599 Menlo Drive, Suite 100
Rocklin, California 95765, USA
Office: (916) 624-8333
Fax: (916) 624-8003

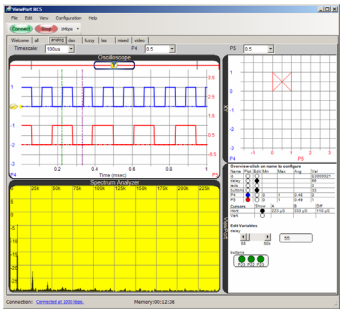
Sales:sales@parallax.com
Technical: support@parallax.com
Web Site: www.parallax.com

Propeller + PC Applications with ViewPort

PROPELLER EDUCATION KIT LAB SERIES

Draft Notice
This is a partial draft posted for feedback. Please send comments to editor@parallax.com

Introduction



ViewPort is a graphical user interface (GUI) developed by Hanno Sander for use with the Propeller Microcontroller at run time. It utilizes the Propeller chip's multiple processor design by devoting one of its processors (cogs) to provide an information conduit between the Propeller and ViewPort running on a PC. This information conduit is typically high speed serial over USB with communication at speeds up to 2 Mbps. This high speed communication, and in some cases judicious use of other cogs, makes a variety of applications possible. Here are a few examples:

- 32-bit logic analyzer displaying all I/O pin states with sampling rates of up to 80 MHz
- Multi-channel oscilloscope view displays voltage measurements, variable values, and/or I/O pin states over time
- Signal analysis - spectrum, XY plots, Bode plots
- Signal generator with dials, control switches, scroll bars and text fields to define signals generated by the Propeller chip
- Display video streams from signals sampled by the Propeller chip + A/D converter
- Control system projects such as PID tuning and fuzzy logic systems
- Simple program debugging with a variety of variable control and display options

ViewPort Example Applications and Video Demonstrations: Many of the applications listed above have preconfigured examples that are available from a list under the software's Welcome tab. You can either view the Spin code or launch the example applications from there. There are also several impressive video demonstrations available from Hanno Sander's website – www.mydancebot.com.

The system is very flexible, in many cases allowing for two or more of the features to be used at the same time. For example, the signal generator can be used with the signal analysis tools to make the PE Kit a portable electronics workbench. Since separate objects (and the cogs they launch) handle the measurement storage and communication with ViewPort, you can use this "workbench" to perform signal analysis on application circuits and code you are developing with your PE Platform.

In this lab, you will download and use ViewPort in its 30-day evaluation mode to learn some of the basics of integrating ViewPort into your applications. Topics include:

- Adding the ViewPort communication conduit and sharing Spin program variables
- Controlling signal attributes by using the GUI to determine Spin variable values
- Displaying I/O pin states in the Logic Analyzer view and taking I/O pin signal measurements with the Digital Oscilloscope view
- Controlling and monitoring I/O pins with pushbutton multi controls
- Controlling and measuring analog signal generation with D/A and A/D converters that rely on resistors and capacitors included in the PE Kit

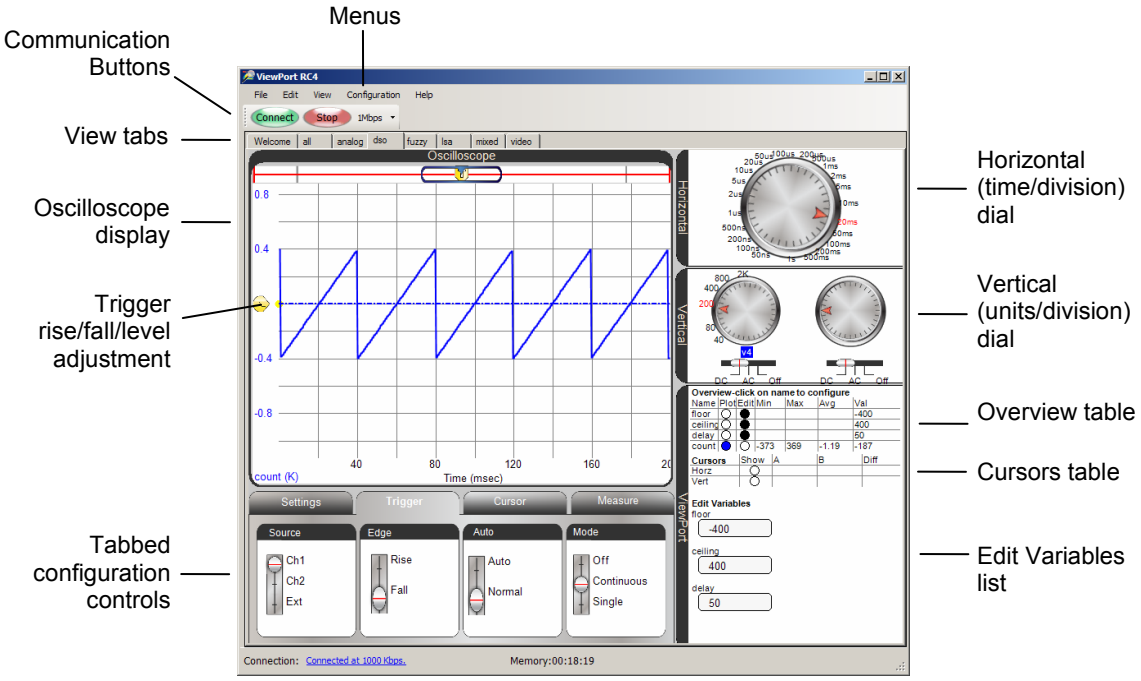
Prerequisites

Please complete the Propeller Education Kit labs before continuing here: *Setup and Testing*, *I/O and Timing*, *Methods*, and *Objects*. These labs are available from the Propeller Education Kit pages at www.parallax.com.

- ✓ Download ViewPort from www.mydancebot.com and install it on your PC.

Share, Display, and Control Variables and Configure the Views

This first example starts with a simple counting program combined with code that shares four of the program's variables with ViewPort. ViewPort can then be configured to display the variable that does the counting in its Oscilloscope view, and control the other three variables that determine the counting loop's behavior. In the figure below, the variable doing the counting (`count`) is shown in the Oscilloscope display. The Edit Variables list in the lower-right corner has fields where you can enter numbers to set the values stored by the other three program variables (`floor`, `ceiling`, and `delay`) that change the counting loop's upper and lower limits and rate.



ViewPort has View tabs that allow you to choose a variety of preconfigured views. Some examples include Logic State Analyzer (lsa) and Digital Storage Oscilloscope (dso). Each View has different features and controls. For example, the Oscilloscope view above has a number of adjustments common to hardware oscilloscopes, including dials for Horizontal and Vertical units per division. Other common hardware oscilloscope features are located in tabbed configuration panels, including Trigger, Cursor, and Measure. ViewPort also has common software features such as menus along the top, and buttons to Connect and Stop communication with the Propeller chip.

