ViewPort v4
December 2010

ViewPort Manual

by Hanno Sander

Table of Contents

Part I

Welcome

1.1 Welcome

ViewPort Development Studio

Welcome to ViewPort- real time graphing, debugging and more

What is ViewPort?
The ViewPort^{TM} Development Studio combines a graphical application with software for the popular microcontrollers to help you build advanced projects more quickly. Variables used in a your programs are shared over a high speed serial connection with ViewPort to allow you to control and monitor the program as it runs. This is known as SCADA (Supervisory Control And Data Acquisition). Data can be graphed, measured, or logged. Various controls allow you to change variables in your program as it's running. ViewPort can be integrated into any program - in Propeller spin it requires one cog and a single line of code at the start of your program. It's easy to get started with plenty of tutorials, videos and documentation. It is also configurable and extensible so you can customize it to your needs.

How can ViewPort help me?
- debug code with breakpoints or step line by line
- view and graph variable values over time to track your program's behavior
- view and graph microcontroller pin status to simplify interfacing with other hardware
- pc-based controller- control an embedded system from your desktop
- tuning and calibrating constants, for example tuning a PID controller
- take measurements with the DSO, LSA, Spectrum Analyzer, Data Logger
- SCADA prototype tool to integrate PC with real time data acquisition and control
- teaching tool for interfacing with hardware like sensors, motors...
- robot design platform to monitor internal robot state as you control it
- fuzzy logic engine and graphical editor
- video capture and vision processing with OpenCV

Features:
- High Speed connection to microcontroller memory
- Change and Monitor variables in your program as it runs
- Analyze data with Oscilloscope, Logic State Analyzer, Spectrum Analyzer modes
- View the state of the Input/Output port at up to 80MHz
- Create Custom Visualizations. Build your own controls with the Development Kit
- Integrate Fuzzy Logic into your program, with Graphical Tuning
- Capture Video, apply Vision Filters, View Video and Apply OpenCV filters
- Comprehensive Help Manual, Easy Install/Uninstall
- Comprehensive Terminal object support existing print style debugging.
- Get started Integrating with ViewPort with Documented, Full Source Tutorials
- Plugins include: Integrated Spin Code Debugger, OpenCV Engine

Installation and Integration:
ViewPort is easy to install and the tutorials help you to get started quickly. Integrating ViewPort into your own programs takes one line of code at the start of your program.

Links:

Home Page:	http://hannoware.com
Blog:	http://blog.hannoware.com
Forums:	http://forums.hannoware.com
Parallax PE Lab Kit for ViewPort:	http://www.parallax.com/Portals/0/Downloads/docs/prod/prop/PE-Lab-ViewPortApps.zip

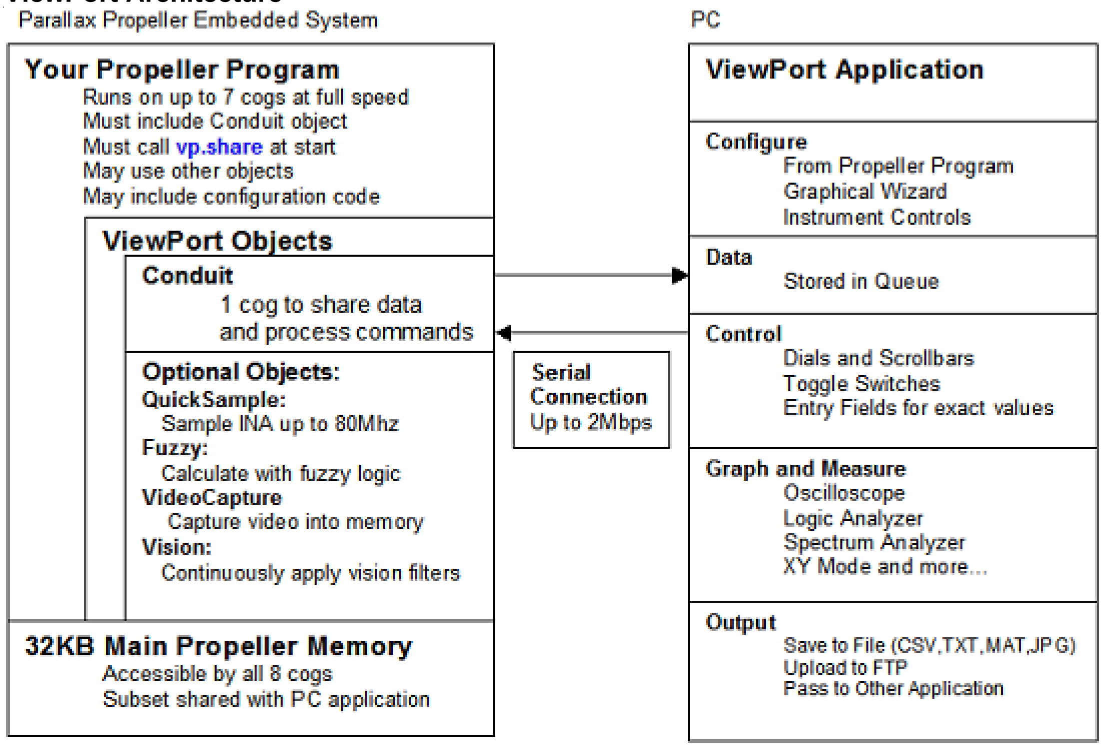
Part II

Introduction

2.1 Overview

Background
ViewPort was first developed for the Parallax Propeller, a powerful microprocessor used in embedded systems, robotics and advanced electronic projects. ViewPort uses one of the Propeller microcontroller eight processors (cogs) to share data with a graphical application that runs on a PC. This allows users to monitor and control the program running on the Propeller. ViewPort now supports other microcontroller's as well

ViewPort Architecture



Architecture Walkthrough
The user's Program runs on the Propeller and has configured some data for sharing with ViewPort. The Program may also register other objects, like the QuickSample object to take quick readings of the IO port.

The Conduit object runs in a separate cog from the user's but has access to the shared variables which are stored in Propeller main memory. It sends the shared data to the Windows host over the serial connection. The conduit can also receive commands from the host to change variables or control ViewPort objects.

Data received by the host is stored in a queue. When the queue is full the oldest measurement is thrown out.

The ViewPort application includes multiple views to display data. Use these views to measure and analyze the data using standard instruments. Data can be logged to different file formats, uploaded to an ftp server, or passed to an external program. Views are defined by XML files which define the layout and base configuration for ViewPort controls. Additional controls can be programmed using the Developer's Kit.

Configuration information such as what variables to graph, time scale and bit labels can be manipulated with the configuration wizard, saved and loaded from configuration files, set using controls presented by the virtual instruments, or included in the user's program

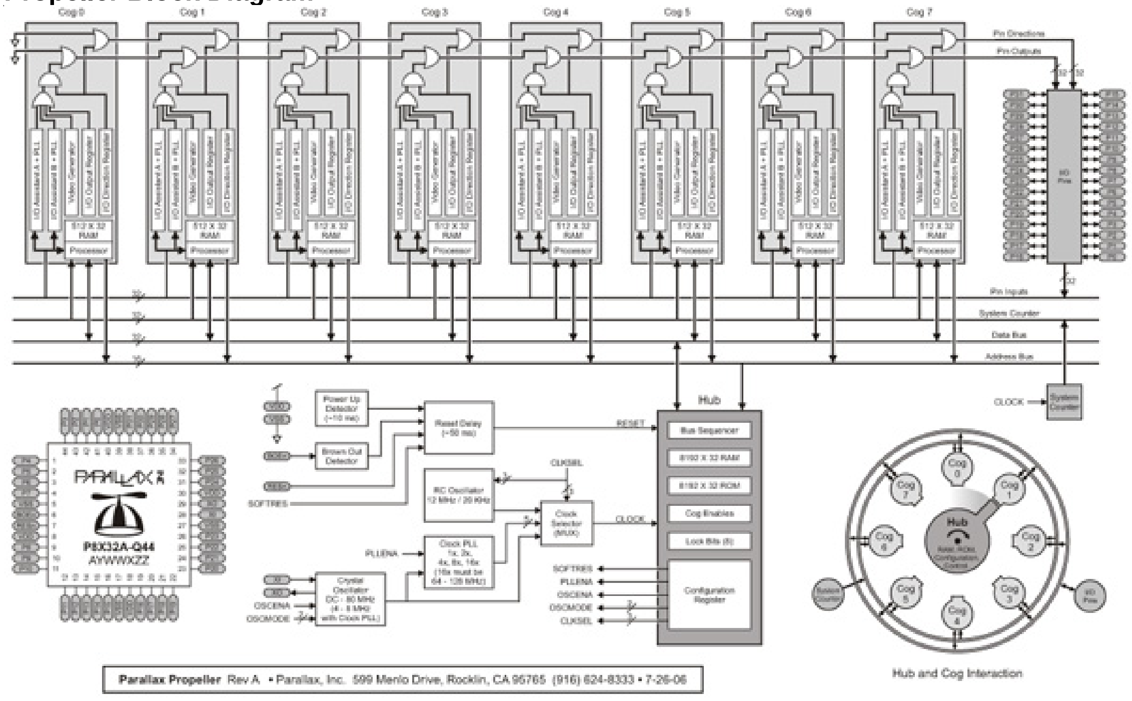
2.2 Parallax Propeller

Parallax Propeller: 32 Bit, 80MHz, 8 cog Microprocessor
The Propeller chip makes it easy to rapidly develop embedded applications. Its 8 processors (cogs) can operate simultaneously, either independently or cooperatively, sharing common resources through a central hub. The developer has full control over how and when each cog is employed; there is no compiler-driven or operating system-driven splitting of tasks among multiple cogs. A shared system clock keeps each cog on the same time reference, allowing for true deterministic timing and synchronization. Two programming languages are available: the easy-to-learn high-level Spin, and Propeller Assembly which can execute at up to 160 MIPS (20 MIPS per cog).

Propeller Specification:
Processors: 8 32 bit processors running at up to 80MHz
Shared Global Resources: 32KB RAM, 32KB ROM, 32 IO Pins
Dedicated Resources Per Processor: 2KB RAM, 2 General Counters, Video Output
Power: 3.3volt DC, each pin can sink up to 40mA

Propeller Development Environment:
The Propeller Tool is a free Editor/Compiler/Loader. Compiled programs are loaded to the Propeller over a serial connection.
A large community of users frequent the Parallax Forums: http://forums.parallax.com
The repository of source code object is known as the Object Exchange: http://obex.parallax.com
Information about the Parallax Propeller can be found here: http://www.parallax.com/propeller

Propeller Block Diagram



Courtesy of Parallax Inc.

2.3 Uses

Use Case	Detail
Debugging Code	Share variables with ViewPort and monitor how they change over time in the DSO mode. Step through spin code, set breakpoints, watch call stack and profiler.
PC-based Controller	Connect Propeller to a device and share the variable controlling it's behavior. For example, use a ViewPort dial to control a hobby servo's position.
Tuning and Calibrating Parameters	Use a scroll bar in ViewPort to change the parameters of a PID Control
Interface with other Hardware	Use the LSA view to monitor timing signals when working with I2C devices like a compass or eeprom
Take Measurements	View and measure the signals from an ADC with the DSO, Spectrum Analyzer, Data Logger and more.
SCADA Prototype Tool	Control a remote system and log data for sharing with other programs.
Teaching Tool	Introduce operation of basic instrumentation such as DSO, LSA etc.
Robot Design Platform	View sensor values and control actuators with ViewPort
Fuzzy Logic Platform	Use the control panel to adjust fuzzy logic rules to provide fuzzy logic control for your program.
Vision Platform	Watch raw and vision-processed video from a camera with ViewPort. Integrate with OpenCV library

Projects Ideas*	Detail
Internet Weather station	1. Add sensors to the Propeller: temperature, wind direction and speed, camera... 2. Create a weather station view 3. Automatically upload screenshots to ftp server
Balancing Robot	1. Build robot with tilt sensor and wheel encoder 2. Tune the control algorithm
Digital Oscilloscope/Spectrum Analyzer	1. Add an ADC to the Propeller to measure analog values 2. View signal in DSO/Analog views
Function Generator	1. Add a DAC to the Propeller to generate analog values 2. Assign controls to control the waveform: pwm, sine, square...
Measure sensors	Gyro, accelerometer, Ping, IR , wheel encoder, compass, GPS, temperature …….
Control actuators	Solenoid, Hobby Servo, motors, stepper motor, H-bridge
Study Control Theory	Bode plot, step response, PID, fuzzy logic, state space, Kalman-filter, Signal process DSP, filter, sample, FFT…
Image Processing	TV, camera, NTSC, Image process, Pixel manipulation, blob finder, contour…
Process Control	Measure temperature and pH while fermenting beer
Pick and Place	Tune software to use vision to recognize objects then move them

*For additional ideas visit our site: http://hannoware.com/viewport/applications.php to see how people use ViewPort.

2.4 About HannoWare

The people behind HannoWare believe in building sophisticated, yet affordable products by pushing off the shelf components to their limits. We aim to understand the components we use through detailed measurements and analysis- then we engineer the optimal solution.

Thus far, we have developed:

- ViewPort-The Premier Debugging Environment for the Propeller
- 12Blocks-Drag blocks together to quickly build programs
- PropScope-USB Oscilloscope, Function Generator, Logic Analyzer
- TBot-Sophisticated wireless robot
- DanceBot-Vision Guided Balancing Robot
- Parallax Propeller Book- Programming & Customizing the Multicore Propeller Microcontroller: Official Guide

Links:

Home Page:	http://hannoware.com
Blog:	http://blog.hannoware.com
Forums:	http://forums.hannoware.com
Parallax Propeller:	http://parallax.com/propeller

Part III

Getting Started

3.1 Installation

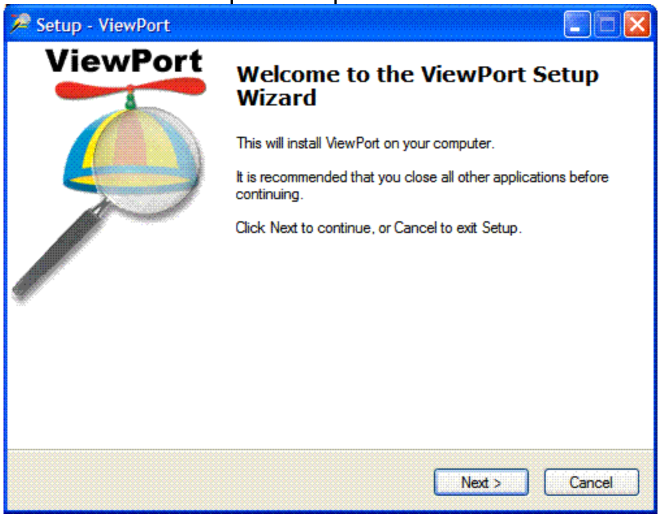
Requirements:
PC host system:
- >500MHz CPU with 5MB HDD Space, 500MB RAM, Mouse
- Windows 95,98,2k,XP,Vista, Win7
- USB 2.0 connection with standard USB cable (preferred) or serial to Propeller
- Parallax Propeller Tool software installed with USB-serial driver installed

Parallax Propeller target
- Parallax propeller chip with power supply
- 5MHz crystal
- EEPROM, LEDs, video input are optional
- The Parallax ProtoBoard is a low-cost, high-quality solution for Propeller projects

BasicStamp2 target
- Parallax BS2 microcontroller, like the BS2-IC module

Users should be familiar with their microcontroller. Familiarity with technical instruments like digital storage oscilloscope (DSO) and logic state analyzer (LSA) is a bonus but not required.

