits 8-31	unused

The default value for formatX and formatY is 0x8B. This draws a graph with the following format:

- a border is draw around the display window
- the X axis has tick marks and scale
- the Y axis has tick marks and scale
- automatic scaling of X and Y values
- no title or axis labels are drawn
- a mark is displayed at each data point on line graph

gxWinMin, gyWinMin, gxWinMax, gyWinMax

These registers are used to specify the display window for the graph. By default, the *initGraph* routine sets these registers to select the entire graphic display. These registers can be changed by the user to specify a different display window. When the *clearGraph*, *drawLineGraph* and *drawHistogram* routines are called, the values for the graph window are checked and adjusted as follows:

If *gxWinMin* < minimum X value for display, then *gxWinMin* = minimum X value
If *gxWinMax* > maximum X value for display, then *gxWinMax* = maximum X value
If *gxWinMax* <= *gxWinMin*, then *gxWinMax* = minimum X value and *gxWinMax* = maximum X value
If *gyWinMin* < minimum Y value for display, then *gyWinMin* = minimum Y value
If *gyWinMax* > maximum Y value for display, then *gyWinMax* = maximum Y value
If *gyWinMax* <= *gyWinMin*, then *gyWinMax* = minimum X value and *gyWinMax* = maximum Y value

If the graph window is still too small for the format specified, an X will be displayed through the graph window.

fillColor, axisColor, dataColor

These registers specify the color to be used to display the graph. The background color for the display window is specified by `fillColor`. The axis, annotations and labels are displayed using `axisColor`. Data points, lines, and the histogram are displayed using `dataColor`. The format of these values depends on the particular graphic display being used. The *initGraph* function sets default values for each color.

xUnits

This register specifies the type of scale used for the X axis. The values are as follows :

0 **Long Integer**
The scale values are displayed as long integer values.

1 **Time (Long Integer)**
The scale values are displayed as time values. The long integer value is interpreted as time in seconds. The value is displayed as HH:MM:SS, with leading zeros are suppressed.
e.g. a value of 100 would be displayed as 1:40.

2 **Floating Point**
The scale values are displayed as a floating point values with leading and trailing zeroes suppressed.

3 **Pi (Floating Point)**
The scale value are displayed as multiples of pi.
e.g. the value 6.2831853 would be displayed as 2pi.

4 **Frequency (Floating Point)**
The scale values are displayed as frequency values. If the floating point value is less than 1000 the value is displayed in hertz, otherwise the value is displayed in kHz.
e.g. the value 2000.0 would be displayed as 2kHz.

default value: 0

xOffset

This register specifies the offset for the X value. The data points are numbers starting at zero. The `xOffset` value is added to the data point number and the result is multiplied by `xScale`. For example, to display sequential X scale values from -8 to 8, `xOffset` is set to -8.

default value: 0

xScale

This register specifies a scale multiplier for the X value. It must be a long integer value if `xUnits` is 0 or 1, and a floating point value if `xUnits` is 2, 3 or 4.

default value: 1 (long integer)

xTick

This register specifies the number of data points between ticks or grid marks.

default value: 1

yMin, yMax

If automatic scaling of the Y axis is enabled, these registers will be set automatically. If manual Y axis scaling is enabled, these registers specify the range for the Y values.

default values: -1.0, 1,0

Storing Data Points

Each data point on the graph has an *X* and *Y* value. The *X* values are assumed to be sequential and occur at a fixed interval. The *X* value for each data point is calculated based on the sequential number of the data point and the values specified by the *xUnits*, *xOffset*, and *xScale* registers. The units on the *X* axis can be integer numbers, floating point numbers, time values, frequency values, or units of pi. The *Y* values can be any floating point value, and are stored in an array on the FPU by calling the *storePoint* function. The maximum number of data points that can be stored in the array on the FPU is specified by the *maxPoints* register (up to a maximum of 65). If the *storePoint* function is called with more data points then the value specified by *maxPoints*, only the last number of data points (as specified by *maxPoints*) will be saved.