The ViewPort panel in the lower right corner gives you access to the variables your Spin program has designated as shared. It starts with a variable Overview table that displays Spin variable names and values along with buttons that allow you to plot and/or edit the variable values. The Cursors table displays summary data when the Oscilloscope Cursors are turned on for graphical measurements of plotted variable values on the Oscilloscope screen. If the Edit button next to a variable in the

Overview table is selected, the variable's name and value appears in the Edit Variables list. The default text entry fields shown in the figure allow you to control the variable values by typing in numbers. Other options for Spin variable control include configurable controls such as dials, scroll controls, and buttons.

Sharing a list of variables with ViewPort takes just three additional steps in your Spin program. Here are excerpts of these three steps from this section's Display Counting.spin example program.

1) Declare a contiguous list of long variables to share with ViewPort. They must be longs:

```
VAR

  long floor, ceiling, delay, count
```

2) Declare the ViewPort Conduit object:

```
OBJ

  vp : "Conduit"
```

3) At the beginning of your runtime code, pass the Conduit object's `share` method the addresses of the first and last variables in the list that you want to share with ViewPort:

```
PUB Counting

  vp.share(@floor, @count)
```

The Display Counting object is an example of a simple application that has code added to display and control variable values with ViewPort. Comments indicate the three lines of code for sharing the variables with ViewPort. The application code has a **`repeat`** loop that increments a `count` variable from the value stored in its `floor` variable to the value stored in its `ceiling` variable, and the rate at which the counting cycle repeats is controlled by a `delay` variable. Since this **`repeat`** loop is nested in another **`repeat`** loop with no conditions, the counting cycle repeats indefinitely.

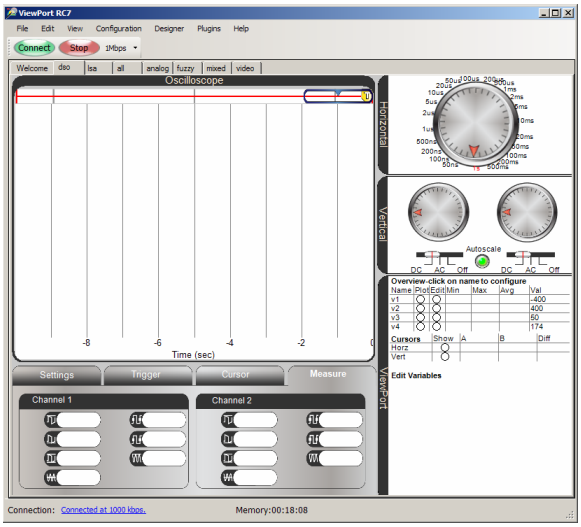
- ✓ Make sure the folder with the Display Counting object also has a copy of the ViewPort Conduit object. The Conduit object's default location is C:\Program Files\ViewPort\mycode.
- ✓ Load Display Counting.spin into the Propeller chip's EEPROM with F11.

```
''Display Counting.spin
''Display Variable States with ViewPort.

CON

  _clkmode = xtal1 + pll16x
  _xinfreq = 5_000_000

VAR

  long floor, ceiling, delay, count             ' Step 1 - Contiguous longs
  long i, us, usDelay

OBJ

  vp : "Conduit"                                ' Step 2 - Declare Conduit object

PUB Counting

  vp.share(@floor, @count)                      ' Step 3 – Share addresses with Conduit
```

```
floor := -400
ceiling := 400
delay := 50

us := clkfreq/1_000_000

repeat
  usDelay := delay * us
  repeat count from floor to ceiling
    waitcnt(usDelay + cnt)
```

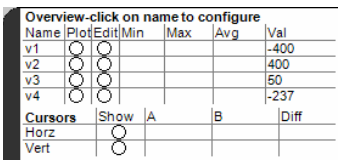
- ✓ Run ViewPort.
- ✓ Click the green Connect button. 

ViewPort opens by default to the dso (digital storage oscilloscope) View tab, and no variables are displayed on the Oscilloscope screen. The Autoscale feature between the two Vertical adjustment dials is also enabled by default so that when a variable is selected for graphing, the visible plot area will automatically adjust the vertical scale units to fit the plot.



Notice that the ViewPort Overview table lists four variables: v1, v2, v3, and v4. The `vp.share(@floor, @count)` method call makes the Conduit object share the four variables with ViewPort, in the order in which they were declared. That means v1 is the Display Counting object's `floor` variable, v2 is `ceiling`, v3 is `delay`, and v4 is `count`.

- ✓ Take a look at the values in the Val column and compare them to DisplayCounting.spin to verify the correspondence of v1..v4 to `floor..count`.

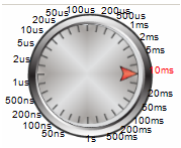


Overview-click on name to configure

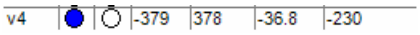
Name	Plot	Edit	Min	Max	Avg	Val
v1	○	○				-400
v2	○	○				400
v3	○	○				50
v4	○	○				-237

Cursors	Show	A	B	Diff
Horz	○			
Vert	○			

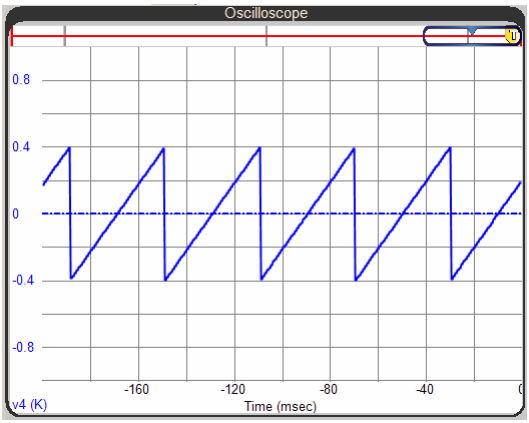
✓ Click the 20 ms label by the Horizontal dial to set the display to 20 ms per division.



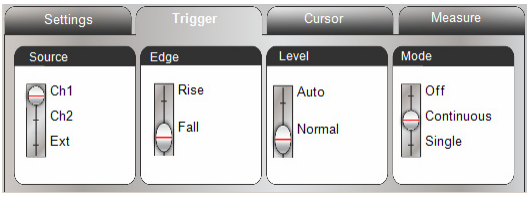
✓ Click the Plot button next to v4.



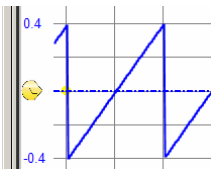
The Oscilloscope section should start to display the plotted value of the `count` variable over time, but since the trigger settings are not yet configured, the display will probably be difficult to read because it scrolls horizontally across the screen.



✓ To get the trace to hold still, click the Trigger tab, and set the Edge switch to Fall, Level to Normal, and Mode to Continuous. This should stabilize the Oscilloscope display of the `count` variable values over time.