Installation:
Download ViewPort and follow the installation wizard to install it. If you're upgrading, the installer will replace old ViewPort files with the updated versions. Before starting ViewPort, make sure a microcontroller is connected to your computer- ideally with a USB connection. USB allows for faster connection up to 2Mbps.



Welcome to the ViewPort Setup Wizard

This will install ViewPort on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Uninstall:
To uninstall, select the uninstall item from the Windows Start menu.
Video Tutorials:
The video tutorials show ViewPort in action- view them here:
http://hannoware.com/viewport/videos.php
Problems:
If you encounter problems, check the Problem Solving guide or contact us: http://hannoware.com

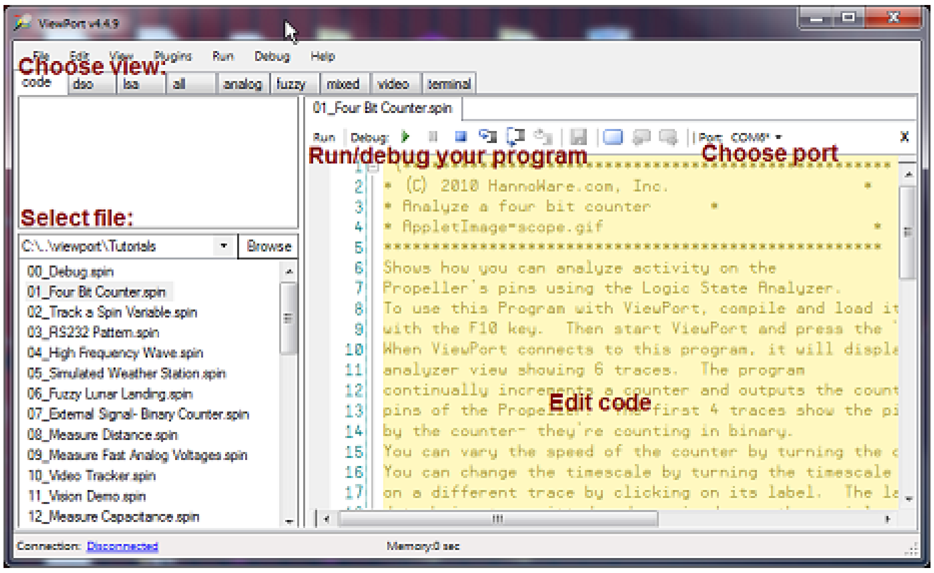
3.2 Load a Tutorial

Pre-requisites:

- ViewPort should be installed
- Propeller turned on and connected to this PC

Load Tutorial #1

ViewPort starts with the "code" view. You'll use this view to manage files and edit your code. The file browser remembers which directories you've used- it starts out with the tutorial folder which contains programs that were designed to demonstrate the capabilities of ViewPort- we suggest you work through them in order. An overview of all tutorials and source code for the first one are in the Reference section.

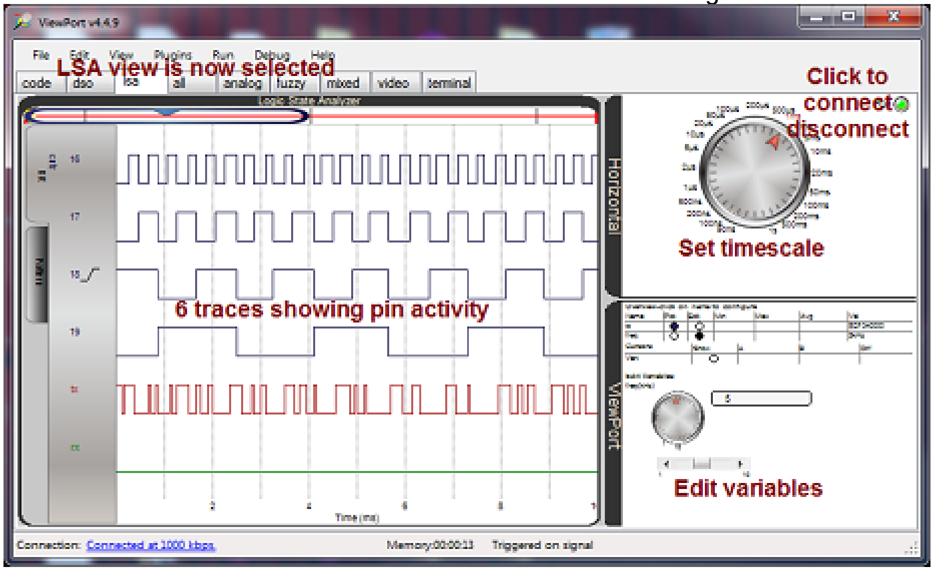


```
* (C) 2010 HannoWare.com, Inc.                            *
* Analyze a four bit counter        *
* AppletImage=scope.gif                                    *
**********************************************************
Shows how you can analyze activity on the
Propeller's pins using the Logic State Analyzer.
To use this Program with ViewPort, compile and load it
with the F10 key.  Then start ViewPort and press the
When ViewPort connects to this program, it will displa
analyzer view showing 6 traces.  The program
continually increments a counter and outputs the count
pins of the Propeller.  The first 4 traces show the pi
by the counter- they're counting in binary.
You can vary the speed of the counter by turning the c
You can change the timescale by turning the timescale
on a different trace by clicking on its label.  The la
```

Start by selecting the tutorial you wish to work with- in our case, "01_Four Bit Counter.spin", a program that exercises the Parallax Propeller's IO pins. This first tutorial demonstrates how you can use ViewPort to analyze activity on the Propeller's pins using the Logic State Analyzer. You can read and change the file in the large edit area. Use the "Port" chooser to specify which port to use- or leave it in "Auto" mode. Make sure a Propeller is connected to your PC then load the tutorial to the Propeller by clicking "Run".

3.3 Interact with a Tutorial

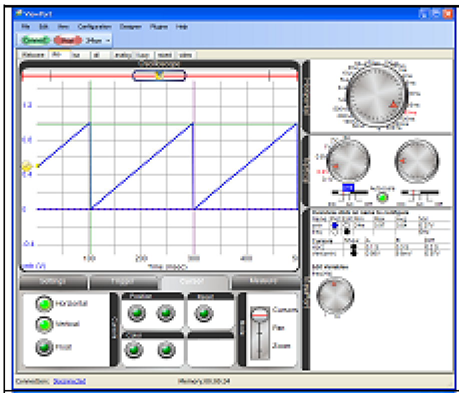
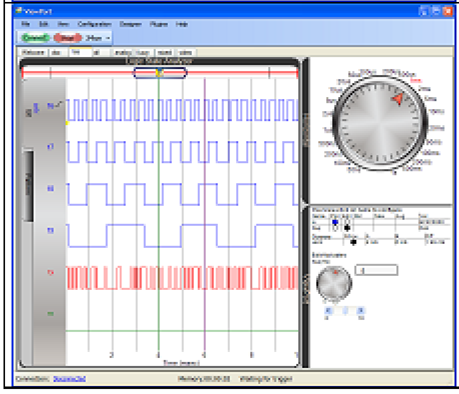
ViewPort will load the tutorial's spin code to the Propeller and establish a connection with the program running on the Propeller. The tutorials' spin code includes an optional configuration section that makes ViewPort switch to the LSA view showing 6 traces


ViewPort v4.4.9
File Edit View Plugins Run Debug Help
LSA view is now selected
code dso lsa all analog fuzzy mixed video terminal
Logic State Analyzer
Click to connect/disconnect
Set timescale
6 traces showing pin activity
Horizontal
ViewPort
Edit variables
Time (ms)
Connection: Connected at 1000 kbps.
Memory:00:00:13
Triggered on signal

The program continually increments a variable counter and outputs the counter value to the pins of the Propeller. The first 4 traces show the pins being toggled by the counter- they're counting in binary. You can adjust the speed of the counter by turning the dial in the edit section. You can change the timescale by turning the timescale dial and you can trigger on a different trace by clicking on its label. The last 2 traces represent the data being transmitted and received over the serial connection. Click the top right "Connect" button to stop/run the connection.
Click on the "code" view to edit/run other tutorials.

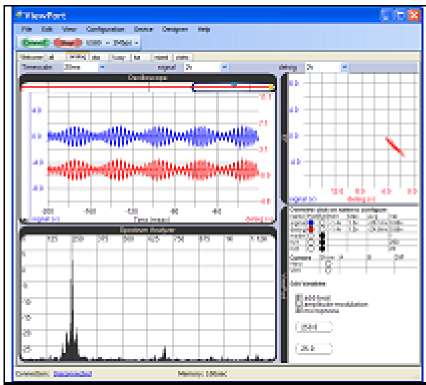
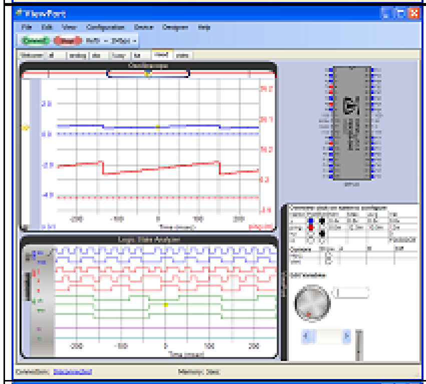
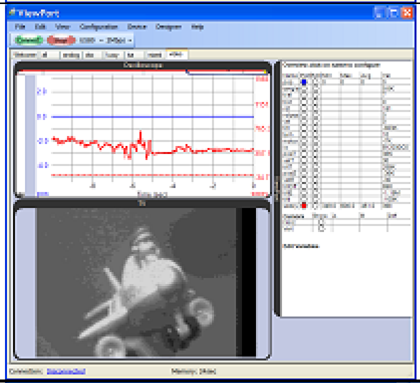
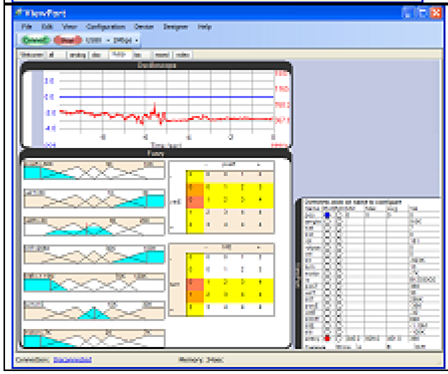
3.4 Views

ViewPort uses Views to Organize Your Screen
You've already seen two views, "code" to view/edit code and "lsa" to graph the Propeller's I/O pin states. When you loaded the "Bit Counter" tutorial, ViewPort switched to the "lsa" view because the configuration of the program specified to use that view. The configuration also specified how to graph the data- what labels to use, what to trigger on, and even the timescale. Specifying the configuration in your program gets you going quickly. Later on you'll learn how to use the graphical wizard to create configuration codes for your own programs- for now, explore the other views and then return to the "code" view. Here's a brief summary of the most commonly used views.

	**Digital Storage Oscilloscope (DSO)** graphs data on an oscilloscope and provides controls for measuring and triggering on the data. Use to monitor how variables change over time. Shown in graph is the result of tutorial #2: Track a Spin Variable. The blue trace's frequency can be changed with the dial.
	**Logic State Analyzer (LSA)** graphs the individual bits making up a variable. Use to view the state of the IO port and communication with other devices Shown in the graph is the result of tutorial #1: Four Bit Counter. The traces show the state of the Propeller's IO pins- which are toggled with an incrementing variable.

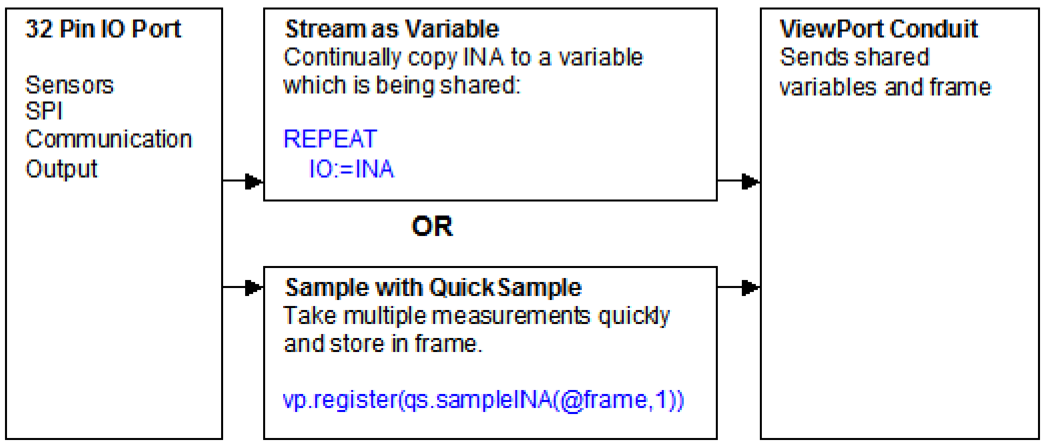
3.5 Advanced Views

These advanced views combine multiple graph elements in one view. For example, the "Mixed" view displays the DSO graph and the LSA graph together. Typically, you'll start by setting the configuration (time scale, triggers, resolution) in the pure views and then switch to an advanced view to analyze your data all in one view. All changes you make in a view will carry across to other views.

	**ANALOG** Graphs data on an oscilloscope, in an xy graph, and in a spectrum analyzer. Use to examine analog waveforms. Shown in graph is the result of Tutorial #13: Amplitude Modulation. The blue signal is composed of a carrier signal onto which another frequency is modulated. The Spectrum Analysis shows the carrier signal with sidebands.
	**MIXED** Graphs data on an oscilloscope and shows the status of the IO port graphically and in a logic state analyzer. Use to analyze both digital and analog waveforms at the same screen. Shown is a sawtooth pattern in the DSO and a binary counter in the LSA.
	**VIDEO** shows video being streamed from the Propeller- typically the output of vision processing applied to digitized video. Use to tune vision algorithms, or just to watch the output of a camera Shown is tutorial #11: Vision Demo showing the image of a toy airplane.
	**FUZZY LOGIC:** provides a control panel to tune the parameters of a program's fuzzy logic engine. Use to tune a fuzzy logic engine. Shown is tutorial # 6: Fuzzy Lunar Lander showing the controls required to land on the moon with fuzzy logic.

3.6 View IO Port Status

Some tutorials share a variable called "IO". This variable reflects the state of the microcontroller's IO pins. Tracking this variable helps you understand which pins are "high" or "low". To communicate with other devices, pins are set "high" or "low" at specific times- being able to monitor the IO port helps you troubleshoot communication problem. Typically the LSA graph is used to view the individual bits that make up the "IO" variable- each bit corresponds to an IO pin. Your configuration can label and group the bits to simplify the graph.