Using a Different Serial Graphic Display

This application note uses the ezLCD-002 serial graphic display. Other serial graphic display could be used by modifying the uM-FPU V3.1 user-defined functions supplied with this application note to accommodate the graphic primitives of a different display. Only the low-level routines that initialize the display, draw lines, draw rectangles, draw text and specify colors would need to be modified to use a different display.

Interfacing at Higher Baud Rates

Many serial devices, like the ezLCD-002 graphic display used in this application note, require serial connections at high baud rates. This can be a problem for microcontrollers that have limited resources. For example, the PICAXE has a maximum baud rate of 4800 baud. This limitation can easily be overcome by connecting the serial device to the uM-FPU V3.1 chip and routing the serial data through the FPU. The FPU can perform serial input and output at up to 115,200 baud. Data is sent to the FPU over an SPI or I^2C interface, then sent to the serial device using the `SEROUT` instruction. This is the method used by the examples in this application note to allow the PICAXE and BASIC Stamp to draw graphs on the ezLCD-002 display. The same method could also be used in other applications that require interfacing to serial devices at high baud rates.

Sample Screen Shots

Figure 1 and 3 show the default formats for a line graph and histogram. Figure 2 is the same graph with the X and Y grid enabled. Figure 4 uses a smaller display window and includes a border, title and axis labels. Figure 5 shows an example that displays two line graphs. The top graph shows the cosine function and the bottom graph shows a multiplier function. A title is displayed for each graph, and the *xUnits* are set for pi scaling, and *xOffset* is set to display both positive and negative X values. Figure 6 shows the line graph obtained when the two functions in Figure 5 are multiplied together.

Figure 1
Line Graph using full display.

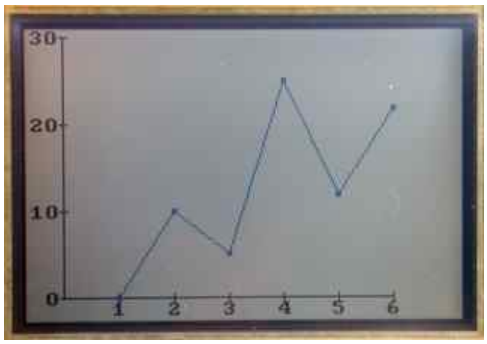


Figure 2
Line Graph with X and Y grid enabled.

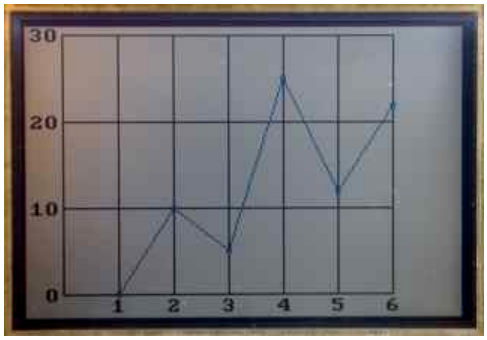


Figure 3
Histogram using full display.

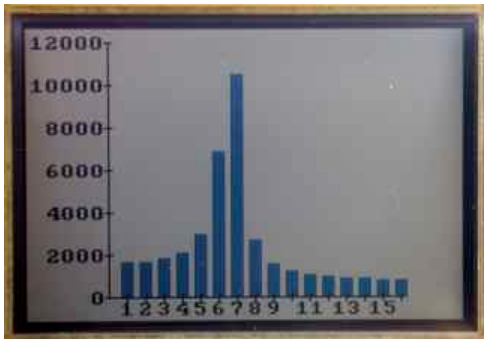


Figure 4
Line Graph using smaller display window with border, title and axis labels.

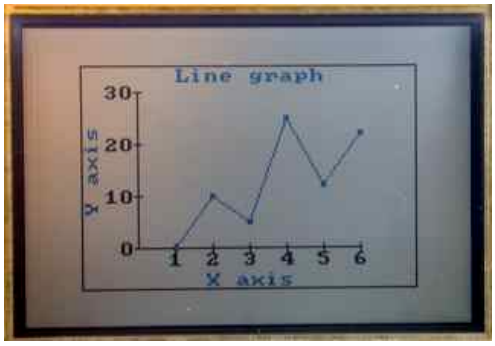


Figure 5
Two Line Graphs with pi units on X axis.