You can set the trigger level with the yellow diamond along the left edge of the Oscilloscope display. It defines the level at which the waveform triggers a refresh of the oscilloscope display. As long as the waveform repeatedly crosses the trigger level, the display refreshes. Since we set the Trigger Level slider to Normal (instead of Auto) under the Trigger tab, if you drag the trigger level above or below the waveform the Oscilloscope display will stop refreshing.

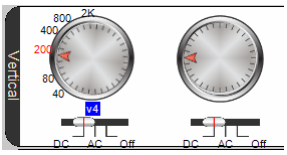


Tips: You can turn the trigger feature on by clicking the column to the left of the waveform where the amplitude values are displayed. You can also select Rise and Fall Edge options by clicking the yellow trigger level diamond repeatedly. You can also click and hold the plot to drag it up, down left and right. On a standard oscilloscope this is typically controlled by Offset and Delay controls.

✓ Use your mouse to grab the yellow diamond and drag the trigger level up and down (but not above the top or below the bottom of the waveform). It shouldn't have much of an effect on the Oscilloscope display. Try changing the Trigger Edge setting from Fall to Rise and drag the trigger level up and down again. What's changes, and why? Set the Trigger Edge back to Fall and place it at the center of the waveform before continuing.

✓ Click the Edit buttons next to v1, v2, and v3. Text fields with the values stored in the program's `floor` (v1), `ceiling` (v2) and `delay` (v3) variables should appear in the Edit Variables list. Try entering values such as 5, 20, and 100 into the v3 field and note the changes in the waveform's period. Change v3 back to 50 before continuing.

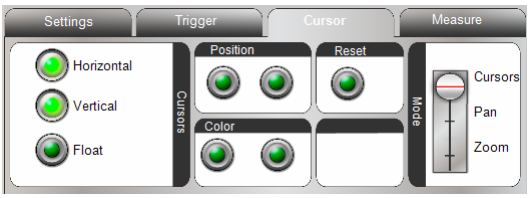
✓ Try rotating the v4 dial in the Vertical adjustment from 200 to 400 units per division and note the effect. Try a few other values, and then rotate it back to the 200 units per division setting before continuing.



✓ Try modifying the values in the v1 and v2 fields. Keep in mind that these control the `floor` and `ceiling` variables, so they will change the height of the waveform. Also, you may need to drag the trigger level up or down if the waveform no longer crosses it. Change v1 back to -400 and v2 back to 400 before continuing.

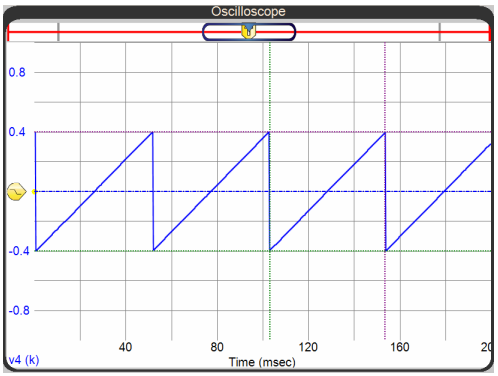
You can use cursors to measure the amplitude and period of your waveform. This is a common practice when examining electrical waveforms in an oscilloscope, but you can also use it to measure the timing of your **`repeat`** loop.

✓ Click the Cursor tab, and click the Horizontal and Vertical buttons in the Cursors control box.



Two color-coded pairs of vertical/horizontal lines (cursors) will appear in the Oscilloscope display.

- ✓ Drag the green cursors so that they intersect at the corner of the waveform at (0.1s, -0.4 k), and the purple cursors so that they intersect at the corner located at (0.15 s, 0.4 k). The ViewPort Cursors table displays the locations of the vertical and horizontal cursors and you can use it to verify that the cursors are at the correct locations.

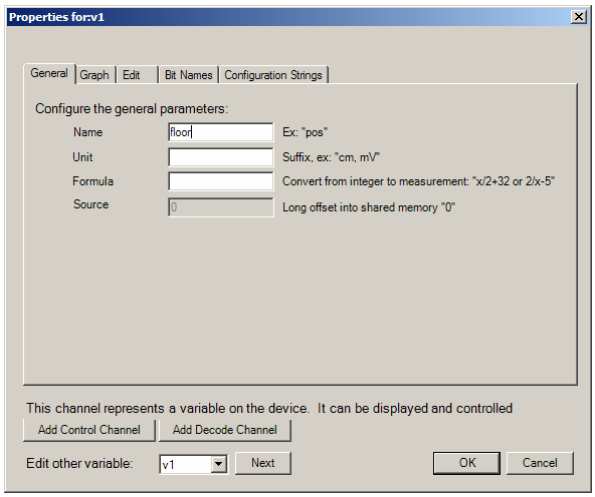


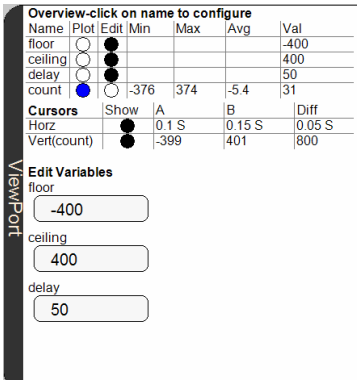
Cursors	Show	A	B	Diff
Horz	●	0.1 S	0.15 S	0.05 S
Vert(v4)	●	-400	400	800

Note that the Diff column in the Cursors table shows the difference between the two cursor locations in terms of Horiz (time) and Vert (amplitude) measurements.

You can rename v1..v4 so that they reflect the variable names in your program.

- ✓ In the ViewPort Overview table, click v1 in the Name column. The Properties window for v1 will appear. Replace v1 in the Name field with `floor`. Click the Edit other variable dropdown menu (bottom-left in Properties window) and select v2. Replace v2 with `ceiling`. Repeat for v3, replacing it with `delay` and for v4, replacing it with `count`. Then, click OK. Note that the names in the ViewPort Overview section now correspond to the variable names in the Spin program.





ViewPort has a text-based configuration language that allows your Spin program to make the Propeller configure ViewPort's view as soon as it connects. This is an important feature because it can save a lot of mouse clicks the next time you start ViewPort and connect to your Propeller chip. You can start by copying the configuration string that is generated for a given view from ViewPort and pasting it into your Spin program. It is also possible to modify the view and copy/paste the new

ViewPort string into your program. Or, when you get the hang of the configuration language you may find yourself just modifying the text in your Spin code.

- ✓ In ViewPort, click the Configuration menu and select Copy to Clipboard.
- ✓ Click the red Stop button to disconnect the serial connection with the Propeller chip.
- ✓ In the Propeller Tool software, place your cursor between the **PUB** `Counting` and `vp.share` lines, then click the Edit menu and select Paste.
- ✓ Rename the program to Display Counting with ViewPort Config.spin.
- ✓ Press F11 to load it into the Propeller chip.