32 Pin IO Port
Sensors
SPI
Communication
Output
Stream as Variable
Continually copy INA to a variable which is being shared:
OR
Sample with QuickSample
Take multiple measurements quickly and store in frame.
ViewPort Conduit
Sends shared variables and frame

```
REPEAT
   IO:=INA
```

```
vp.register(qs.sampleINA(@frame,1))
```

Streaming As Variable
When the Conduit sends data from the Propeller to the PC it streams one variable after another. At 2Mbps, the Conduit can send 50K longs to the PC every second. This allows ViewPort to store 50,000 32bit measurements per second- good for graphing a single variable that changes a up to 25KHz (Shannon's law) With more variables, this maximum rate decreases. When you "share variables" with ViewPort, they are streamed- this allows you to look at a variable's value anywhere in time with good time resolution.

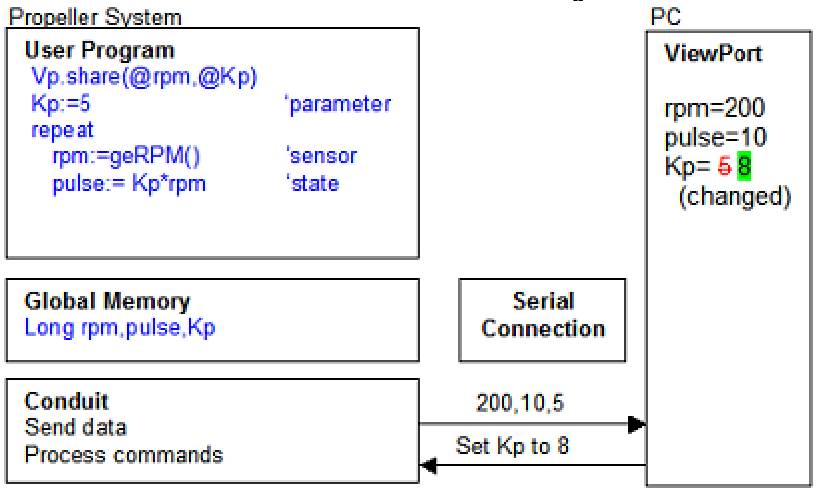
Sample with QuickSample
To support faster IO transitions, ViewPort can use a technique called "sampling". Sampling allows you to look at IO transitions that happen very quickly- up to 80MHz. Sampling works by taking multiple samples in rapid succession, storing them in Propeller memory "frame", and then transmitting all measurements. To use sampling in your program, just register the QuickSample object. Sampling provides a high sample rate for the IO port, but it will decrease the sampling speed of other variables in your program.

3.7 View and Change Variables

ViewPort displays the current value of variables in your live program in the ViewPort section of the view. This section also lets you control if a variable should graphed in the active graph or edited with a control. You can click on a variable's name to change the configuration of that variable.

ViewPort constantly transfer data from the microcontroller's memory to the PC. This allows ViewPort to graph and show you the value of "shared" variables. When you "edit" a variable, ViewPort sends a command to the Propeller which sets the variable to the new value.

The following diagram shows how ViewPort is used to view and change variables in a motor control program. The user's program shares global memory from rpm to kp. The conduit sends the variable's values to ViewPort for display. When the user changes the Kp variable from 5 to 8, that command is send to the conduit which changes the variables value in global memory.



```
Vp.share(@rpm,@Kp)
Kp:=5                    'parameter
repeat
   rpm:=geRPM()          'sensor
   pulse:= Kp*rpm        'state
```

3.8 Connect/Disconnect

The Connect button let you start or stop the connection with the Propeller. You can still view, manipulate, and save data, but no new data will be captured while the connection is stopped.

Every time you start a connection with the Propeller, ViewPort will query the program's configuration. If the program's configuration has changed (because you loaded a new program), then ViewPort will use that new configuration to it receives from the Propeller to configure itself. Configuration can also be saved and loaded as files or included in a program.

Auto-Connect/Auto-Disconnect
Only one Program on your PC can use the COM/USB port that connects the PC to the Propeller chip. Typically you'll want to use the Propeller Tool to develop new programs and use ViewPort to debug them. Auto-Connect/Disconnect makes this easy by stopping the connection when the ViewPort Window is no longer active (like when you select the Propeller Tool window). The connection will restart when the window becomes active again (when you click on the ViewPort window). You can disable this behavior under "Edit/Preferences".

Part IV

Integrating your Program with ViewPort

4.1 Overview

Integrating a Propeller Spin program with ViewPort takes 3 Steps:

1) Open your program with ViewPort inside the "code" view. Your Program should:
   - compile and run without problems,
   - use the standard clock setting to run the Propeller at full speed- ie 80MHz
   - have at least 1 cog available for ViewPort,
2) Integrate using ViewPort Objects
   - Start by including the ViewPort conduit object in the OBJ section of your main program like this:

     ```
     OBJ
         vp : "Conduit"
     ```

   - Next, declare the variable for your program.

     ```
     VAR
         long a,b,c,d,e
     ```

   - Finally, at the beginning of your main program call the conduit object's **share** method and pass it the address of the first and last variable in memory you want to share with ViewPort.

     ```
     PUB MAIN
         vp.share(@b,@d)
     ```

     In this case, vp.share(@b,@d) shares 3 variables, b, c and d with ViewPort. Each packet will contain 3 longs- equal to 12 bytes. Notice that you can only share global "long" variables.

   - Optionally, add other ViewPort objects or configuration code BEFORE the vp.share command. Click on an object for additional information:
     - o QuickSample: uses 1 to 4 cogs to sample the IO port at any speed up to 80Msps.
     - o Fuzzy: provides spin routines that add a fuzzy logic engine to your program to simplify control problems.
     - o PropCVCapture: uses 1 cog to capture video into memory for processing or streaming to ViewPort.
     - o PropCVFilter: uses 1 cog to continuously apply vision filters to captured video.
     - o Terminal: use instead of Conduit when you need a serial terminal.

   - Save your program before loading it to the Propeller.

3) Load your program to Propeller RAM (F4 key), or EEPROM (F3 key)

   ViewPort will compile your program and link it with objects stored in either the ViewPort library "/viewport/lib" or the Propeller library. It will then find and load your Propeller, establish a connection with the conduit, switch to the specified view, and display data for the variables being shared. At this point you can interact with the view to "plot" variable values, change timescale, etc. You can also use the Configuration Wizard (F2 key) to edit the configuration data.

4.2 Integrate ViewPort into spin code

This is a an example where we start with a simple counting program and walk through the steps discussed on the previous page.

Here is a simple Spin program that counts from 0 to b using variable a:

```
CON
 _clkmode        = xtal1 + pll16x
 _xinfreq        = 5_000_000
OBJ
VAR
      long a,b
PUB count
      b:=1000
      repeat
            a++
            if a>b
                  a~
```

The program uses the standard 80MHz clock, includes no objects, and stores a and b in global memory. The program can easily be modified so that you can use ViewPort to monitor and adjust variable values.

Integrate ViewPort into the Program

Add the lines:

```
      vp:"Conduit"
```

and

```
      vp.share(@a,@b)
```

so your program reads:

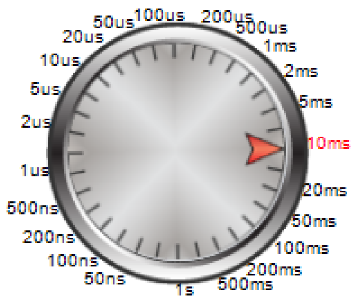
```
CON
 _clkmode        = xtal1 + pll16x
 _xinfreq        = 5_000_000
OBJ
      vp:"Conduit"
VAR
      long a,b
PUB count
      vp.share(@a,@b)
      b:=1000
      repeat
            a++
            if a>b
                  a~
```

Save the file then load to Propeller RAM using the F4 key.
ViewPort should connect to your Program and graph 2 variables in the DSO mode. Learn how to configure ViewPort in the next step

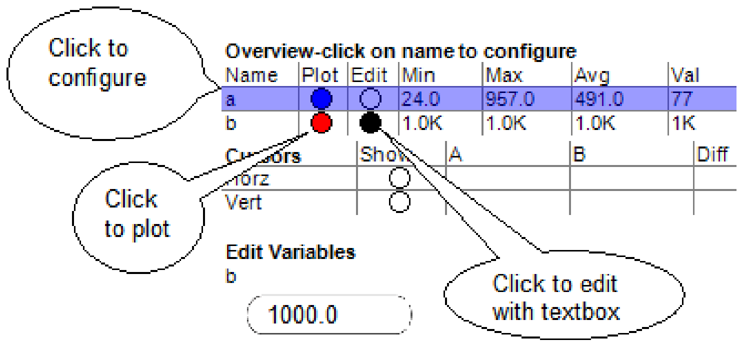
4.3 Configure ViewPort Settings

In the previous step we've edited and loaded a simple program to the Propeller. Since ViewPort was not provided with any configuration, it starting in the default view (DSO), has assigned default names to the variables (v1 and v2), and has not started graphing any data. The following pictures illustrate the result of configuring ViewPort:

Let's first change the time scale of the graph- by turning the time scale dial from 1 sec/div to 10ms/div.



Next, find the "ViewPort" section in the view.
Click the "Name" column for each variable to open the configuration wizard. For now, change just the name of the variables to "a" and "b".
Next, click on the radio button in the "edit" column for variable "b" to bring up a textbox control. Finally, click the "plot" button to graph the data.



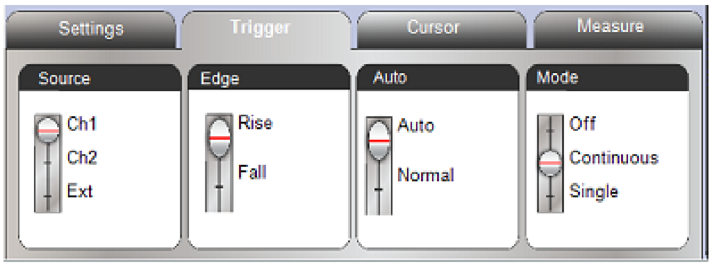
Overview-click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
a	●	○	24.0	957.0	491.0	77
b	●	●	1.0K	1.0K	1.0K	1K

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

Edit Variables
b
1000.0

For the last step, set a "Trigger" to stabilize the graphical display of the signal. Click on the "Trigger" tab and set the "Mode" to "Continuous". Now, the graph should be stable because the trigger will continuously look for a rising edge on ch1- with an automatic level.



Source: Ch1, Ch2, Ext
Edge: Rise, Fall
Auto: Auto, Normal
Mode: Off, Continuous, Single

ViewPort's configuration is global across all views. This means you can configure a time scale or name in one view and use it in another.
Configuration data can be saved and loaded from configuration files using the Configuration menu. This allows you to quickly set up your instruments.
You can use the "Copy to Clipboard" item in the "Configuration" menu to copy and paste the configuration before the **vp.share** command in your program. This allows your Program to set up ViewPort. Here is the configuration string for this program:

```
vp.config(string("var:a,b"))
vp.config(string("start:dso"))
vp.config(string("dso:view=[a,b],trigger=a>auto,timescale=10ms"))
vp.config(string("edit:b(mode=text)"))
vp.config(string("lsa:timescale=10ms"))
```

The commands should make some sense when we read them. For now, we can use ViewPort's graphical configuration wizard and instrument dials to create our configuration and then copy/paste it into our program. With time, we'll learn how to copy/paste configuration from other programs, or type the configuration into our program directly.

4.4 Source Code for Tutorial 1

```
 {*************************************************
* (C) 2010 HannoWare.com, Inc.                    *
* Analyze a four bit counter      *
* AppletImage=scope.gif                            *
*************************************************
Shows how you can analyze activity on the
Propeller's pins using the Logic State Analyzer.
To use this Program with ViewPort, compile and load it into your Propeller's
RAM
with the F10 key.  Then start ViewPort and press the "Connect" button.
When ViewPort connects to this program, it will display a logic
analyzer view showing 6 traces.  The program
continually increments a counter and outputs the counter value to the
pins of the Propeller.  The first 4 traces show the pins being toggled
by the counter- they're counting in binary.
You can vary the speed of the counter by turning the dial in the edit section.
You can change the timescale by turning the timescale dial and you can trigger
on a different trace by clicking on its label.  The last 2 traces represent
the
data being transmitted and received over the serial connection.}
CON
 _clkmode     = xtal1 + pll16x
 _xinfreq        = 5_000_000         'use standard 5MHz clock
OBJ                                  'include 2 ViewPort objects:
 vp    : "Conduit"                   'transfers data to/from PC
 qs    : "QuickSample"      'samples INA continuously in 1 cog- up to 20Msps
VAR
  long cntr            '             'binary counter
  long nextCycle                     'time for next increment
  long freq                          'counter frequency- edited with ViewPort
dial
  long frame[400]                    'stores measurements of INA port
pub demo
 vp.register(qs.sampleINA(@frame,1)) 'sample INA into <frame> array
 optional_configure_viewport     'optionally configure viewport's interface
 vp.share(@freq,@freq)               'share the <freq> variable
 dira[16..19]~~                      'output pulses on pins 16..19
 nextCycle:=cnt+clkfreq              'initialize first cycle time
 repeat
  outa[19..16]:=(cntr+=1)            'count in binary on pins 19..16
  nextCycle+=((clkfreq/freq)#> 6000 <#clkfreq/10)
  waitcnt(nextCycle)                 'wait until next cycle- controlled by
<freq> variable
pub optional_configure_viewport
vp.config(string("var:io(bits=[cntr[16..19],30tx,31rx]),freq(unit=kHz,f=x/1000
,min=1,max=10)"))
   'the io frame is filled by the sampleINA, we're only interested in specific
   'bits which we name: 16..19 make up the cntr, 30 is tx and 31 is rx
   'share the freq variable.  it has a unit of Hz with range from 1 to 10
 vp.config(string("lsa:view=io,timescale=1ms,trigger=io[18]r"))
  'configure the lsa graph to display the io variable with 1ms timescale
  'and trigger set on bit 18 rising
 vp.config(string("edit:freq(default=5,mode=[dial,text,scroll])"))
  'configure the edit section to display a dial and text for the freq variable
 vp.config(string("start:lsa"))
     'start in lsa mode
```

Part V

Reference (Windows Application)

5.1 Menus

File

- **New:** creates a new file
- **Open:** opens a file
- **Save/Save As:** saves a file
- **Configuration:** opens the configuration wizard
- **Export Visible Data:** saves data being viewed in the active graph and ViewPort configuration to a csv, txt, jpg, png or matlab file.
- **Export All Data:** saves all data and ViewPort configuration to a csv, txt, or matlab files.
- **Close:** closes a spin file in "code" view or a "view".
- **Stream:**periodically saves data to disk in a specified format, executes a specified program, and/or uploads the file to an ftp server.
- **Restart:** clears data
- **Exit:** closes program.