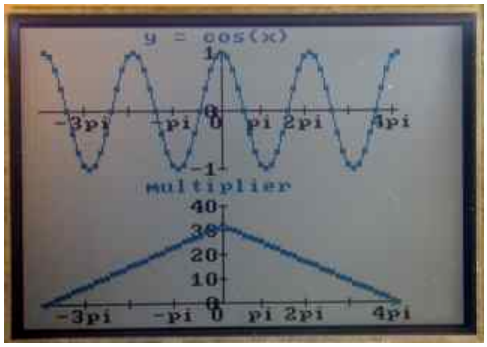
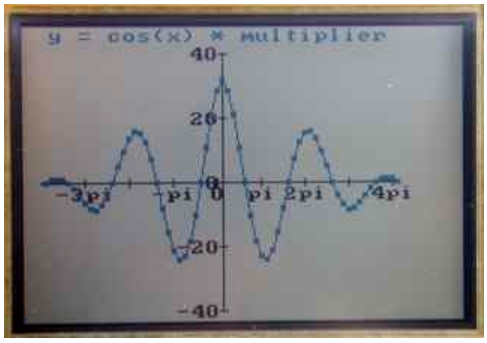
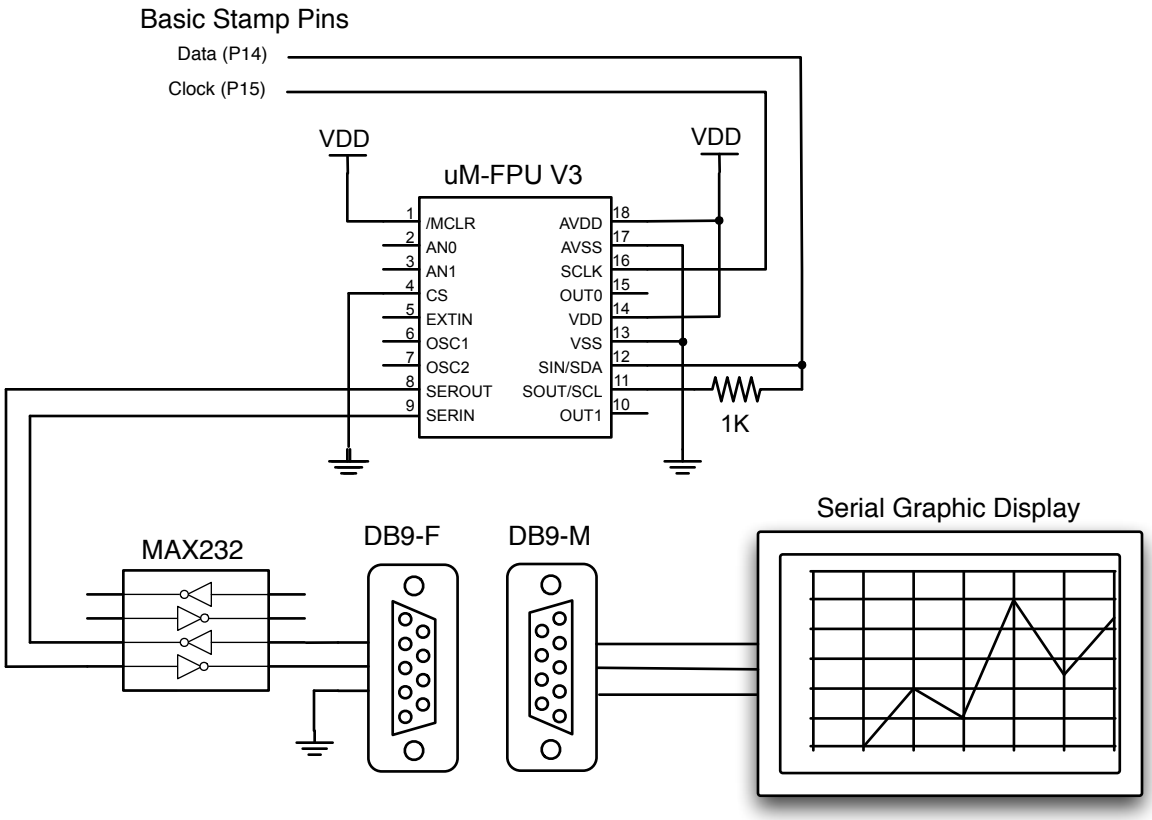


Figure 6
Line Graph with pi units on X axis.