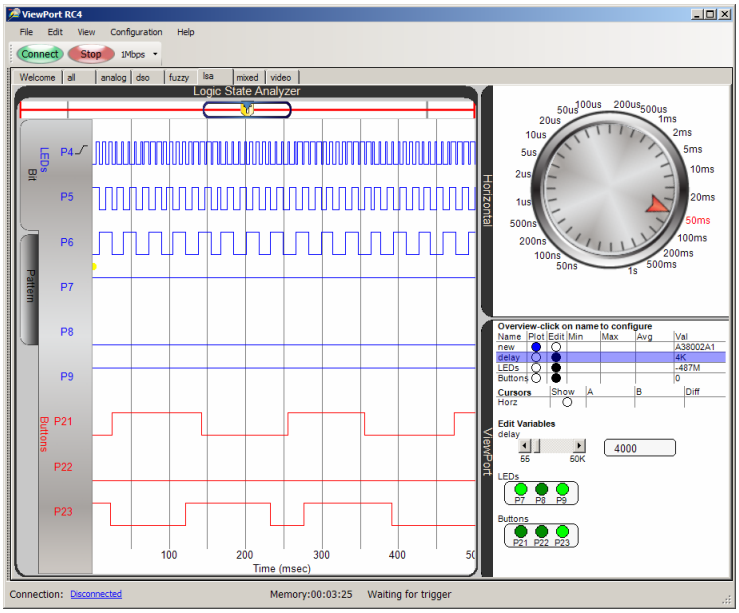
```
PUB Counting

vp.config(string("var:floor,ceiling,delay,count"))
vp.config(string("start:dso"))
vp.config(string("edit:floor(mode=text),ceiling(mode=text),delay(mode=text)"))
vp.config(string("dso:view=count(offset=-
7.295,scale=200),trigger=count<0,timescale=20ms,cursors=[on,0.513,0.768,on,0.698,0
.297],ymode=manual"))

vp.share(@floor, @count)                    ' Step 3 - Pass addresses to Conduit
' ...
```

After loading your modified program into the Propeller chip, it is safe to close and reopen ViewPort because you won't lose your settings and have to repeat all the steps to get the view back. With the settings pasted into your code and loaded into the Propeller chip, the `View` will return as soon as you click Connect. If you don't want the view settings to apply, simply comment them out of the code with braces **{ }** at the beginning and end of the block of `vp.config` method calls. Then, you'll be back to the default blank Oscilloscope screen next time you start ViewPort and click Connect.

- ✓ In ViewPort, click File and select Restart to clear the current configuration. This has the same effect as closing and restarting ViewPort.
- ✓ Click the green Connect button. This time, instead of starting with a blank Oscilloscope display, the Propeller `vp.config` calls will send the configuration strings to ViewPort and restore the display to the settings you configured.

High Speed IO Monitoring Plus Output Control

The application code in the next Spin object sends a binary counting sequence to the Propeller chip's I/O pins P4..P6. ViewPort will sample the I/O pins at 20 MHz and display I/O pin states in its Logic State Analyzer (lsa) view. It will also let you control the rate of the counting sequence, turn P7..P9 LEDs on and off, and notify you of pushbutton states.



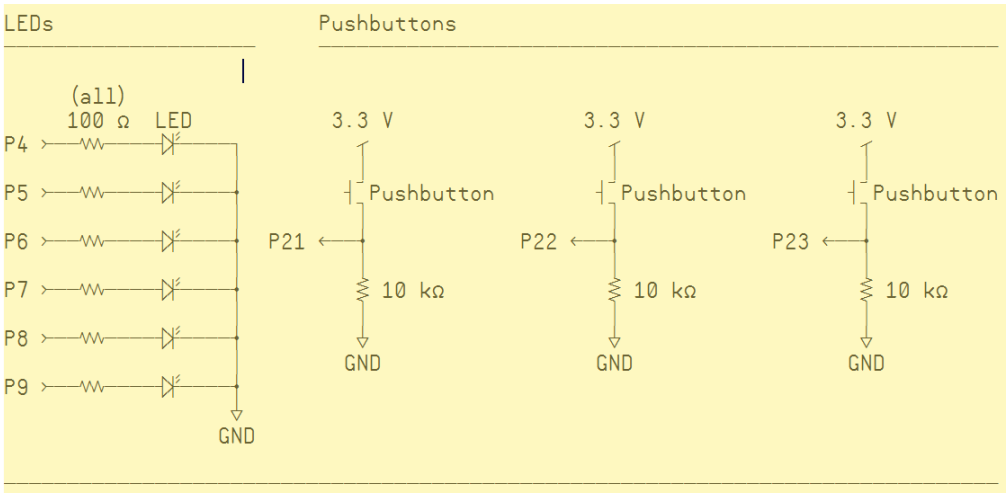
The previous example relied on continuously streaming variable values between the Propeller chip and ViewPort at 1 Mbps. By streaming 4 long variables repeatedly at 1 Mbps, the effective sampling rate is 6.25 kHz.

$$f = \frac{1\ Mbit}{second} \times \frac{1\ byte}{10\ bit} \times \frac{1\ long}{4\ byte} \times \frac{1\ sample}{4\ long} = \frac{6250\ samples}{second} \rightarrow 6250\ Hz$$

Although this sampling rate was fine for the process that was examined, the fastest sampling rate would be one variable at 25 kHz. However, ViewPort has a QuickSample object that supports sampling rates up to 80 MHz. The QuickSample object samples the states of all the Propeller chip's 32 I/O pins very rapidly and stores them in a `frame` array in the Propeller chip's main memory. The Conduit object then forwards the `frame` array values to ViewPort. The QuickSample object requires a minimum of one extra cog and a 400 long `frame` array for sampling rates up to 20 MHz. For rates up to 80 Mhz, it requires four extra cogs and a 1600 long `frame` array.

In the previous example, all Conduit had to transmit was a list of four contiguous long variables. In this example, the Conduit object will instead take turns between transmitting two contiguous long variables and the 400 (sampling rate = 20 MHz) long variables that store I/O pin states. The tradeoff is that you get a much faster I/O pin sampling rate, but it slows down the Spin program variable data to some extent. It isn't an issue in this example, but with this in mind, it's best to choose program variables that do not require rapid display updates when using QuickSample. In this example's Display and Control Selected IO.spin object, one variable (`delay`) will control the rate of the application's binary counting sequence, and the other (`leds`) will control a few low-speed on/off signals.

- ✓ If it is not already built on your PE Platform, build and test the schematic shown below.



Setting up an application to use the QuicksSample object involves adding an additional line of code to each of the three steps we followed for setting up Conduit.