Edit

- **Undo, Cut, Copy, Paste** performs operation on text
- **Program Preferences**
  - **Disconnect while Window is Inactive:** If selected, will disconnect when the window is not active. This happens when you Alt-Tab to another application, minimize the window, or click on another application. When the window is activated again, ViewPort will reconnect.
  - **Baud Rate**:Set rate at which ViewPort communicates with your program. ViewPort's default baud rate is 1MBaud- equivalent to 1 million bits/second or roughly 100,000 bytes/second. This allows ViewPort to quickly transfer data back to support high sampling rates as well as streaming video. Under ideal conditions, you can increase the baud rate to 2MBaud- but we recommend you stay with the default. ViewPort lite is restricted to a slower rate of 115kbps.
  - **Display Hints:** If checked, ViewPort will display tooltips when you hover over a control.
  - **Buffer Size:** ViewPort reads data from the microcontroller and stores it for analysis in a rolling buffer. You can set that size here, smaller to use less memory, larger to allow more data to be analyzed over longer periods.
  - **Auto Scale Interval:** Change the interval at which ViewPort autoscales your data here
  - **Video Settings:** Tune the brightness, contrast and gamma here for streaming video.
  - **File Granularity:** When saving "All Data", this setting is used to skip samples to reduce file size.
  - **C compiler:** Location and information to help ViewPort compile C files
  - **Folder Manager:** ViewPort remembers which folders you've used for storing programs. Manage them here.
- **Copy Screenshot:** copies a screenshot of the view to the windows clipboard, you can paste it into MS Word and some Email programs to share your work.
- **mark like modifications, outlining, bookmarks, macros, find/replace, goto:** use these, and more to edit your code

View

- **Full Screen:** switches to full screen mode- taking over the entire monitor. Great for projection for a class. Press "Esc" to return.
- **Designer Mode:** Toggle edit mode on and off

5.2 Menus Part 2

Plugins

- **Add:** visits the http://hannoware.com/viewport/plugins website to download and install additional capabilities.
- **Manage:** get information about particular plugins or uninstall them
- **List of Installed Plugins:** shows information for installed plugins

Run

- **Identify/Reset Hardware:** searches the active port for your hardware and resets it
- **Compile:** compiles your program
- **Load RAM:** compiles program, loads to your hardware to RAM and starts the connection
- **Load EEPROM:** compiles program, loads to your hardware's EEPROM and starts the connection
- **Connect:** connect to your hardware to monitor/change variables
- **Disconnect:** disconnects from your hardware to release the communication port and freeze the data

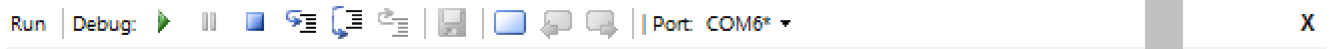
Debug

- **Start Debugging:** Loads the debug enabled program and starts the debugging session
- **Pause:** Pauses the active cog and shows the current line
- **Stop:** Stops the debugger and connection
- **Step Into:** Steps one instruction further
- **Step Over:** Step one instruction/function call further
- **Step Out:** Step out of current function

Help

- **Help Index:** opens the search and browseable Help file.
- **PDF Manual**: links to the pdf manual available at: http://mydancebot.com/viewport/manual.pdf, suitable for printing
- **Check for Updates:** Visits the myDanceBot website to check for an update
- **Register:** Displays Registration screen
- **About:** Displays information window

Editor Toolbar



- **Run:** compiles program, loads to your hardware to RAM and starts the connection
- **Debug:** compiles program with debug information, loads to your hardware to RAM and starts the connection in debug mode
- **Pause:** Pauses the active cog and shows the current line
- **Stop:** Stops the debugger and connection
- **Step Into:** Steps one instruction further
- **Step Over:** Step one instruction/function call further
- **Step Out:** Step out of current function
- **Save:** Saves file
- **Bookmark:** creates/removes a bookmark at the current locaiton
- **Previous/Next Bookmark:** moves cursor to previous/next bookmark
- **Port:** Select which COM port to use. Defaults to "Auto"- to automatically find your hardware. Once found, it'll stay in Auto mode, but indicate the current port used, by displaying something like "COM6*". This indicates it's currently using COM6, but is still in AUTO mode, so when you remove your hardware from COM6, it'll search all ports again. You can also select a fixed COM port to only use that port.

5.3 Folder Structure

The ViewPort installer checks for a valid .net Framework installation, presents the license agreement and installs files as follows:

Program Files/ViewPort Directory

- **ViewPort.exe**
  - core application
- **Readme.txt**
  - contains information about the release
- **unins000.dat,unins00.exe**
  - uninstall files
- .**tpl**
  - template files used when you create a new file
- **/lib**
  - ViewPort stores library files included by your program here (conduit.spin...)
- /**Plugins**
  - **.chm:** Information about a plugin
  - **.vpc:** Install/Uninstall information about a plugin
- **/View**
  - **.xml files:** layout data for each view
  - **hierarch of image files:** files used by views

My Documents/ViewPort

- **/Tutorials**
  - **.spin .c .bs2** sample ViewPort files

5.4 Tutorials

The following tutorials are included with ViewPort. They're designed to help you understand how to use ViewPort with your own projects. Here's a brief overview of each:

Debug	Learn how to use the Spin Debugger with this sample. Step through the code, set breakpoints and watch variables. Start by clicking the "code" tab and reading the comments.
Four Bit Counter	When ViewPort connects to this program, it will display a logic analyzer view showing 6 traces. 4 count in binary while the bottom two show the status of teh serial connection. Vary the speed of the counter with the bottom right dial or change the trigger or timescale.
Track a Spin Variable	Displays a sawtooth signal in the oscilloscope view. This signal represents the value of a variable in the spin program. Manipulate the trigger, time scale and measure the signal with the cursors.
RS232	The program sends two bytes using RS232. The first byte's value slowly increases. You can control the second with a textbox control in ViewPort. The "mixed" view shows both the logic traces and the values being sent.
High Frequency	This program uses one of the two hardware counters per cog to generate a square wave with a configurable frequency from 1 to 40MHz- controllable through a ViewPort dial. The waveform is displayed in the "analog" view- showing the signal and its spectrum analysis.
Simulated Weather Station	Displays simulated weather measurements in a custom view with custom controls.
Fuzzy Logic Lunar Lander	This applet simulates Height, Speed, and Throttle for two landers-one under your control, the other controlled by Fuzzy Logic. See if you can beat the Propeller!
External Signal- Binary Counter	This program shows how you can use ViewPort's oscilloscope and logic analyzer modes to analyze a signal from an external integrated circuit- the MC14029 (binary counter), available on Digikey for $.65
Measure Distance	The Parallax PING ultrasonic sensor is a great tool to measure distance up to 3m. This program uses ViewPort's oscilloscope to graph the sensor's value over time. You can also use the Logic Analyzer view to time how long it takes for the echo to return to the sensor.
Measure Fast Analog Voltage	An Analog to Digital converter is used to convert an analog value to digital. This program uses the QuickSample object to quickly sample the the outputs of an external high-speed ADC. ViewPort can decode the binary representation of the analog signal and graph it as an analog value. An interesting pattern to look at is an NTSC TV signal.
Video Tracker	This program uses the VideoCapture object in 1 cog to capture video signals and display a video stream within ViewPort. Another cog uses simple vision processing to find a bright spot and highlight it in ViewPort.
Vision Demo	This program shows how to use the Vision object to apply simple algorithms to captured Video. A spin program specifies the algorithms to run inside the Vision engine.
Measure Capacitance	Use this Program to measure the capacitance of a capacitor using a simple RC circuit.
Spectrum of 2 Signals	Analyze a composite signal with the spectrum analyzer to determine the frequencies of the source signals.
Terminal	Demo of the terminal replacement object.
OpenCV	Advanced vision processing with the OpenCV library
Lissajous	Uses the XY view to plot two signals which result in Lissajous figures.
DDE Sample	Shares arrays and variables for use with DDE clients
Debug Pin Toggle	Toggles pins with a variable- useful for learning how to use the debugger.
Manchester	Generate and decode Manchester encoded signals.
First C program	C program will blink the LED's on a Propeller DemoBoard.
BS2 Blink	BS2 program to blink LED's with a BS2

5.5 Communication Protocol

Receiving Data
The Conduit object continually sends packets marked by out-of-band markers at the set baud rate. ViewPort discards any packets where the markers are out of alignment.

Sending Data and Commands
ViewPort sends data and commands to the conduit. The conduit times the start bit to set its baud rate.

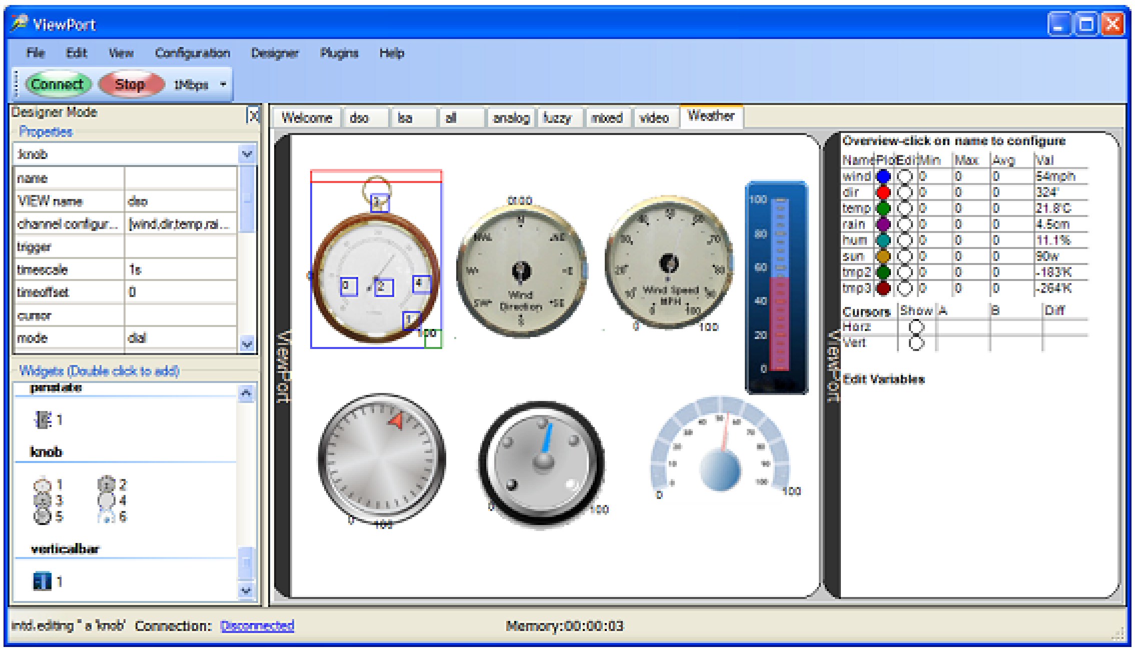
Reading Configuration
ViewPort sends a READCFG command to the conduit which returns configuration string and length of memories being shared. ViewPort then parses the configuration data to initialize the ViewPort view

Loading Firmware
ViewPort opens the selected Port (defaults to Auto which will try all COM ports) and tries to identify your hardware. It issues a RESET and proceeds to load the compiled program.

Detecting the Propeller
ViewPort issues a RESET and issues a VERSION command to the microcontroller. A valid response indicates the Propeller/BS2 is on that port.

5.6 View Customization

Each ViewPort view is specified by a separate xml file. The file specifies the layout, graphical attributes and behavior of a collection of widgets. ViewPort ships with many widgets (switches, scrollers, dials, dso, lsa) but you can also develop your own using the Widget Development Kit- available at: http://hannoware.com/viewport/plugins. You can right click on widget's to set some of their properties, edit the xml file by hand, or use the ViewPort Designer to customize views.



knob

name	
VIEW name	dso
channel configur...	[wind,dir,temp,rai...
trigger	
timescale	1s
timeoffset	0
cursor	
mode	dial

pinstate
knob
verticalbar

Overview-click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
wind	●	○	0	0	0	54mph
dir	●	○	0	0	0	324'
temp	●	○	0	0	0	21.8'C
rain	●	○	0	0	0	4.5cm
hum	●	○	0	0	0	11.1%
sun	●	○	0	0	0	90w
tmp2	●	○	0	0	0	-183'K
tmp3	●	○	0	0	0	-264'K

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

Edit Variables

intd.editing " a 'knob' Connection: Disconnected Memory:00:00:03

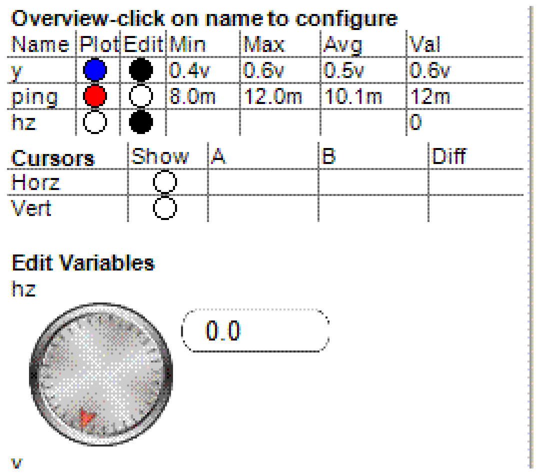
To visually edit a View, turn on the designer mode from the "Designer" menu. In the "Designer" mode, the left column shows the properties of the currently selected widget and a library of widgets. Double click a widget to add it to your view. Change a widget's properties by clicking on the value column and selecting a new value. Once a widget is selected it will be framed with a blue rectangle. Drag the red top bar to move the widget. Drag the green square to resize the widget. Drag the blue numbered squares to configure the widget.

You must purchase a license that supports Designer to save your changes.

5.7 Data Widget

The Data Widget includes:

- **Data Overview**
  The most recent value and measurements from the active graph are shown for each shared value. The Plot and Edit columns allow you to quickly plot the variable in the active graph and/or edit the variable. Click on the variable name to start the configuration wizard. The configuration wizard (described in next section) allows you to change properties for a variable- like its display name, its unit and more.
- **Cursor Measurements**
  Click the "show" column to turn a cursor on/off. The measurements are shown in the A,B, and Difference columns
- **Edit Controls**
  Controls like textboxes, dials, and scrollbars let you change a variable's value from this section.