Connection Diagram

The following diagram shows an example using a Basic Stamp microcontroller connected to the uM-FPU V3.1 chip using a 2-wire SPI connection. The serial graphic display is connected to the SERIN and SEROUT pins. The ezLCD-002 graphic display used in these examples uses a 115,200 baud serial connection. The SERIN and SEROUT pins are connected through a MAX-232 converter and DB-9 female. When the ezLCD-002 display is not being used, SERIN an SEROUT can be connected to a desktop computer for debugging using the uM-FPU V3 IDE software.



Sample Code - Demo1

Sample code for both BASIC Stamp and PICAXE microcontrollers are included with the support files for this application note. The following example shows the BASIC Stamp code for generating the graph in Figure 1.

Step 1 - Set the SEROUT pin and initialize the serial graphic display.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FCALL, initDisplay]
```

Step 2 - Initialize the graph.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FCALL, initGraph]
```

Step 3 - Store the data.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADUBYTE, 0, FCALL, storePoint]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADUBYTE, 10, FCALL, storePoint]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADUBYTE, 5, FCALL, storePoint]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADUBYTE, 25, FCALL, storePoint]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADUBYTE, 12, FCALL, storePoint]
```

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [LOADUBYTE, 22, FCALL, storePoint]
```

Note: This example uses simulated data. In a real application the data would be calculated, or read from some device connected to the microcontroller.

Step 4 - Draw the graph.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FCALL, drawLineGraph]
```

To add the grid lines shown in Figure 2, the formatX and formatY registers are simply changed before drawing the graph. e.g.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, $8F, formatX]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, $8F, formatY]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FCALL, drawLineGraph]
```

To draw the graph in Figure 4, the graphic window is set to a smaller size before drawing the graph, and the titles and labels are added after the graph is drawn. e.g.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, $AB, formatX]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, $EB, formatY]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, 30, gxWinMin]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, 210, gxWinMax]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, 20, gyWinMin]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [COPYI, 140, gyWinMax]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [FCALL, drawLineGraph]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [STRSET, "Line graph", 0, FCALL, drawTitle]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [STRSET, "X axis", 0, FCALL, drawXLabel]
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [STRSET, "Y axis", 0, FCALL, drawYLabel]
```

Sample Code - Demo2

The following example shows the PICAXE code for generating the histogram in Figure 2.

Step 1 - Set the SEROUT pin and initialize the serial graphic display.

```
writei2c 0, (FCALL, initDisplay)
```

Step 2 - Initialize the graph.

```
writei2c 0, (FCALL, initGraph)
```

Step 3 - Store the data.

```
value = 1659
gosub StoreWord
value = 1685
gosub StoreWord
value = 1817
gosub StoreWord
value = 2121
gosub StoreWord
value = 2963
gosub StoreWord
value = 6858
gosub StoreWord
value = 10491
gosub StoreWord
value = 2688
gosub StoreWord
value = 1593
gosub StoreWord
```

```
value = 1234
gosub StoreWord
value = 1011
gosub StoreWord
value = 924
gosub StoreWord
value = 851
gosub StoreWord
value = 840
gosub StoreWord
value = 814
gosub StoreWord
value = 808
gosub StoreWord
```

Note: This example uses simulated data. In a real application the data would be calculated, or read from some device connected to the microcontroller.

Step 4 - Draw the histogram.