1) Declare a `frame` array for QuickSample, AND declare contiguous longs (`delay` and `leds`) for the Conduit object. (Choose the number of elements in the `frame` array by the desired maximum sampling rate: 20 MHz → 400 longs or 80 MHz → 1600 longs.)

```
VAR

  long frame[400]
  long delay, leds
```

2) Declare the QuickSample object, AND then, declare the Conduit object.

```
OBJ

  qs : "QuickSample"
  vp : "Conduit"
```

3) Add a line of code that tells QuickSample the address of the `frame` array and how many cogs to use, AND then, pass the addresses of the first and last longs in your variable list to Conduit. (Choose the number of cogs to use by the desired maximum sampling rate: 1 cog → 20 MHz or 4 cogs → 80 MHz)

```
PUB LedControl

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

Let's take a moment for a closer look at the QuickSample line of code in step 3: `vp.register(qs.sampleINA(@frame,1))`. This is really two steps. First, the QuickSample object's `sampleINA` method gets passed the address of the `frame` array and the number of cogs to use. Second, the `sampleINA` method returns an address that has to be registered with the Conduit object (using its `register` method) so that the Conduit object can fetch and transmit the values QuickSample stores in the `frame` array.

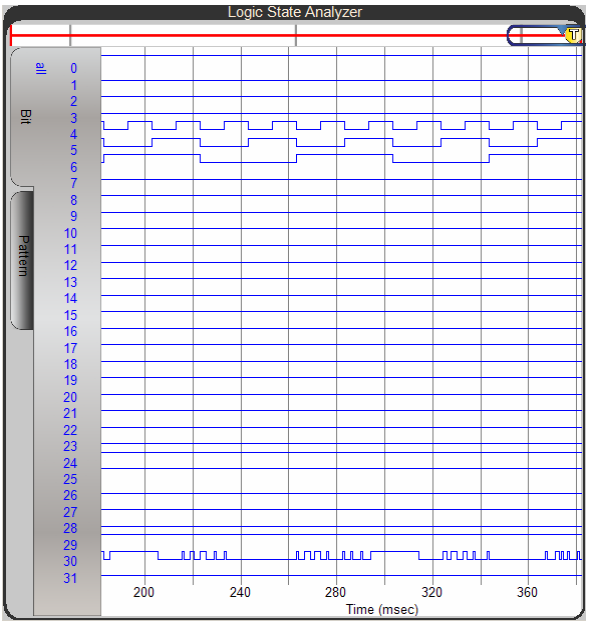
- ✓ Click the File menu and select Restart to clear any settings.
- ✓ Click the ViewPort software's Stop button to make sure it is disconnected from the COM port.

- ✓ Make sure the folder with the Display and Control Selected IO.spin object also has a copy of the ViewPort QuickSample object. The QuickSample object's default location is C:\Program Files\ViewPort\mycode.
- ✓ In the Propeller Tool software, load the Display and Control Selected IO.spin object into the Propeller chip with F11.

```
''Display and Control Selected IO.spin
''Display states of:
''  o I/O pins functioning as outputs under program control
''  o I/O pins functioning as inputs receiving signals from circuits
''  o I/O pins functioning as outputs under ViewPort control

CON

  _clkmode = xtal1 + pll16x                     ' System clock → 80 MHz
  _xinfreq = 5_000_000

VAR

  long frame[400]                               ' 400 longs for QuickSample
  long delay, leds                              ' Congtiguous longs for Conduit
  long us, i, T, mask

OBJ

  qs : "QuickSample"                            ' Declare
  vp : "Conduit"                                ' Declare ViewPort Conduit

PUB LedControl

  vp.register(qs.sampleINA(@frame,1))           'sample INA into <frame> array
  vp.share(@delay, @leds)