Overview-click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
y	🔵	●	0.4v	0.6v	0.5v	0.6v
ping	🔴	○	8.0m	12.0m	10.1m	12m
hz	○	●				0

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

Edit Variables

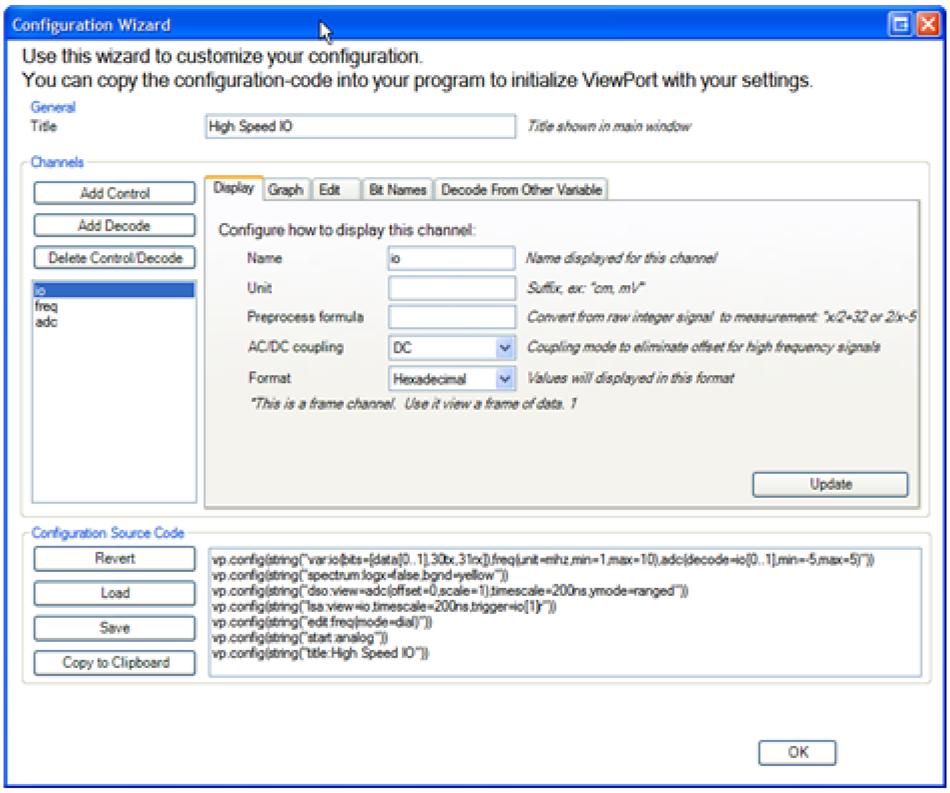
hz

0.0

v

5.8 Configuration Wizard

ViewPort uses configuration strings to define how data channels from the Propeller are displayed For example, a channel's name, controls, and graphic properties are all defined by configuration strings. You can include the configuration strings in your spin program to set up ViewPort the way you like. The configuration wizard helps beginners create and understand these strings. Start by watching how the strings change as you try out the tutorials. Then, connect to your own program (remember to include vp.share) and use the wizard to assign names and other properties to your shared channels. Once everything is set up in ViewPort, copy the strings into your program.



Use this wizard to customize your configuration.
You can copy the configuration-code into your program to initialize ViewPort with your settings.

Title: High Speed IO — *Title shown in main window*

io
freq
adc

Configure how to display this channel:

Name: io — *Name displayed for this channel*
Unit: — *Suffix, ex: "cm, mV"*
Preprocess formula: — *Convert from raw integer signal to measurement: "x/2+32 or 2/x-5*
AC/DC coupling: DC — *Coupling mode to eliminate offset for high frequency signals*
Format: Hexadecimal — *Values will displayed in this format*
"This is a frame channel. Use it view a frame of data. 1

```
vp.config(string("var:io(bits=[data[0..1],30tx,31rx]),freq(unit=mhz,min=1,max=10),adc(decode=io[0..1],min=-5,max=5)"))
vp.config(string("spectrum:logx=false,bgnd=yellow"))
vp.config(string("dso:view=adc(offset=0,scale=1),timescale=200ns,ymode=ranged"))
vp.config(string("lsa:view=io,timescale=200ns,trigger=io[1]r"))
vp.config(string("edit:freq(mode=dial)"))
vp.config(string("start:analog"))
vp.config(string("title:High Speed IO"))
```

The channels section shows the channels you shared with ViewPort. You can add/delete control and decode channels to this list. The tabs to the right are specific to the selected channel.

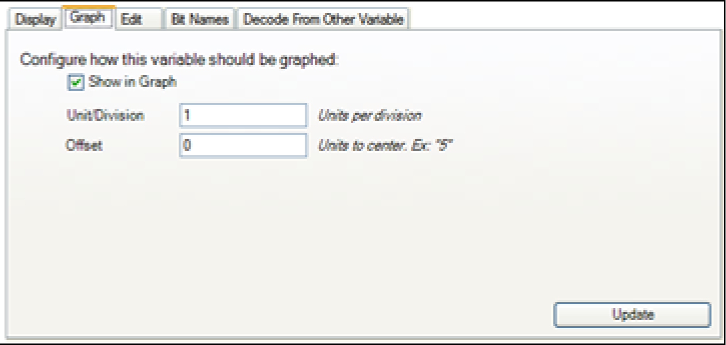
Display Tab

- **Name:**the name to display for that channel
- **Unit:**Suffix for the channel's value
- **Formula:** Values from Propeller represent measurements of things that have units. Temperature in 'C, Voltage, distance in cm. The propeller sends all values as integers. To convert to a unit, a scaling factor and offset is applied. The formula must look like this: scale*x+offset or this: scale/x+offset, where scale and offset are real numbers. Examples: 10.2*x+5.2, 10.2/x+5.2
  Besides floating numbers, you can also use other channel names to scale/offset the variable.
- **AC/DC mode:** In AC mode, a variable's value is first low-pass filtered to remove the DC offset. Use AC mode if you're monitoring high frequency values and you don't care about the low frequency offset.
- **Formula:** display value in base 10 (decimal), engineering notation, 16 (hexadecimal), 2(binary) as a string, or as a float.

5.9 Configuration Wizard Part 2

Graph

- **Show in Graph:** toggle to include/remove from active graph.
- **Scale:** resolution/division in units.
- **Offset:** units for bottom of graph.



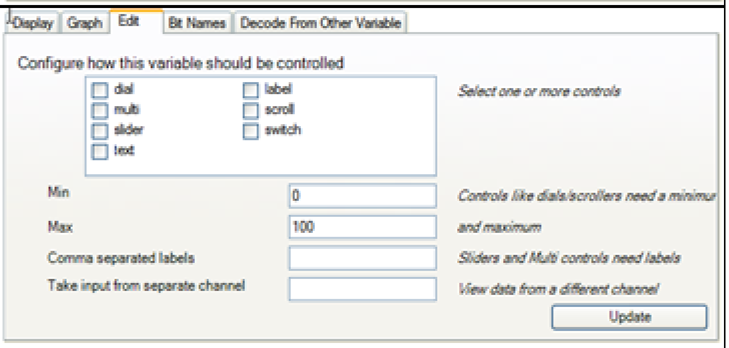
Configure how this variable should be graphed:

☑ Show in Graph

Unit/Division	1	*Units per division*
Offset	0	*Units to center. Ex: "5"*

Edit

- **List of Controls:** choose which controls to use for editing the variable.
- **Min:** sets the minimum value sent by scrollbar and dial knob.
- **Max:** sets the maximum value sent by scrollbar and dial knob.
- **Labels:** labels assigned to controls-only used with the "slider" control.
- **Input:** the value shown on this control can be from a different channel



Configure how this variable should be controlled

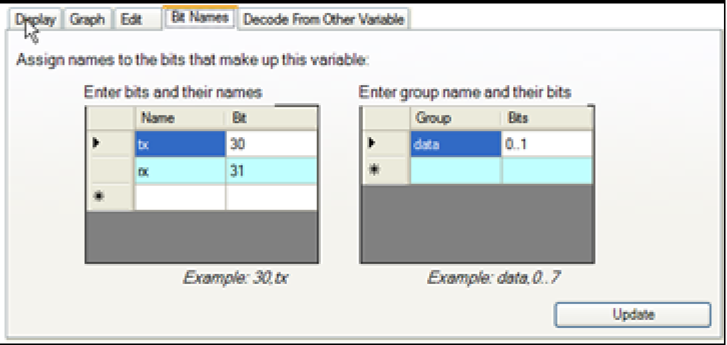
☐ dial ☐ label
☐ multi ☐ scroll
☐ slider ☐ switch
☐ text

Select one or more controls

Min	0	*Controls like dials/scrollers need a minimu*
Max	100	*and maximum*
Comma separated labels		*Sliders and Multi controls need labels*
Take input from separate channel		*View data from a different channel*

Bit Names

- **Table of Bits and Names:** label individual bits for display in the LSA widget
- **Table of Groups and Bits:** label groups of bits for display in the LSA widget



Assign names to the bits that make up this variable:

Enter bits and their names

Name	Bit
tx	30
rx	31

Example: 30,tx

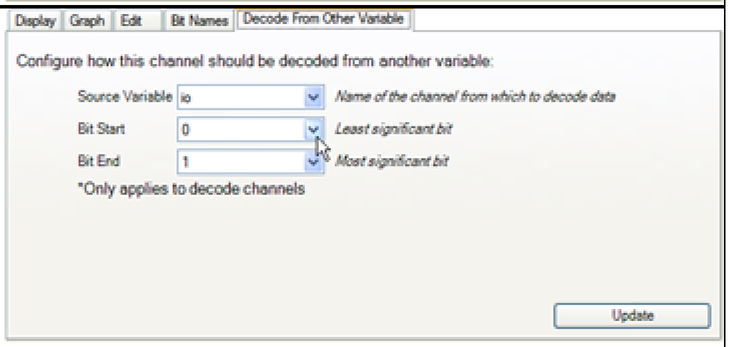
Enter group name and their bits

Group	Bits
data	0..1

Example: data,0..7

Decode from Other Variable

- This tab only applies to "decode channels". Click on "Add Decode Channel" to add one.
- **Source Variable:** decode value from this source variable
- **Bit Start:** this bit will be the lsb
- **Bit End:** this bit will be the msb



Configure how this channel should be decoded from another variable:

Source Variable	io	*Name of the channel from which to decode data*
Bit Start	0	*Least significant bit*
Bit End	1	*Most significant bit*

*Only applies to decode channels

The configuration strings are continually updated in the "Configuration Source Code" panel. You can revert to the original configuration provided at the last connect, load/save configurations, and copy the strings to the clipboard to paste into your source code.

5.10 Channel Types

Besides interacting with data that map directly to variables shared in a Propeller program, ViewPort also supports frame, control decode and array channels.

- A **Variable Channel** is created when you share a variable. You can graph this data and edit the variable's value. Sharing more variables will decrease the effective sampling rate of the streamed data.
- A **Frame Channel** is created when you register the QuickSample. Doing so will allow you to sample the IO port at fast rates, but slow the sampling rate of the other streamed variables channels. You can't edit the frame channel.
- A **Decode Channel** is used when the value you want to display is "embedded" in another value. For example, when you read a binary counters value with pin 10-17 of the Propeller, the counter's value is embedded in bits 10-17 of the INA variable. A Decode Channel gets its values by decoding a source channel with set parameters- in our case- masking out bits 10-17 and shifting right by 10. Since it's value is computed on the fly within ViewPort, it does not affect the sampling rate and you can't edit this channel.
- A **Control Channel** is used when you need to change variables in Propeller memory but don't need to stream them back to the PC- ie parameters as opposed to sensor readings. The variables edited by control channels start with the one after the last shared variable. For example, if you allocate these variables:
  ```
  main |a,b,c,d
  ```
  and share just a and b with this statement
  ```
  vp.share(@a,@b)
  ```
  then the first control channel will map to variable c, and the second to variable d
- **Arrays** can be shared using the `vp.array(@array,arraylength,arraytype)` The first element of the array can be graphed inside of ViewPort. The full array is accessible through DDE. The arraytype should be a unique number starting with 5.

5.11 Graph Configuration

Each graph maintains this information:

- **Variables** being viewed: the list of variables to be plotted by this graph. For each variable you can specify its scale, offset and coupling.
- **Time scale:** the horizontal resolution of the graph.
- **Time offset:** when triggered, this represents the offset from the trigger point. When data is sampled, this offset is from the start of the sample. When in realtime mode, the graph label shows the date and time. In normal mode, the offset is relative to the most recent sample- which is taken at time 0.
- **Trigger:** You can configure each trigger separately- there are 4 modes:
  - **Bit Mode:** Select R/F for rising/falling edge, then select the bit
  - **Pattern Mode:** Use this mode to trigger on multiple bits. Enter a string of characters- 1 for each bit. The right most character corresponds to the least significant bit. 0=low, 1=high, r=rising, f=falling, x=don't care
  - **Value Mode:** Use this mode to trigger on a rising/falling decimal value. Set a value and choose to trigger on rise/falling edge
  - **Expression Mode:** Use this mode to trigger on matching values. Choose a qualifier and set a value. Qualifier can be: = to equal the value, > to rise, < to fall, d to be different, g to be greater, l to be less than.

Some views include multiple graphs- this allows them to data with different variables, time scales, triggers and offsets.

5.12 Widget Development Kit

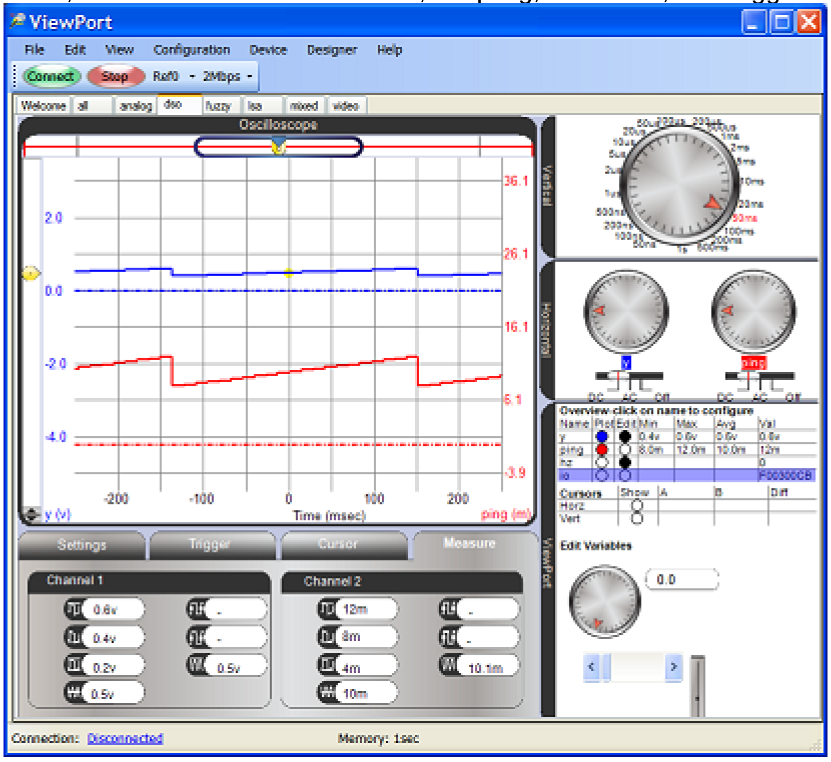
The ViewPort Development Kit allows you to build plugins for ViewPort to let users interact with data in new ways. Widgets have access to all of ViewPort's data, including data values, meta data about each channel, and general data. Widgets can access ViewPort controls like the timescale selector and can send data back to the Propeller. Developers can package graphic files, views, dll's and a help file into a vpc plugin file for easy distribution. Users install a plugin by opening it with ViewPort.