```
writei2c 0, (FCALL, drawHistogram)
```

Sample Code - Demo3

The graphs shown in Figures 5 and 6 are drawn by the demo3 program. It calculates the points for a cosine functions and a multiplier function and displays the two graphs shown in Figure 5. The points are then multiplied together using the FPU, and the resulting graph is displayed as shown in Figure 6. Some of the registers that control the format of the X axis are used in this example.

The following instructions set *xUnits* to 3, which display the X scale in multiples of pi.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA, xUnits, LSETI, 3]
```

The following instructions set *xOffset* to -32. Since there are 65 data points collected, this puts the origin of the X axis in the center of the graph.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA, xOffset, LSETI, -32]
```

The following instructions set *xScale* to pi/8, to match the value used in the calculation.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA, xScale, LOADPI, FSET0, FDIVI, 8]
```

The following instructions set *xTick* to 8, so annotations are only shown every 8 data points.

```
SHIFTOUT FpuOut, FpuClk, MSBFIRST, [SELECTA, xTick, LSETI, 8]
```

graph-ezLCD2.fpu Functions

The *graph-ezLCD2.fpu* file contains uM-FPU V3.1 user-defined functions to do the following:

- initialize the serial graphic display
- initialize and set default values for the display window
- store data points in a floating point array
- draw a line graph
- draw a histogram
- calculate and draw various graph components
- format and manipulate strings
- delay for a number of milliseconds

The section below entitled *Summary of External Function Calls* provides a description of the functions that are intended to be called by the user program. The section entitled *Summary of Internal Function Calls* describes the support functions that are called internally by the other functions.

Summary of External Function Calls

getID

Returns an ID number that the main program can use to determine if the correct set of user-defined functions have been programmed on the uM-FPU V3.1 chip. If the correct `getID` function is programmed, a value of 42 is returned.

Input:

none

Output:

`register 0`	42 (long integer)

initDisplay

Sets the SEROUT pin for serial output at the required baud rate and initializes the serial graphic display. It returns the size of the display.

Input:

none

Output:

`register 1`	display width (pixels)
`register 2`	display height (pixels)

initGraph

Initializes the graphing routines to default values and clears the data array.

Input:

none

Output:

`gxWinMin`	minimum X value for graphic display
`gyWinMin`	minimum Y value for graphic display
`gxWinMax`	maximum X value for graphic display
`gyWinMax`	maximum Y value for graphic display
`fillColor`	WHITE
`axisColor`	BLACK
`dataColor`	BLUE
`formatX`	0x8B (draw X axis, ticks, scale, automatic X scale, mark data points)
`formatY`	0x8B (draw Y axis, ticks, scale, automatic Y scale, draw border)
`xUnits`	0 (Long Integer Scale)
`xOffset`	0

`xScale`	1
`xTick`	1
`yMin`	-1.0
`yMax`	1.0
`maxPoints`	65
`nPoints`	0

clearGraph

Clears the graph window.

Input:

`gxWinMin`	minimum X value for graphic display
`gyWinMin`	minimum Y value for graphic display
`gxWinMax`	maximum X value for graphic display
`gyWinMax`	maximum Y value for graphic display

Output:

`gxWinMin`	minimum X value for graphic display
`gyWinMin`	minimum Y value for graphic display
`gxWinMax`	maximum X value for graphic display
`gyWinMax`	maximum Y value for graphic display

clearPoints

Clears the data array. The data array is also cleared by the *initGraph* function.

Input:

none

Output:

`nPoints`	0

storePoint

Stores the next point in the data array.

Input:

`register 0`	data point (floating point)

Output:

`nPoints`	nPoints = nPoints + 1
`pointArray`	point stored in data array

drawLineGraph

Draw a line graph using the input values specified. Note: the `initGraph` routine provides default values for all of the input variables. The user only needs to modify the input variables if a different format is required.

Input:

`gxWinMin`	minimum X value for display window (long integer)
`gyWinMin`	minimum Y value for display window (long integer)
`gxWinMax`	maximum X value for display window (long integer)
`gyWinMax`	maximum Y value for display window (long integer)
`fillColor`	fill color for display window (long integer)
`axisColor`	color for axis, labels and scale values (long integer)
`dataColor`	color for data line and marker (long integer)
`formatX`	format of X axis (long integer)
`formatY`	format of Y axis (long integer)
`xUnits`	X axis scale units (long integer)
`xOffset`	X axis scale offset (long integer)
`xScale`	X axis scale multiplier (long integer or floating point)
`xTick`	X axis tick per scale annotation (long integer)
`yMin`	minimum Y value (floating point)
`yMax`	maximum Y value (floating point)
`maxPoints`	maximum points in data array (long integer)
`nPoints`	number of points in data array (long integer)

`pointArray` data array (floating point)

Output:

none

drawHistogram

Draw a histogram using the input values specified. Note: the `initGraph` routine provides default values for all of the input variables. The user only needs to modify the input variables if a different format is required.