  dira[4..9]~~
  us := clkfreq/1_000_000
  delay := 10_000

  T := cnt
  mask := %111<<7

  repeat
    waitcnt(T+=(delay*us))
    outa[6..4]++
    outa[9..7]:=(leds&mask)>>7
```

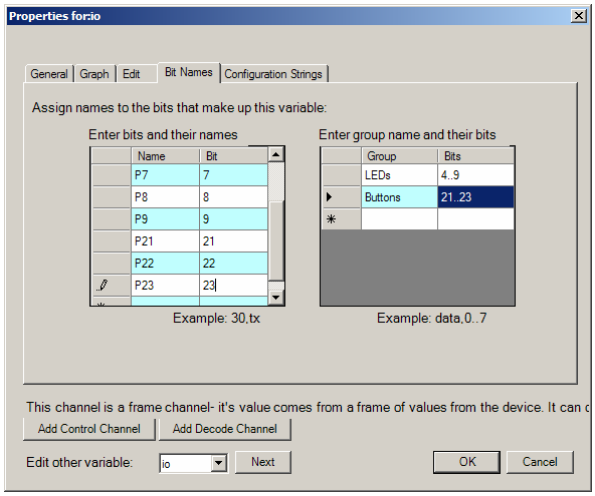
- ✓ In ViewPort, click the Connect button.
- ✓ Click the Isa tab to switch to the Logic State Analyzer view.
- ✓ Set the Horizontal scale to 20 ms.
- ✓ Click the Plot button next to the io variable name in the ViewPort Overview section.

The default Logic State Analyzer view shows all 32 I/O pin states. Active signals are currently on P4, P5 and P6 (binary counting), and on P30. The activity on P30 is Conduit sending serial data to ViewPort.

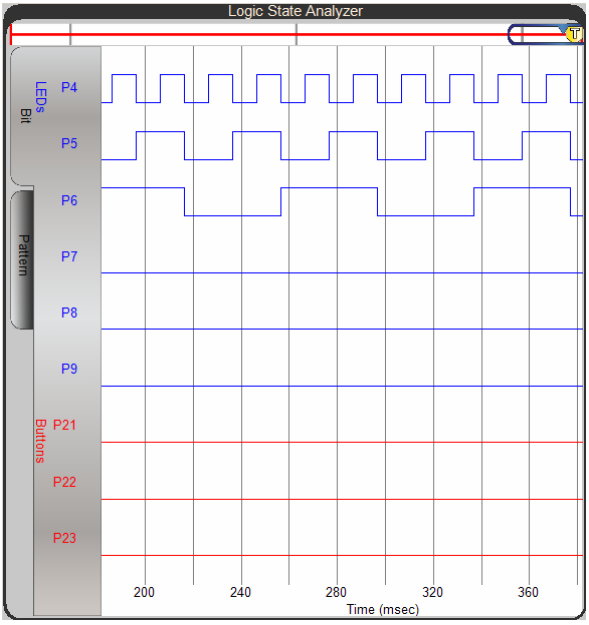


ViewPort can be configured to display any combination or permutation of Propeller I/O pins in its Logic State Analyzer display. Recall from the circuit schematic that the I/O pins of interest are the LEDs on P4..P9, and the pushbuttons on P21..23.

- ✓ In the ViewPort Overview table, click the io variable name. In the Properties window that opens, click the Bit Names tab.
- ✓ In the Enter group names and their bits table, replace the group name all with the group name LEDs, and replace 0..31 with 4..9. In the row below, add a group named Buttons with bits 21..23.
- ✓ In the Enter bits and their names table, enter P4 for the first Name, and enter 4 in the Bit cell to its right. On the next row, enter P5 for the second name, and enter 5 in the Bit cell to its right. Keep adding new rows until you get to P9 for Name and 9 for Bit. Then, add similar names and bits for P21, P22, and P23 as shown in the figure below. Then, click OK.



After clicking OK, your Logic State Analyzer should show the only the bits you have selected. Note on the left that the pin groups are color coded under the LEDs and Buttons categories you created in the Enter Group name… table.



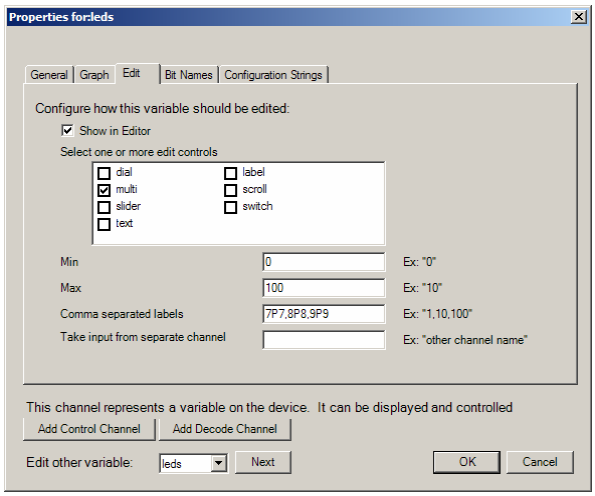
- ✓ Click P4 to the left of the Logic State Analyzer display, to make it trigger on P4's rising edge. Try clicking it three more times to cycle through the other options (falling edge, no trigger, and back to rising edge).

ViewPort's Horizontal dial is currently set to 20 ms/division. Since there are 10 divisions across the Logic State Analyzer screen, it takes 200 ms for QuickSample to take all the samples to fill the screen. So, if you were to press the button, you might notice a delay of up to one fifth of a second. For a much quicker response, you can select a smaller Horizontal time/division setting, like 200 µs.

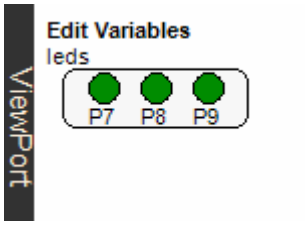
- ✓ Set Horizontal dial to 200 µs/divisions.
- ✓ Try pressing and releasing the pushbuttons connected to P21, P22, and P23, and verify that their states change in the Logic State Analyzer.
- ✓ Now, try setting the Horizontal dial to 20 ms/division.
- ✓ Press the pushbuttons on your PE Platform again. There should now be a noticeable delay between when you press the pushbutton and when the state changes on the Logic State Analyzer screen.

Next, let's set up ViewPort to use its virtual buttons control the LEDs connected to the Propeller chip's P7..P9 I/O pins. Remember from the Spin program that `leds` is the second of the two variables that `vp.share(@delay, @leds)` shared with the Conduit object. So, the `leds` variable is going to be v2 in the Overview variable table.

- ✓ Click v2 in the ViewPort Overview table. In the General tab, change its name to leds.
- ✓ Click the Edit tab, and check the Show in Editor textbox. Uncheck text, and check multi. In the Comma separated labels field, you have to enter a string that begins with the Propeller I/O pin number and ends with the label that ViewPort displays. Enter 7P7,8P8,9P9, and then click OK.



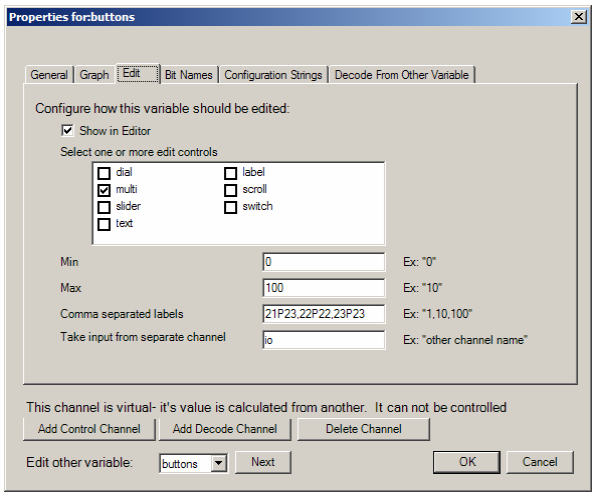
You just configured a multi-bit control with buttons you can click to set/clear bits 7, 8, and 9 in the `leds` variable. The Spin program copies bits 7..9 in the `leds` variable to **`outa[`**`7..9`**`]`**, which causes the lights on the PE Platform to turn on. When the I/O pins change state to control the LEDs, it should also show in the Logic State Analyzer.



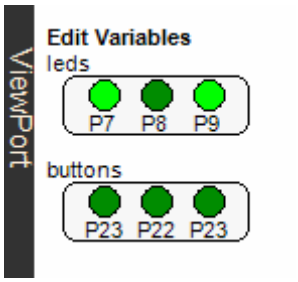
- ✓ Verify that the P7, P8, and P9 multi controls can be clicked to toggle the LEDs connected to P7, P8, and P9 on your PE Platform. The first click on a given multi control should turn the LED on, the second click should turn it back off, and so on. Also verify that the I/O pin output states are reflected in the Logic State Analyzer.