The Kit is only made available to registered users of ViewPort + Developer.

More details available at: http://hannoware.com/viewport/plugins

5.13 DSO Widget

The oscilloscope graphs the value of one or more variables over time. You can change the time scale, time offset vertical resolution, coupling, autoscale, and trigger mode.



Overview click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
y	●	●	0.4v	0.6v	0.6v	0.6v
ping	●	○	8.0m	12.0m	10.0m	12m
hz	○	●				0
io	○	○				F00300C8

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

Edit Variables: 0.0

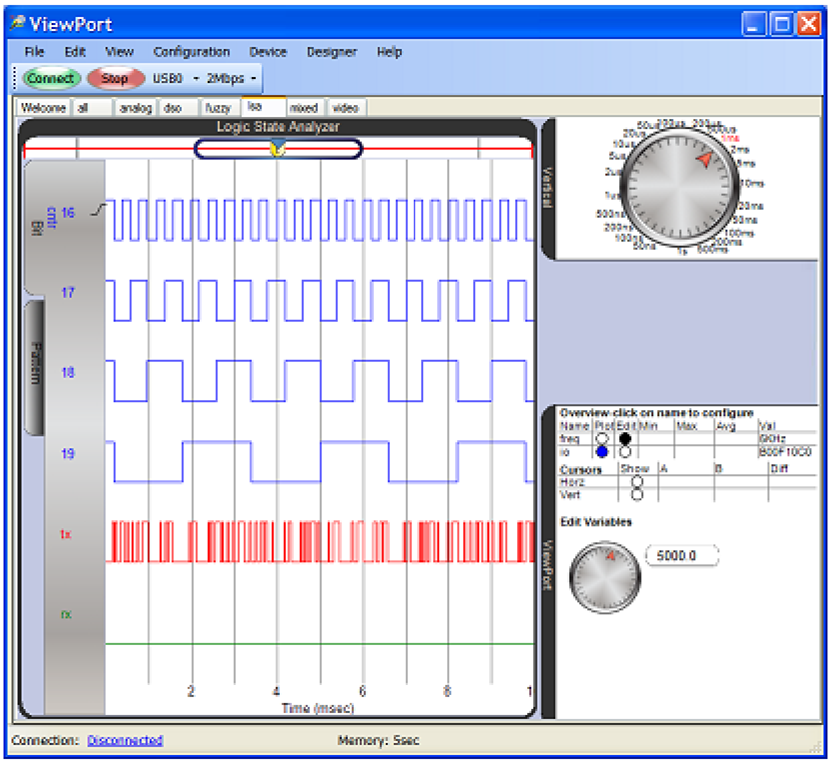
Channel 1		Channel 2	
0.6v	-	12m	-
0.4v	-	8m	-
0.2v	0.5v	4m	10.1m
0.5v		10m	

Connection: Disconnected   Memory: 1sec

AutoScale automatically scales and center each trace so it fits on the screen at the right place. Of course you may override these by dragging the traces with your mouse or changing the resolution.
You can move a channel trace my dragging it with the mouse. You can scroll back in time by using the scroll bar at the bottom or dragging the grid back and forth. To change the time resolution use the timescale control. Add and change value-based triggers by clicking/dragging just to the left of the graph. Use cursors to measure the signal. Click the "Connect"/"Stop" buttons to start or stop the connection. Auto measurements display the signals amplitude, limits, average, period, frequency, and root mean square.

5.14 LSA Widget

The logic analyzer graphs individual bit traces over time. Since the Propeller is a 32 bit processor, ViewPort graphs up to 32 bit traces- but you can configure the variable to label individual bits or groups of bits. Unlike real instruments, ViewPort lets you use the LSA on variables running inside the Propeller, not just the IO port.



Overview click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
freq	○	●				0KHz
io	●	○				B00F10C0

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

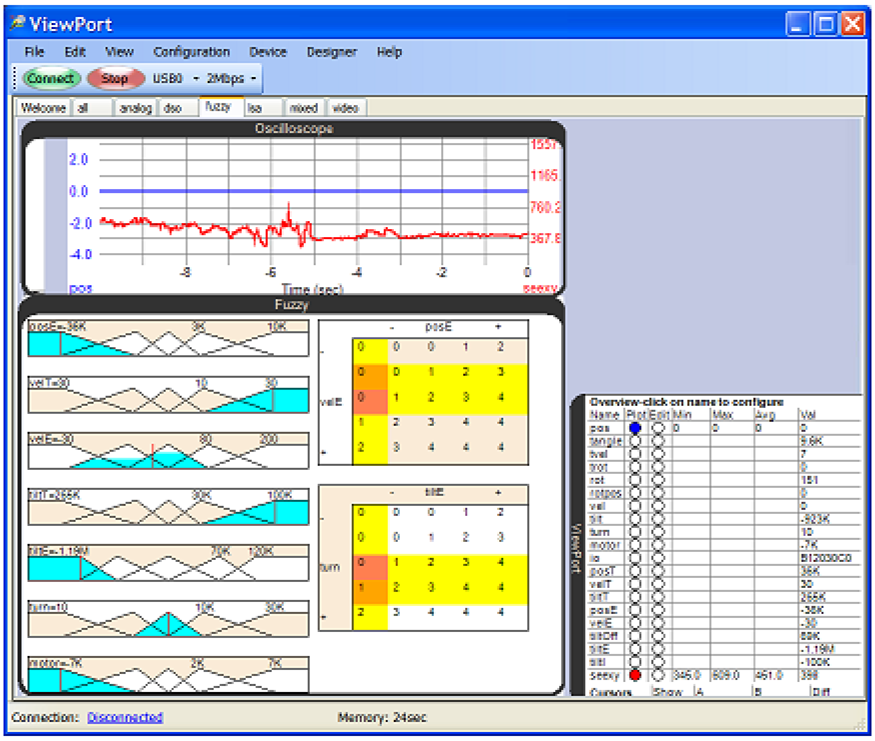
Edit Variables

5000.0

The graph shows the individual bit traces- numbered and labeled on the left. You can change the timescale, time offset, and choose which channel to graph. You can move the traces horizontally by dragging them with the mouse or using the top scrollbar. To change the time resolution use the timescale control. Add and change single bit/edge/pattern triggers by clicking on the bit's label. Use cursors to measure the signal.

5.15 Fuzzy Logic Control Panel Widget

Fuzzy Logic simplifies some control problems by making control values understandable to humans. See: http://en.wikipedia.org/wiki/Fuzzy_system on the web, or the Fuzzy Logic Concepts section, or the section on the Fuzzy Logic Engine Object. This page covers the Fuzzy Logic Control Panel.



		-		posE		+
-		0	0	0	1	2
		0	0	1	2	3
velE		0	1	2	3	4
		1	2	3	4	4
+		2	3	4	4	4

		-		tiltE		+
-		0	0	0	1	2
		0	0	1	2	3
turn		0	1	2	3	4
		1	2	3	4	4
+		2	3	4	4	4

Overview-click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
pos	●	○	0	0	0	0
tangle	○	○				9.6K
tvel	○	○				7
trot	○	○				0
rot	○	○				151
rotpos	○	○				0
vel	○	○				0
tilt	○	○				-923K
turn	○	○				10
motor	○	○				-7K
io	○	○				B12030C0
posT	○	○				36K
velT	○	○				30
tiltT	○	○				265K
posE	○	○				-36K
velE	○	○				-30
tiltDiff	○	○				89K
tiltE	○	○				-1.19M
tilti	○	○				-100K
seeky	●	○	345.0	609.0	451.0	398

Cursors	Show	A	B	Diff

When your program includes a shared fuzzy logic engine, ViewPort enables the fuzzy logic control panel view. This view shows an oscillloscope on top and the overview panel on the right. The Fuzzy panel displays fuzzy maps and fuzzy rules.

Fuzzy Maps

Fuzzy maps are graphed on the left. In the image above, you'll see 7 maps. Each one maps a variable's value onto 5 classes. The vertical red line in each map indicates the exact value of the variable. The area in blue indicates the proportion of membership of each class. If the area is completely blue, membership is 100%. If the variable's value lies between 2 classes, both classes will have some blue- see the 3rd map in the image above. You can change how classes are mapped in real time by clicking on the value and typing a new value.

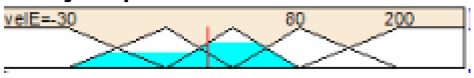
Fuzzy Rules

Fuzzy rules are graphed on the right. You'll see 2 rules in the image above. Rules use 2 fuzzified variables as input to a lookup table. The inputs are indicated as horizontal and vertical axis. You can click and change the values of the lookup table to change the rules behavior. When your program is running, the yellow/orange/red areas indicate the proportion of the rule that's firing- the red area has the highest impact, then orange, then yellow.

5.16 Fuzzy Logic Concepts

Fuzzy Logic is a technique to efficiently solve some control problems. ViewPort comes with a Graphical Control Panel view found in the "Fuzzy" view, and a Fuzzy Logic Engine implemented in the "fuzzy.spin" object. ViewPort's fuzzy logic implementation consists of fuzzy maps, fuzzy rules, and fuzzy logic functions.

Fuzzy Maps

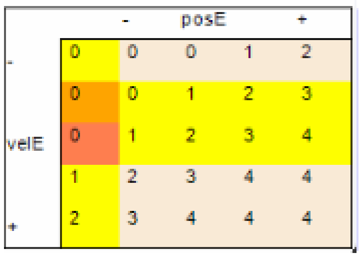


Fuzzy Maps "fuzzify" a number, by mapping it to 5 classes. In the image above there are 5 classes to which a number can belong:

	Class Minimum	**"Center"**	**Class Maximum**
Class 0: Very negative	-infinity	-200	-80
Class 1: Negative	-200	-80	0
Class 2: Zero	-80	0	80
Class 3: Positive	0	80	200
Class 4: Very Positive	80	200	infinity

Notice that a number can belong to 1 or 2 classes. For example, -30 (as shown above) belongs mostly to the "zero" class, but also to the "negative" class. The red line indicates the exact value, the blue level shows degree of membership of the class. Only 2 parameters are required for each map since the center for class 2 is 0 and the classes are symmetric around class 2. These parameters (the centers for class 3 and class 4: 80 and 200 in our example) are provided as a parameter to the "setMap" function for the "Fuzzy" spin object. They are also shown in the control panel and can be edited. The fuzzy logic engine has 2 "fuzzy registers", A and B. To fuzzify an integer into A, use "fuzzifyA(valueToBeFuzzified, mapNumber)" where mapNumber is the number set up by the "setMap" function. "FuzzifyB" does the same for fuzzy register B. Most of the time you will want to defuzzify the result of a fuzzy calculation with the "deFuzzify(mapNumber)" function. This maps the fuzzy register A back to a number using the specified map. You could also call "getTopClass" which returns the class number in register A with the highest membership:0..4

Fuzzy Rules



		-		posE		+
-		0	0	0	1	2
		0	0	1	2	3
velE		0	1	2	3	4
		1	2	3	4	4
+		2	3	4	4	4

Fuzzy Rules allow you to control 1 output variable using 2 input variables and a lookup table. You need to specify the input variables and lookup table using the "setRule" function. The contro panel allows you to change the "result" values in the lookup table by clicking to edit the value. Calling the "doAnd" or "doOr" functions will calculate the result by looking up all classes which have some fuzzy membership and applying the appropriate fuzzy logic. In the above example, posE is "very negative" and velE is mostly "zero", so the result of a "doAnd" call will be class "0". When your program is running, the red square tells you which result has the highest weight. Fuzzy logic ensures that the output function is smooth and continuous across the input space.

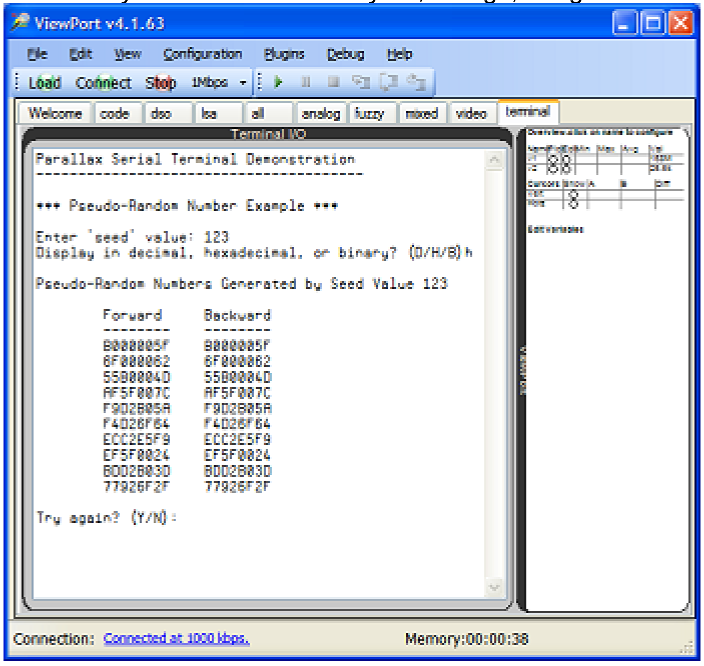
Fuzzy Logic Functions

The engine also supports boolean functions: doAnd, doOr, and doInv. doInv takes the fuzzy inverse of A and returns it into A. doAnd and doOr take the fuzzy AND/OR of registers A, B and place the result into A.

Lastly, to share the fuzzy logic engine with ViewPort, use the vp.register(f.start(@FuzzFrame)) line after initializing all your fuzzy maps and rules. Then, call f.share(@FuzzFrame) once in your control loop to share the data.

5.17 Terminal

ViewPort includes a terminal widget that shows the terminal output of your program when using the terminal object. Entering text into the text area will send the keystrokes to the Propeller where they can be read in as bytes, strings, integers and floats.