Input:

`gxWinMin`	minimum X value for display window (long integer)
`gyWinMin`	minimum Y value for display window (long integer)
`gxWinMax`	maximum X value for display window (long integer)
`gyWinMax`	maximum Y value for display window (long integer)
`fillColor`	fill color for display window (long integer)
`axisColor`	color for axis, labels and scale values (long integer)
`dataColor`	color for data line and marker (long integer)
`formatX`	format of X axis (long integer)
`formatY`	format of Y axis (long integer)
`xUnits`	X axis scale units (long integer)
`xOffset`	X axis scale offset (long integer)
`xScale`	X axis scale multiplier (long integer or floating point)
`xTick`	X axis tick per scale annotation (long integer)
`yMin`	minimum Y value (floating point)
`yMax`	maximum Y value (floating point)
`maxPoints`	maximum points in data array (long integer)
`nPoints`	number of points in data array (long integer)
`pointArray`	data array (floating point)

Output:

none

clearWindow

Sets all pixels in the display window to the `fillColor`.

Input:

`gxWinMin`	minimum X value for display window (long integer)
`gyWinMin`	minimum Y value for display window (long integer)
`gxWinMax`	maximum X value for display window (long integer)
`gyWinMax`	maximum Y value for display window (long integer)
`fillColor`	fill color for display window (long integer)

Output:

none

drawTitle

Draws the title string. The string contained in the FPU string buffer is displayed at the top of the display window centered horizontally. If bit 6 of `formatY` is set, space will be reserved at the top of the display window for the title string.

Input:

FPU string buffer
internal registers

Output:

none

drawXLabel

Draws the X axis label string. The string contained in the FPU string buffer is displayed at the bottom of the display window centered horizontally. If bit 5 of `formatX` is set, space will be reserved at the bottom of the display window for the X axis label string.

Input:

FPU string buffer
internal registers

Output:

none

drawYLabel

Draws the Y axis label string. The string contained in the FPU string buffer is displayed at the left of the display window centered vertically and rotated 90 degrees. If bit 5 of `formatY` is set, space will be reserved at the left of the display window for the Y axis label string.

Input:

FPU string buffer
internal registers

Output:

none

drawLine

Draws a line from the first coordinate to the second coordinate.

Input:

`gx1, gy1`	first coordinate (long integer)
`gx2, gy2`	second coordinate (long integer)

Output:

none

drawBox

Draws a box from the first coordinate to the second coordinate.

Input:

`gx1, gy1`	first coordinate (long integer)
`gx2, gy2`	second coordinate (long integer)

Output:

none

fillBox

Draws a filled box from the first coordinate to the second coordinate.

Input:

`gx1, gy1`	first coordinate (long integer)
`gx2, gy2`	second coordinate (long integer)

Output:

none

drawString

Draws the string contained in the FPU string buffer. The top left of the string is located at the coordinate specified.

Input:

`gx1, gy1`	coordinate (long integer)

FPU string buffer

Output:

none

Summary of Internal Function Calls

The following functions are not normally called directly by the user program. They are used internally by the other functions.

drawBackground

Draws the background of the display window and calls the `graphSize` function to calculate the graph size and

position of the various elements of the graph. If the display window is too small for the graph specified, an X is drawn over the background.

Input:

`gxWinMin`	minimum X value for display window (long integer)
`gyWinMin`	minimum Y value for display window (long integer)
`gxWinMax`	maximum X value for display window (long integer)
`gyWinMax`	maximum Y value for display window (long integer)
`fillColor`	fill color for display window (long integer)
`axisColor`	color for axis, labels and scale values (long integer)
`dataColor`	color for data line and marker (long integer)
`formatX`	format of X axis (long integer)
`formatY`	format of Y axis (long integer)
`xUnits`	X axis scale units (long integer)
`xOffset`	X axis scale offset (long integer)
`xScale`	X axis scale multiplier (long integer or floating point)
`xTick`	X axis tick per scale annotation (long integer)
`yMin`	minimum Y value (floating point)
`yMax`	maximum Y value (floating point)
`maxPoints`	maximum points in data array (long integer)
`nPoints`	number of points in data array (long integer)
`pointArray`	data array (floating point)

Output:

`cnt`	number of points in graph (long integer) if display window is too small for the graph specified, zero is returned

drawXaxis

Draws the X axis, including ticks, grid lines and scale annotations..