A multi control can also be used to monitor input in the way some control panels do, with a light turning on when a sensor's measurement indicates a problem. In this example, the multi control indicates the states of the pushbuttons. The steps for setting up the multi controls to monitor the pushbuttons are similar to the steps for the LEDs, with one exception. To make the multi control monitor and display the states of the pushbuttons, the multi control has to take its input from the io channel. ViewPort has a Take input from separate channel feature that can be used for this purpose.

- ✓ In the ViewPort Overview, click leds in the Name column, then click the Add Decode Channel button. In the General tab, change the name from new to buttons. In the Edit tab, check Show in Editor, uncheck text, and check multi. In the Comma separated labels field, add 21P21,22P22,23P23. ***Enter io into the Take input from separate channel field.*** ← This last step is important! Then, click OK.



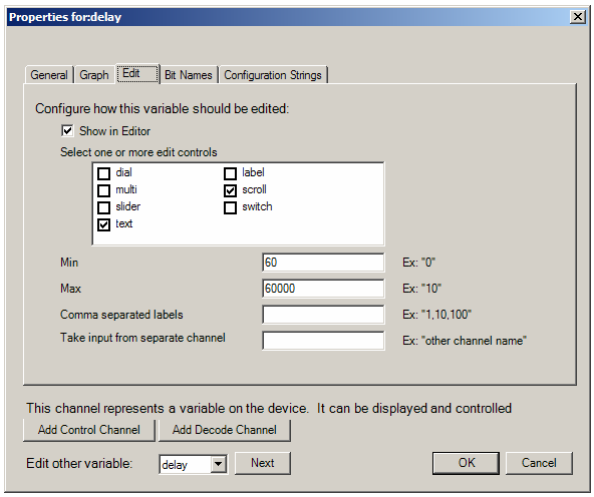
After clicking OK, there should be three more multi controls in the Edit Variables list that display pushbutton (buttons) states.



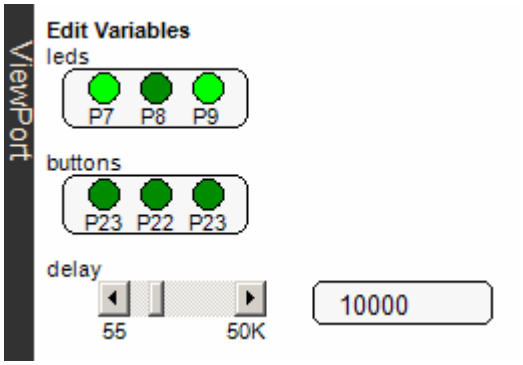
- ✓ Set the Horizontal dial to 200 µs/division.
- ✓ Press each pushbutton on your PE Platform and verify that the buttons multi control indicates the button was pressed. The I/O pin trace in the Logic State Analyzer graph should also update.
- ✓ Adjust the Horizontal dial back to 20 ms/division before continuing.

A graphical control can be added to control the Spin program's `delay` variable, which in turn controls the rate at which the binary counting pattern on P4..P6 repeats. The changes to the rate of the binary counting pattern will be observable on the PE Platform's LEDs as well as on the Logic State Analyzer display.

- ✓ In the ViewPort Overview table, click the v1 name. In the Properties for v1 window, change the name to delay. Click the Edit tab, check the Show in Editor checkbox, and check the scroll checkbox. Set the Min field to 60, and the Max field to 60000. Then, click OK.



Now, in the ViewPort Edit Variables list, there's a delay scroll bar that you can use to change the frequency of the counting pattern.



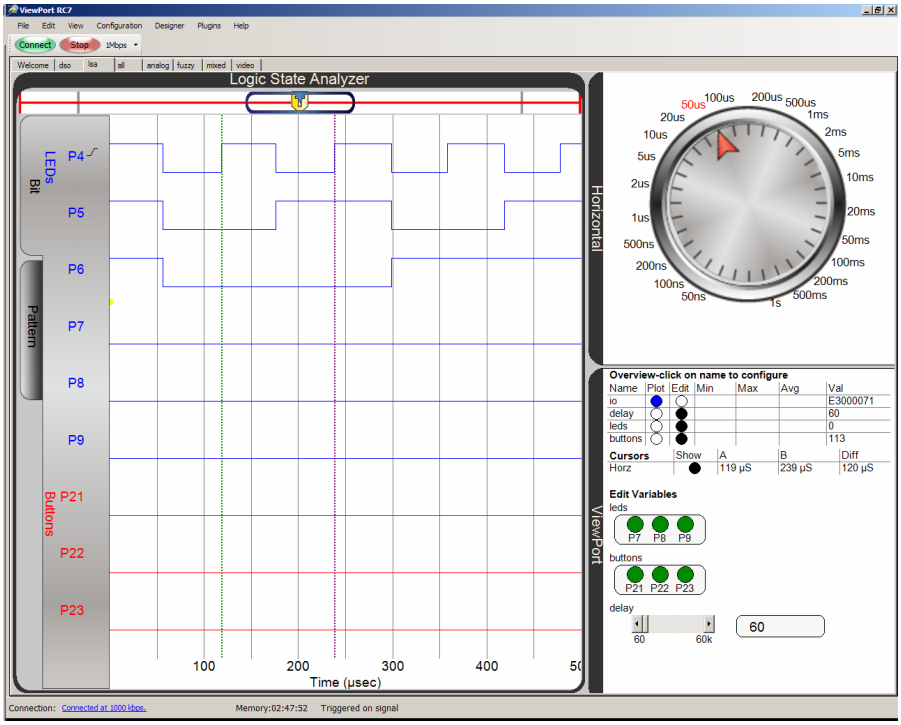
- ✓ Try sliding the delay scroll bar all the way to the left.

The LED count pattern will repeat so quickly that the LEDs on the PE Platform will appear to glow dimly instead of flicker. The Logic State Analyzer should have clear display of the counting pattern timing at the 200 µs/division Horizontal setting.

- ✓ Change the Horizontal time/division dial from 20 ms/division to 50 µs/division.

You can also use cursors to measure the signal cycle time of the signal sent by one of the I/O pins.

- ✓ Click the Show button next to Horiz in the Cursors table.
- ✓ Line the cursors up with successive rising edges on P4 signal. What's the cycle time? (Check the Diff column in the Cursors table.) Does that agree with what the cycle time should be if the `delay` variable in Display and Control Selected IO.spin stores the value 60?



✓ Try sliding the scroll bar all the way to the right.

The LEDs on the PE Platform will display a slower count pattern, easily discernible to the eye. Now, 50 µs/division is much too fast of a sampling rate to allow full cycles of the count pattern to display on the Logic State Analyzer.

✓ Adjust the Horizontal dial from 50 µs/division to 50 ms/division, and the Logic State Analyzer display should again show the entire counting sequence.

This display is almost ready to save for future use.

✓ Adjust the Horizontal dial to 20 ms/division and enter 10000 into the text entry field next to the delay scroll control. This will make a stable and clear display for the speed the Propeller chip transmits when Display and Control Selected IO.spin is booted.

In the previous activity, the approach of copying the configuration information directly into the Spin program was introduced. It involved clicking Configuration and Copy to Clipboard, and then pasting the `vp.config` method calls that were generated directly into the Spin code. You could do that here too and just make sure that the two lines of code immediately following all the `vp.config` calls are `vp.register…` and then `vp.share…`

Sometimes, it is desirable to keep different configurations for the same Spin program, so a view configuration can also be saved as a .cfg file. In order to keep it easily accessible, it's best to store it in the same folder and with a name that is similar to the Spin application object.