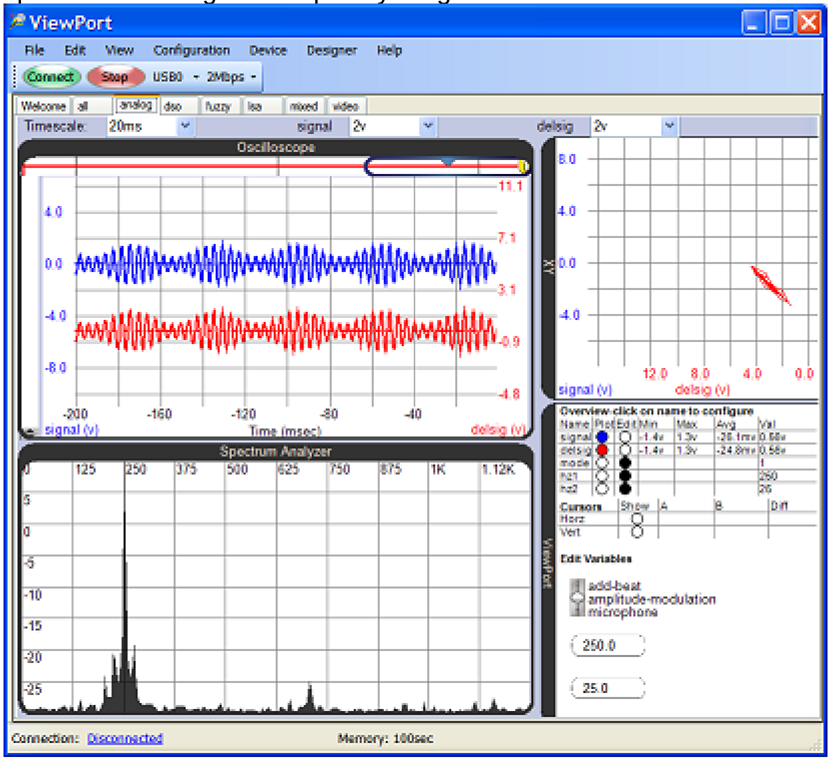

ViewPort v4.1.63
File Edit View Configuration Plugins Debug Help
Load Connect Stop 1Mbps
Welcome code dso lsa all analog fuzzy mixed video terminal
Terminal I/O

```
Parallax Serial Terminal Demonstration
----------------------------------------

*** Pseudo-Random Number Example ***

Enter 'seed' value: 123
Display in decimal, hexadecimal, or binary? (D/H/B) h

Pseudo-Random Numbers Generated by Seed Value 123

        Forward     Backward
        --------    --------
        B000005F    B000005F
        6F000062    6F000062
        5500004D    5500004D
        AF5F007C    AF5F007C
        F9D2B05A    F9D2B05A
        F4D26F64    F4D26F64
        ECC2E5F9    ECC2E5F9
        EF5F0024    EF5F0024
        BDD2B03D    BDD2B03D
        77926F2F    77926F2F

Try again? (Y/N):
```

Connection: Connected at 1000 kbps.   Memory:00:00:38

5.18 Spectrum Analyzer Widget

Use the Spectrum Analyzer to analyze the spectral composition of a variable's values over time. Start with the Oscilloscope to ensure you signal is properly scaled and triggered, then view the signal and its analysis on the spectrum analyzer. The spectrum analyzer displays a power spectrum over a given frequency range in real time.



Overview-click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
signal	●	○	-1.4v	1.3v	-26.1mv	0.58v
delsig	●	○	-1.4v	1.3v	-24.8mv	0.58v
mode	○	●				1
hz1	○	●				250
hz2	○	●				25

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

Edit Variables

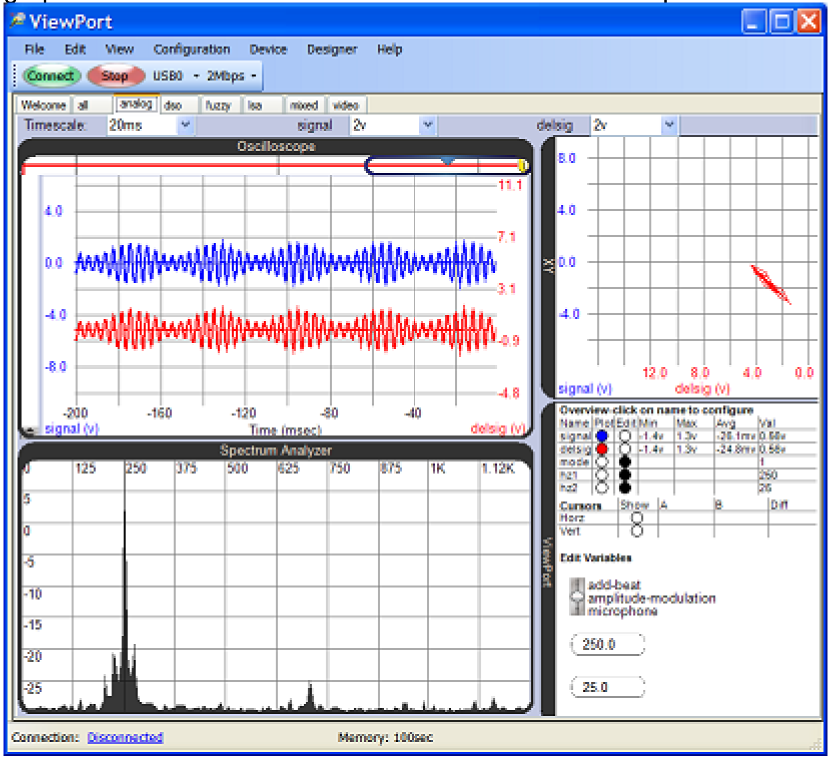
add-beat
amplitude-modulation
microphone

250.0

25.0

5.19 XY Widget

Use the XY graph to analyze how 2 signals relate to each other across time. The signals are assigned to the X and Y axis and the values are plotted one sample at a time. Start with the Oscilloscope to ensure your signal is properly scaled and triggered, then view the signal in the XY graph. Move and scale the waveform's in the oscilloscope mode to suit the XY graph.



Overview-click on name to configure

Name	Plot	Edit	Min	Max	Avg	Val
signal	●	○	-1.4v	1.3v	-26.1mv	0.56v
delsig	●	○	-1.4v	1.3v	-24.8mv	0.56v
mode	○	●				1
hz1	○	●				250
hz2	○	●				25

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

Edit Variables

add-beat
amplitude-modulation
microphone

250.0

25.0

5.20 Debugger

The Spin Debugger greatly simplifies debugging code. It lets you stop the program at a given point (a breakpoint) or pause it at the press of a button. You can step one line of code at a time while you're watching your variable values change in real time. The call stack and profiler tell you how your program got to it's current state and how much time is spent in each function.

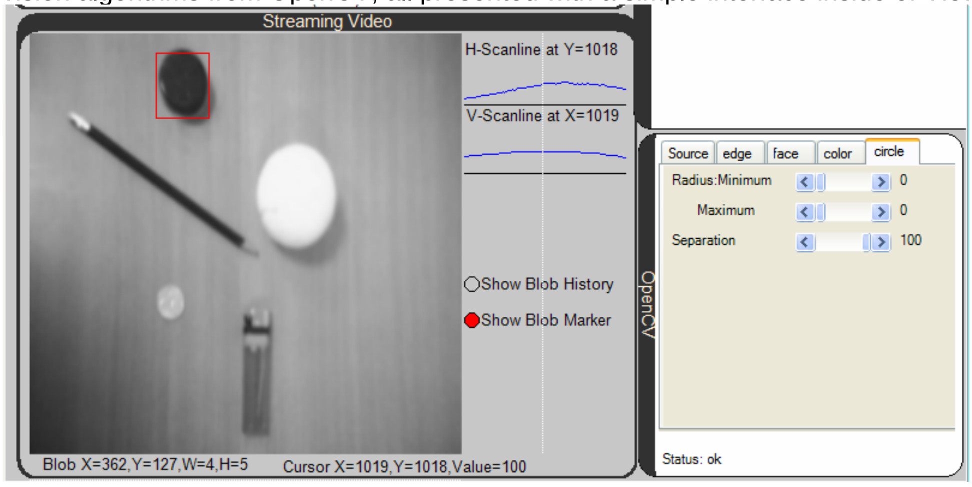
The Debugger is part of the "code" view, so start by clicking the "code" tab in the main ViewPort window. You should see your current spin file with familiar "Propeller Tool" syntax highlighting.

After pressing the triangular "play" button to "start debugging", you can:
- set a breakpoint by clicking on the line number you wish to pause the program at. Once the cog running your program reaches this line, it will pause execution
- resume from a breakpoint by clicking the "play" button again. To remove your breakpoint, click the line number again.
- pause your code where it's currently executing by clicking the "pause" button.
- step into functions called by a line of spin code by pressing the "step into" button
- step over a line of spin code by pressing the "step over" button
- step out of a function by pressing the "step out" button
- view the call stack window to see which functions were called to get to the current state
- the watch window shows variables you've shared with ViewPort- including their address in main Propeller memory and their value. Click on the address to scroll the "Memory" window to that address. Click on the value to change it.
- the profiler shows how much time is spend in different functions
- the memory shows a complete snapshot of the Propeller's 32kb main memory- taken at each step
- the command interpreter window lets you interact with the debugger:
- "h": for a help listing
- "set VAR=VALUE": sets variable named "VAR" to "VALUE"
- "print VAR": prints the value of variable named "VAR"
- "r":run until breakpoint
- "s":step 1 line of spin code
- "sN":step N lines of spin code
- "p" : pause execution
- "w VAR=<>VALUE":conditionally runs while variable "VAR" =<> "VALUE"
- "u" VAR=<>VALUE":conditionally runs until variable "VAR" =<> "VALUE"
- "a":animates the program, by taking ~5 steps/second
- mouse over a shared variable to see it's value or change it.
- load a file by clicking "File:Open" or selecting from the file browser
- save a file by clicking "File:Save"
- view a spin project's objects in the Object view
- view a spin file's documentation by clicking the "Documentation" button
- manage multiple spin files in tabs, close with the "x" button.
- use any of the other ViewPort windows while you're debugging

To use the Debugger on your own programs you should:
- use the 80MHz clock mode
- declare vp:"Conduit" as an object
- call vp.share with the address of memory you wish to watch

5.21 OpenCV

The OpenCV integration lets you easily experiment with state-of-the-art computer vision with your ViewPort/Propeller combination. Find x,y of faces, colored blobs, circles, textures all using spin code!
OpenCV has been the leading Computer Vision library for 10 years, it was used by Stanford to win the DARPA race. Until now, it was difficult to do vision processing with OpenCV and control real-world devices. With the ViewPort integration, people will have the best of all worlds- easy integration with all sorts of real world sensors and actuators with the Propeller and state of the art vision algorithms from OpenCV, all presented with a simple interface inside of ViewPort.



Blob X=362,Y=127,W=4,H=5 Cursor X=1019,Y=1018,Value=100

Radius:Minimum 0
Maximum 0
Separation 100

Status: ok

To get started, click on the "video" view inside of ViewPort. This will bring up the "video" view with a "video" panel and an "opencv" control panel. Use the "openCV" panel to control from which source (image file, video file, webcam, propeller frame grabber) video should be shown. Click the "filter" tabs (edge, face, color, circle...) to apply different filters to the video. If the video has already been filtered by the Propeller Vision Filter, it will show that result. Each "filter" tab has additional controls- play around!
For info on capturing video with the Propeller look at the VideoCapture object.
For info on using a video file, visit: http://opencv.willowgarage.com/wiki/VideoCodecs
For info on filtering video with the propeller look at the Vision Filter object.
If you have a webcam connected that works with other Windows applications like Skype, it should work here- you may have to try different numbers in the box next to "Webcam". Try 0,1,100,101,200,201...
The video output will show the filtered video, the location of found objects, and the cursors location and value- all standardized to return numbers between 0 and 1000.

Part VI

Reference (Propeller Objects)

6.1 Overview

There are 6 components in the ViewPort library.

- **Conduit** runs in one cog and is mandatory. It moves data back and forth to and from the host computer and manages the ViewPort system on the Propeller.
- **QuickSample** runs in either 1 or 4 cogs and takes quick measurements of the INA port.
- **Fuzzy Logic Engine** is a library of spin functions which provide fuzzy logic operators. It does not use a separate cog.
- **PropCVCapture** runs in 1 cog and captures video data into an array.
- **PropCVFilter** runs in 1 cog to apply vision filters to the memory array
- **Terminal** is a wrapper around the Conduit object- use it instead of Conduit when you need a terminal session as well as ViewPort.

Propeller Resource Requirements
At minimum, 1 cog is required to integrate with ViewPort- this cog will run the ViewPort Conduit code. Additional components may require additional cogs- as specified above. ViewPort requires 2 Pins for serial communication with the host system. The code for ViewPort objects requires relatively little global memory space. However, the QuickSample, VideoCapture and Vision objects require memory space for the sample measurements and video data- use the Propeller Tool to make sure you're program doesn't exceed the Propeller's memory bounds.

6.2 Conduit

Your Program must include and start this Object to use ViewPort. This Object allows you to register additional components, share data, configure and start ViewPort.

Features:
-Transfers data at up to 2Mbps Full-Duplex over a Parallax USB-Serial connection or
 115Kbps over other Serial connections.
-Manages configuration of variables and registration of ViewPort objects
-Automatically switches between baud rates depending on ViewPort setting

Use:
-Include this object in your program (MANDATORY):

```
Obj vp:"Conduit"
```

-Share Data and Start (mandatory, but must be last vp command in your program):

To share data, ViewPort needs to know the location of memory to track. The share command lets you specify the start and end of memory to share:

```
vp.share(@varA,@varC) 'one cog is started to share memory between varA and varC
```

-Share Arrays (optional)

The first element of the array can be graphed inside of ViewPort. The full array is accessible through DDE. The arraytype should be a unique number starting with 5.

```
vp.array(@array,arraylength,arraytype)
```

-Register Components (optional)

If you're using other ViewPort components like QuickSample or VideoCapture, you need to register them with ViewPort:

```
vp.register(startValue) 'the start function of each components provides the StartValue
```

Configure ViewPort (optional):

All configuration is optional and can be performed from within the ViewPort interface. You may configure it here to start ViewPort in a certain mode. See Configuration String section for more detail- or use ViewPort to create the configuration for you.

```
vp.config(configStr)
```

6.3 QuickSample

Use this Object to take measurements of INA at very high speeds.
Users can use ViewPort to analyze those measurements alongside other variables from their program.

Features:
-Measure 1440 samples at 32bit up to 80MHz using 4 cogs.
-Measure 360 samples at 32bit up to 20MHz using 1 cog.
-Flexible timescale from seconds to nanoseconds
-Edge and Pattern Triggers- can be reset by ViewPort if not Trigger found
-Samples are continuously taken by 1 or 4 interleaved cogs running self modifying assembly code.

Use:
-Include this object in your program:

```
Obj qs:"QuickSample"
```

-Allocate memory for the INAFrame:

```
LONG INAFrame[400]
```

Allocate ~400 longs for the INAFrame if using 1 cog, or 1600 if using 4.
-Register with ViewPort before the vp.start command

```
vp.register(qs.sampleINA(@INAFrame,NumberOfCogs))
```

NumberOfCogs should be 1 cog to sample up to 20MHz, or 4 to sample up to 80MHz

Frame Internals
The first 10 longs are configuration variables managed by ViewPort for timescale, trigger, as well as synchronization markers. This is followed by data samples

ViewPort will update configuration parameters and restart the QuickSample cogs when:
- frame timescale changed
- trigger changed
- every now and then

The quicksample code supports either 1 or 4 cog mode. It dynamically changes its code for each sampling run.

6.4 Fuzzy Logic Engine

Use this Object to add a fuzzy logic engine to your program.
Fuzzy Logic simplifies some control problems by making control values understandable to humans.

See the section on Fuzzy Logic Control Panel for more details on the control panel, or Fuzzy Logic Concepts for additional help on basic concepts.