Input:

internal registers

Output:

none

drawYaxis

Draws the Y axis, including ticks, grid lines and scale annotations..

Input:

internal registers

Output:

none

graphSize

This function calculates the graph size and position of the various elements of the graph. This function is not normally called directly by the user, since `drawBackground` calls this function internally.

Input:

internal registers

Output:

none

setXaxis

Calculates the X axis values.

Input:

`nx, gxMin1, gxMax1`

Output:

`gxMin, gxMax, gxTick, gxAxis`

setYaxis

Calculates the Y axis values.

Input:

```
ny, gyMin1, gyMax1
```

Output:

```
gyMin, gyMax, gyRange, gyAxis
```

setXvalue

Calculates initial X value.

Input:

```
nPoints, maxPoints
```

Output:

```
xval0, xval
```

getX

Gets the X graphic coordinate.

Input:

```
xval, gxTick
```

Output:

```
gx2
```

getY

Gets the Y graphic coordinate.

Input:

```
yval, yMin, yRange, gyMax, gyRange
```

Output:

```
gy2
```

getPoint

Gets the Y value for the current X index.

Input:

```
xval
```

Output:

```
yval
```

move

Sets the current position of the graph to the coordinate specified.

Input:

```
gx1, gy1
```

Output:

none

draw

Draws a line from the current position to the coordinate specified, and sets the current position to the new coordinate.

Input:

```
gx1, gy1
```

Output:

none

getXscale

Gets the X scale annotation string.

Input:

`xval`

Output:

`register 1,` FPU string buffer

getYscale

Gets the Y scale annotation string.

Input:

`yval`

Output:

`register 1,` FPU string buffer

longString

Get long integer string.

Input:

`register 1`

Output:

FPU string selection, `register 0`

timeString

Get time string.

Input:

`register 1`

Output:

FPU string selection, `register 0`

floatString

Get floating point string.

Input:

`register 1`

Output:

FPU string selection, `register 0`

freqString

Get frequency string.

Input:

`register 1`

Output:

FPU string selection, `register 0`

piString

Get pi string.

Input:

`register 1`

Output:

FPU string selection, `register 0`

trim

Trim leading and trailing spaces from the string buffer or string selection.

Input:

FPU string buffer or string selection

Output:

FPU string selection, `register 1`, `register 2`, `register 3`

trimNum

Trim leading and trailing spaces and trim trailing zeroes from the string buffer or string selection. If no digits remain to right of the decimal point, also trim the decimal point.

Input:
FPU string buffer or string selection
Output:
FPU string selection, `register 1`, `register 2`, `register 3`

stringWidth

Return the string width in graphic coordinate units.

Input:
FPU string buffer or string selection
Output:
`register 0`

delay_ms

Delay for the number of milliseconds specified.

Input:
`register 0`
Output:
none

Additional Files

There are additional files located on the Micromega website that accompany this application note. They include:

graph-ezLCD2.fpu	uM-FPU V3.1 user-defined functions for ezLCD-002 graphic display
graph-demo1.bs2	Basic Stamp demo programs
graph-demo2.bs2	
graph-demo3.bs2	
graph-demo1.bas	PICAXE demo programs
graph-demo2.bas	
graph-demo3.bas	

Before running the demo programs, the user-defined functions in *graph-ezLCD2.fpu* must be programmed into the uM-FPU V3.1 chip using the uM-FPU V3 IDE software.

Further Information

See the Micromega website (http://www.micromegacorp.com) for additional information regarding the uM-FPU V3.1 floating point coprocessor, including:

uM-FPU V3.1 Datasheet
uM-FPU V3.1 Instruction Set
Using the uM-FPU V3 Integrated Development Environment (IDE)

ezLCD-002 EarthLCD
http://store.earthlcd.com/ezLCD-002