✓ In ViewPort, click the Configuration menu and select Save.
✓ Navigate to the folder where your Display and Control Selected IO.spin object is stored and save the ViewPort configuration as Display and Control Selected IO. It will be saved as a .cfg vile instead of a .spin file.

Next time you open and load the .spin program, you can click the Configure menu, select Open, and navigate to the .cfg file you saved to reload the configuration.

- ✓ Click the File menu and select Restart to clear the current configuration.
- ✓ Make sure the version of Display and Control Selected IO does not have any configuration strings in it.
- ✓ Load it into the Propeller Chip.
- ✓ In ViewPort, click the Connect button.
- ✓ Click Configuration, select Open, and double-click your Display and Control Selected IO.cfg file.
- ✓ Verify that it properly restores the display.

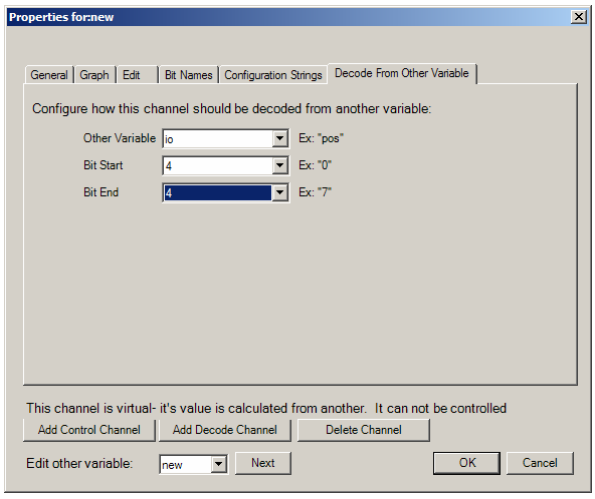
Decode Channels and Other Views

This example will guide you through viewing one of the digital signals from Monitor and Control Selected IO.spin in ViewPort's Spectrum Analyzer. This can be useful for evaluating possible radio interference that a digital signal can broadcast.

- ✓ Start with the View you saved in the previous Activity. If it is not currently set up, load Display LED Upcount.spin into the Propeller chip's EEPROM with F11. Then, click ViewPort's Connect button, and use Configuration → Open to load the Display and Control Selected IO.cfg configuration file. If you did not save this file, repeat the instructions in the High Speed IO Monitoring Plus Output Control activity, which starts on page 8.

The Spectrum Analyzer is part of the Analog view tab. To view I/O pin states in the Oscilloscope and Spectrum Analyzer, you can add a Decode Channel and configure it to display individual io variable bits.

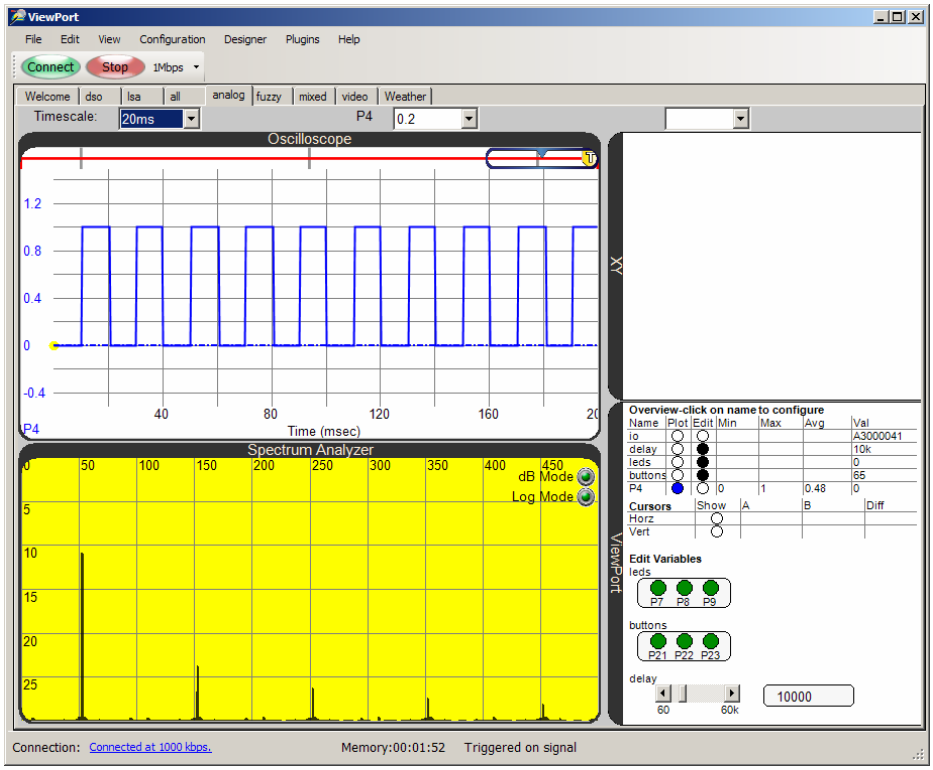
- ✓ Click the Analog tab to switch to the Analog signal view.
- ✓ Click the io Name in the ViewPort Overview table. In Properties window that appears, click the Add Decode Channel button. In the Name field, replace new with P4. Click the Decode From Other Variable tab. Set both the Bit Start and Bit End variables to 4. Then, click OK.



- ✓ Set the Timescale dropdown menu in the upper-left corner of the view to 20 ms.
- ✓ In the ViewPort Overview table, click the Plot buttons next to the P4 variable to start displaying it in the plot.
- ✓ Click 0.4 to the left of the Oscilloscope plot to set a rising edge trigger for the square wave.

- ✓ In the middle of the Spectrum Analyzer plot, click, hold, and drag to the left to adjust the frequency scale. Adjust the frequency scale so that it displays from 0 to 450 along the top of the Spectrum Analyzer display.

The Spectrum Analyzer screen displays the results of a Fast Fourier Transform (FFT) applied to the signal shown in the Oscilloscope. The FFT decomposes the Oscilloscope signal into a sum of sine waves, and plots the amplitude (vertical axis) against the frequency (horizontal axis) for each sine wave that makes up the signal. The spikes in the spectrum analyzer indicate the frequencies of the sine waves that, when added together, compose the square wave.



The Spectrum Analyzer shows spikes at 50, 150, 250, 350, and so on. The spike at 50 Hz signifies the fundamental sine wave that makes up the square wave. The spike at 150 Hz signifies a sine wave called the 3rd harmonic, which is three times the frequency of the fundamental; the 250 Hz sine wave signifies a sine wave called the 5th harmonic, which is five times the fundamental frequency, and so on. As the harmonics increase, their amplitudes decrease. Each successive harmonic sine wave that gets added to the fundamental causes the resulting waveform to more closely resemble the waveform in the Oscilloscope view.

In some cases, as these signals travel through wires, they can broadcast like an antenna, causing radio interference. If it's a digital signal like the square wave you see, it could potentially broadcast interference at 50 Hz, 150 Hz, 250 Hz, and so on. While a square wave's harmonics might be simple to predict since the Fourier series for a square wave is a common math exercise, other forms of digital communication can be more difficult to predict. For these waveforms, the Spectrum Analyzer can be an exceedingly useful tool.

-- End of Partial Draft --

To-Do List

Still to come in this lab:

- Function Generator and Oscilloscope application
- Questions, Exercises, Projects, and Solutions