Features:
-Easily implement fuzzy logic in your Parallax Propeller programs
-Supports all fuzzy logic operators: not, or, and, fuzzify, defuzzify, andrule, orrule, topclass
-Allows multiple fuzzifying maps and rulesets
-Rich user interface via Viewport: variable membership is shown on maps and rulesets. All parameters can be changed on-the-fly by clicking the number
-High performance- typical fuzzy logic calculations run in less than 1ms.

Use:
-Include this object in your program:
    Obj f:"fuzzy"
-Register with ViewPort before the vp.start command
    vp.register(f.start(@FuzzFrame))
    *Allocate ~250 longs for the FuzzFrame
-Set up the mapping functions:
    f.setMap(MapNumber,@VariableToBeMapped,Class4,Class5)
    *The classes will have midpoints at(-Class5,-Class4,0,Class4,Class5)
-Initialize the rules:
    f.setRule(RuleNumber,@Var1,@Var2,@RulePtr)
    *RulePtr should point to an array of 28 bytes

In your control loop:
-Fuzzify your variables into the 2 Fuzzy Registers:
    f.fuzzifyA(varA,MapNumber)
    f.fuzzifyB(varB,MapNumber)
-Perform your calculations:
    f.doOr
    f.doInv
    f.doandRule(RuleNumber)
-Defuzzify the result into a variable:
    varC:=f.defuzzify(MapNumber)
-Share data with ViewPort during development:
    f.share(@FuzzFrame)

6.5 PropCVFilter

Use this Object to continuously apply vision filters to video data captured by PropCVCapture. Spin commands are used to manipulate a vision program which is run in assembly in its own cog.

Features:
-Implements common vision filters in assembly for real time performance- up to 30fps
-Combine multiple filters using a "vision program"
-Supports 1, 2 or 4 video buffers
-Commands: copy, invert, threshold, difference, chaos, max, pattern

Use:
-Supply Propeller with video signal and capture with PropCVCapture
	See VideoCapture for details
-Include this object in your program:
```
	Obj ve:"PropCVFilter"
```
-Start PropCVFilter Object
```
	ve.start(@videoFrame,video#VIDEO4) 'start cog to apply vision filters
```
-Program Vision Filters
```
	ve.filter(0,0,ve#pattern)	'draw a pattern into frame 0
	ve.filter(1,0,ve#invert)	'invert pixels from 0 to 1
```
-The Vision object will continually apply the programmed filters to incoming video

6.6 PropCVCapture

Use this Object to capture video into an array which can be streamed to ViewPort for display. Other programs can analyze the data and use ViewPort to display their output.

Features:
-Capture NTSC video up to 240h x 200v at 16 grayscales into 24kb buffer at 30fps
-Supports lower resolution of 120hx100v at 16 grayscales into a 6kb buffer at 30fps
-Continuously stream video, or take one snapshot
-Realtime tracking of a bright blob without using memory at 30fps

Use:
-Supply Propeller with video signal
    To generate the signal, use a "C Cam 2A" available at http://electronics123.com
    To sample the signal, use a "ADC08100" available at digikey.
    Connect the camera's NTSC composite signal to the ADC's input and connect the ADC's output to the Propeller's pins 0..3. Drive the ADC's clock with Propeller's pin 14.
-Include this object in your program:
    Obj video:"VideoCapture"
-Allocate memory for the VideoFrame:
    LONG VideoFrame[6000] '6000 for hi-res, 1500 for lo-res
-Register with ViewPort before the vp.start command
    vp.register(video.start(@videoFrame,video#HIVIDEO))
-Analyze VideoFrame
    8 pixels are encoded into each long. Item 0 is top left.
    ViewPort will show the video image and can mark a box if supplied with a variable named visionXYWH- where x is top 8 bits, y is next 8, w is next 8, h is last 8.

The low-cost cmos camera which allows the Propeller to capture video at 240x240@256 graylevels at 30fps and stream video to the Viewport at 240x200x16 grays is the "C Cam 2A" available at: http://www.electronics123.com/

The camera's composite NTSC signal is digitized by a high-speed ADC- like the "ADC08100".

Theory:
A picture is imaged by the camera, converted to NTSC, sampled by the ADC into 8 bits and then read as a digital value by the Propeller. The Viewport TV object running on the Propeller samples the INA port, finds the syncs, and "compresses" the data into a frame which is then sent to the Viewport application.

6.7 Terminal

Use this Object instead of a FullDuplexSerial derivative to quickly get started with ViewPort. ViewPort has a terminal view which let's you use your existing terminal code.

Features:
* Full suite of terminal functions to receive and transmit strings and integers
* Fully compatible with "Parallax Serial Terminal"
* Wrapper for all Conduit functions- just use this object

Use:
-Include this object in your program- replace your existing FullDuplexSerial variant:
```
    Obj vp:"Terminal"
```
-Allocate 2 longs, these must be the first 2 longs shared by vp.share
```
    VAR long OutP,InP
```
-Configure and share memory with ViewPort
```
    vp.config(string("start:terminal::terminal:1"))
    vp.share(@outP,@inP) 'use instead of vp.start(30,31,0,9600)
```
Here's a simple program:
```
OBJ
  vp: "terminal"
var
  long OutP,InP
pub main
  vp.config(string("start:terminal::terminal:1"))
  vp.share(@OutP,@InP)
  repeat
    vp.str(string("Hello World!",13))
```

Terminal Input/Output Functions

Char(byte)	Print a byte
Chars(byte,count)	Print a byte count times
Str(stringptr)	Print a string
Dec(value)	Print a decimal number
Hex(value, digits)	Print a hexadecimal number
Bin(value, digits)	Print a binary number
DecFP(value,dp)	Print a floating point number
CharIn	Receive a byte of data
StrIn(stringptr)	Receive a string
StrInMax(stringptr,maxcount)	Receive a string limited by maxcount
DecIn	Returns decimal string as value
BinIn	Returns binary string as value
HexIn	Returns hex string value as value
RxCheck	Check if byte received (never waits)
RxFlush	Flush receiver buffer

Terminal Positioning
Clear, ClearEnd, ClearBelow
Home Position(x,y), PositionX(x), PositionY(y)
NewLine, LineFeed,MoveLeft,MoveRight,MoveUp,MoveDown
Tab, BackSpace,Beep

ViewPort Functions

Share(@varA,@varC)	Share memory from varA to varC with ViewPort. This lets you monitor&change them. You MUST use this function- at the start of your program, but AFTER "register" and "config" statements.
Register(component,cfg)	Register other components (fuzzy logic, video, quicksample..)
Config(cfgstring)	Configure ViewPort. CfgString can be generated by ViewPort.

6.8 Configuration Strings

You can include configuration strings in your Propeller Program to configure ViewPort. Doing this is optional, but gives you the power to control exactly how ViewPort will display the data from your program. The easiest way to create the strings is to configure ViewPort with the configuration wizard and instrument controls and then selecting "Copy to Clipboard" from the "Configuration" menu. You can also copy configuration code from other tutorials or type it in directly.
All configuration strings must be passed to the conduit object using the vp.config method before the vp.share method is called.
There are 2 types of configuration sections: global and graph. The global configuration sections are called "var" and "edit". These let you configure the variables used in ViewPort and the variable to be edited with controls. "Graph" sections correspond to graph widgets defined by a view's XML file. "LSA" is used by the LSA widget, "DSO" by the DSO, Spectrum Analyzer, and XY widgets. All properties have defaults, so feel free to specify only what you need.

Global Configuration Section:
"var"
Configure variable names and properties with "var" followed by a comma separated list of variable names with optional properties in parenthesis. If specified, properties are comma separated, name=value pairs and may include:

- "f": formula for calculating the display value from the Propeller's integer representation. Either "x/scale+offset" or "scale/x+offset" where "scale" and "offset" are numbers.
- "unit": Suffix for the variable's value
- "string=x":Variable point to a string of length "x" bytes. X must be a multiple of 4. This variable can not be the last variable shared.
- "base": value will be displayed in this base: valid=2,10,16
- "decode": to set up a decode channel using "source[startbit..endbit]"
- "bits": to label individual and groups of bits using "[<#><name>,<#><name>,<groupname>[<#><name>,<#><name>]]" where # is the bit's number and <name> and <groupname> start with a non-digit.
- "mode": can be set to "log" to indicate value should be graphed in log mode
- "min"/"max" to set default minimum/maximum for variable

Example from tutorial #1:
```
vp.config(string("var:io(bits=[cntr[16..19],30tx,31rx]),freq(unit=Hz,f=x/1000,min=0,max=10)"))
```
Here, ViewPort will have 2 variables labeled io and freq. IO will have it's bits labeled. Freq is measured in "Hz" and range from 0 to 10. When the Propeller sends an integer of 1000, ViewPort will display a value of 1Hz.

"edit"
Configure edit controls with "edit" followed by a comma separated list of variable names with optional properties in parenthesis. If specified, properties are comma separated, name=value pairs and may include:

- "min"/"max" to set minimum/maximum for the control
- "default": to specify the default value which is sent at connect
- "label": to set the labels shown with certain edit controls
- "mode": to specify which controls to sue for editing, defined by each view- but can include one or more of: "switch,text,dial,scroll,slider". Multiple values must be comma-separated and bracketed.

Example from tutorial #1
```
vp.config(string("edit:freq(default=5,mode=[dial,text,scroll])"))
```
Here, ViewPort will edit 1 variable- freq. When ViewPort first connects, it will set freq to 5. ViewPort will draw a dial, textbox, and scrollbar for this channel.

6.9 Configuration String Part 2

Graph Configuration Section:
"dso","lsa", or other as defined by the view's graph widget
Configure a graph widget with the graph's name: "dso", "lsa", or other followed by a comma separated list of name=value pairs which may include:

- "timescale"= time scale for graph
- "timeoffset"= offset from trigger point or current measurement
- "cursors"= sets cursor state using this format [xstate,xa,xb,ystate,ya,yb] where state can be "on" or "off", and xa, xb, ya, yb can be 0-1000 giving position.
- "ymode"= defines how graph range will be set: can be autoscale, manual or ranged
- "view"=followed by a single variable or multiple comma-separated variables in brackets. Each variable may have graph properties which must be comma separated in parenthesis:
  - "mode"=coupling mode, "ac" or "dc"
  - "offset"= offset to midpoint in graph
  - "scale"=units for a whole screen
- "trigger"=to set a trigger for the graph. There are 3 different types of triggers:
  - "SingleBit": this is the simplest trigger, used to trigger on one single bit in LSA mode. Specify the variable, bit, and edge("r" or "f") you wish to trigger on: "trigger=<variable>[<bit>]<edge>"
  - "MultiBit": this allows complete flexibility in setting a binary trigger. Specify the variable followed by a pattern string: "trigger=<variable>[<pattern>]" Each character in the string represents a bit- lsb is the rightmost. Choose from "0"- bit must be low, "1" must be high, "r" must transition from low to high, "f" must transition high to low, "x" don't care.
  - "Value": this is used to trigger on analog values. Specify the variable, operator and value: "trigger=<variable><operator><value>" where value is a number or "auto" to specify auto-trigger mode. Choose an operator from ">":rise, "<":fall, "=":equal, "{":less than, "}":greater than, "!" is different.

Example from tutorial #1:
vp.config(string("lsa:view=io,timescale=1ms,trigger=io[16]r"))
Here ViewPort will set up the lsa view to view the io channel. It sets the timescale to 1ms/div and starts a trigger on the rising edge of bit 16.

Configuration String Parsing:
:: separates groups
: separates name from value
() is used for for commas-separated lists of name=values minor variables
[] is used for lists of comma-separated values
. is used for decimal point and bits
= is used for assignment
<allconfig> ::= {<cfg> "::" }
<cfg> ::= <config name:string> ":" {<namevalue> ";"}
<namevalue> ::= <name:string> "=" {<value> "," }
<value> ::= e|<int>|<string>| <string> "(" <namevalue> ")"

Part VII

Problem Solving

7.1 Problem Solving

- **Setup.exe doesn't run:**
  Make sure you have sufficient priviledges to install a new application. Also insure that your Antivirus software is not blocking new installs.

- **Setup tells me to install the Microsoft .Net Framework:**
  This is normal. ViewPort was built on .Net Framework. It should detect if you need to install this, and will then help you download and install it from Microsoft.

- **The ViewPort Application starts but doesn't Connect.**
  Check to make sure your Propeller IDE can successfully identify your Propeller and download a program to the chip. Make sure your Propeller Application is starting the ViewPortConduit Object. Set the Port manually under Edit/Preferences. Switch to a slower baud rate.

If you have other problems try our forums: **http://forums.hannoware.com**

Part VIII

Glossary

8.1 Glossary

- DSO: digital storage oscilloscope
- LSA: logic state analyzer
- GUI: guided user interface
- Widget: A view is composed of widgets. Widgets can display data- like a graph or provide control, like a time scale dial.
- View: ViewPort provides views for different tasks. Views display data and allow control of Propeller programs.
- Conduit: The conduit is a required ViewPort object that transmits data from Propeller memory to the PC and executes PC commands on the Propeller.
- Variable Channel: is created when you share a variable. You can graph this data and edit the variable's value. Sharing more variables will decrease the effective sampling rate of the streamed data.
- Frame Channel: is created when you register the QuickSample. Doing so will allow you to sample the IO port at fast rates, but slow the sampling rate of the other streamed variables channels. You can't edit the frame channel.
- Decode Channel: is used when the value you want to display is "embedded" in another value. For example, when you read a binary counters value with pin 10-17 of the Propeller, the counter's value is embedded in bits 10-17 of the INA variable. A Decode Channel gets its values by decoding a source channel with set parameters- in our case- masking out bits 10-17 and shifting right by 10. Since it's value is computed on the fly within ViewPort, it does not affect the sampling rate and you can't edit this channel.
- Control Channel: is used when you need to change variables in Propeller memory but don't need to stream them back to the PC- ie parameters as opposed to sensor readings. The variables edited by control channels start with the one after the last shared variable. For example, if you allocate these variables:

  ```
  main |a,b,c,d
  ```

  and share just a and b with this statement

  ```
  vp.share(@a,@b)
  ```

  then the first control channel will map to variable c, and the second to variable d

Part IX

How To Buy

9.1 How To Buy

Free Evaluation
Download a 30 day free trial version of ViewPort from our website:
http://hannoware.com/viewport

Purchase License Online
Visit our website to securely purchase a license for the components you wish to use using Google Checkout or PayPal:
http://hannoware.com/viewport/register.php

Upgrade your License Online
Visit our website with your existing license to upgrade your license to use additional components.
http://hannoware.com/viewport/upgrade.php

Distribution:
Distribution of ViewPort Trial are allowed. Distribution of licenses is NOT allowed. ViewPort may not be installed on a network.

Copyrights:
ViewPort is a trademark of HannoWare.
ViewPort and all files installed by the ViewPort installer are copyrighted 2007-2010 HannoWare.
Propeller, BS2, Basic Stamp 2 are trademarks of Parallax, Inc.

KEYWORD INDEX