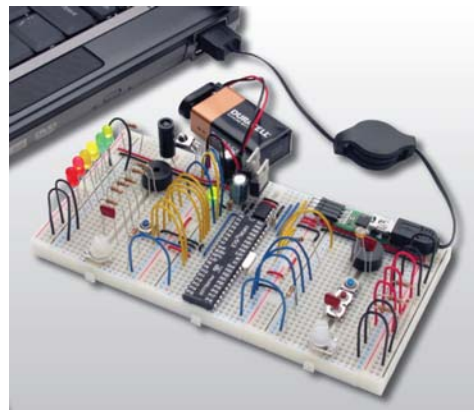


Spin Tips



I/O & Timing Simplified with the Propeller Education Kit



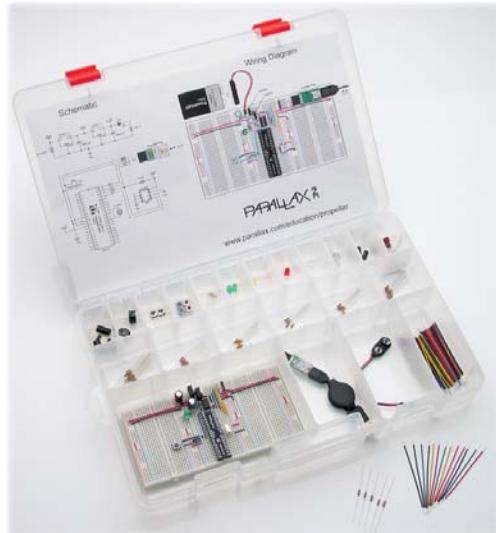
My name is Andy Lindsay, and I develop educational kits and books for Parallax. My first Propeller Education assignment was a kit and book for university students with some prior electronics and microcontroller experience. Now, I'm working on a Propeller Education Kit that's good for beginners. It'll be kind of like a What's a Microcontroller and the Board of Education, but all for the Propeller microcontroller.

This Spin Tip introduces an object that simplifies I/O access and another that simplifies event timing. For background and comparison, we'll first look at the PE Kit documentation's university level introduction to I/O, Timing, and synchronized delays. After that, we'll compare the PE Kit coding approach to the approach I'm using for the new educational kit. It utilizes objects with methods that make coding I/O, timing, and other tasks simple and intuitive.

Note: Synchronized delays are important for applications that require a precisely timed loop.

Propeller Education Kit (PE kit)

40-Pin DIP Version

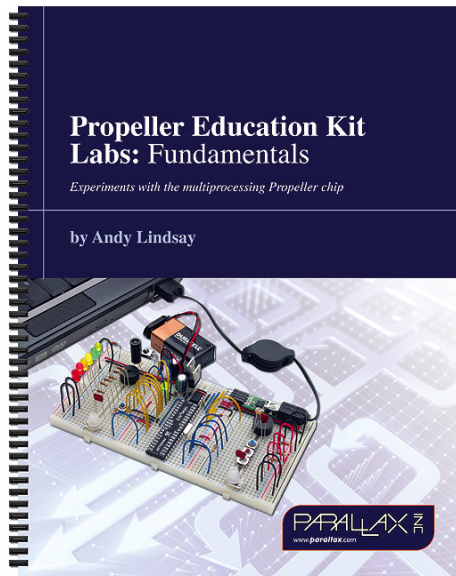


Propeller Education Kit – 40 Pin DIP



This is the Propeller Education (PE) Kit. It's a starter kit designed for university students that provides a flexible Propeller microcontroller prototyping platform. It's especially useful for building student projects on a breadboard and testing before committing the design to a soldered version on the Propeller Proto Board or to a printed circuit board design.

PE Kit Labs: Fundamentals Book



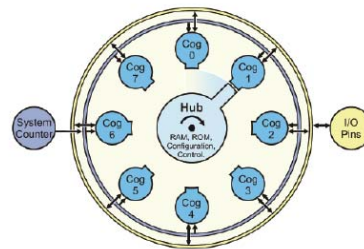
- For those new to the Propeller
- Guided tutorial
- Accompanies both versions of the PE Kit
- Prerequisites
 - At least What's a Microcontroller



This is the cover of Propeller Education Kit Labs: Fundamentals lab manual. It accompanies the PE Kit and provides a university level introduction to multiprocessing with an emphasis on understanding how the Spin programming language works with the Propeller microcontroller's architecture. Hobbyists and younger students have reported that the learning curve is pretty steep. On the other hand, university students typically report success with this primer. Case in point, the CSUS Wireless Lighting Control senior project group that presented at the start of this meeting. Great job CSUS Wireless Lighting, and go Hornets!

This book is available for free PDF download from www.parallax.com/go/PEKit. Make sure to download v1.2; it's the latest.

What is the Propeller Microcontroller?



Hub and Cog Interaction

- 8 parallel 32-bit processors (cogs)
- 32 I/O pins
- 32 KB of shared (main) RAM
- Each cog has 2 KB of RAM
- Clock speeds up to 80 MHz
- DIP, QFP and QFN packages



From PE Kit Labs, page 7

These are excerpts from the PE Kit Labs book. The upper-right illustration is from the Propeller Block Diagram, and the PE Kit Labs book uses it to explain how each cog takes turns accessing Hub resources. This is an example of the book's emphasis on programming + architecture.

Propeller Education Kit Labs: Fundamentals

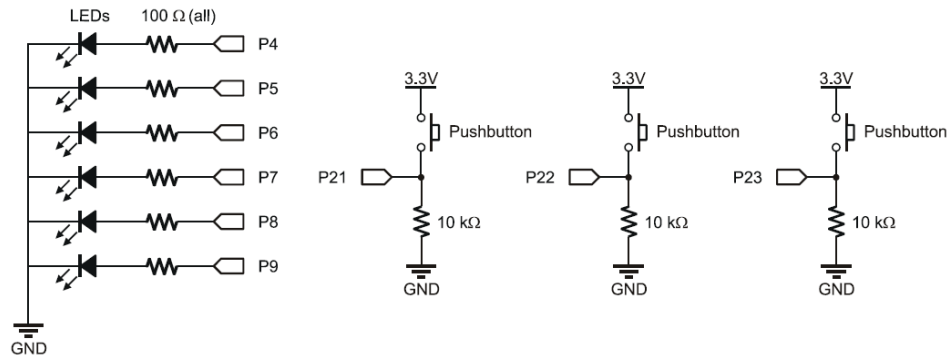
Chapter 4 I/O and Timing



I/O monitoring, control, and event timing are introduced in chapter 4 of the PE Kit Labs book. In this chapter, many Spin programming elements and their relationships to the Propeller chip's architecture are explained. Circuit + coding activities emphasize each point.

Lab 4: I/O and Timing

Circuit for this lab



From PE Kit Labs, page 46

This is the test circuit for Chapter 4 in the PE Kit Labs book.

IMPORTANT: The circuit for this Spin Tip is just an LED circuit connected to P8.

Lab 4: I/O and Timing

Indenting code blocks

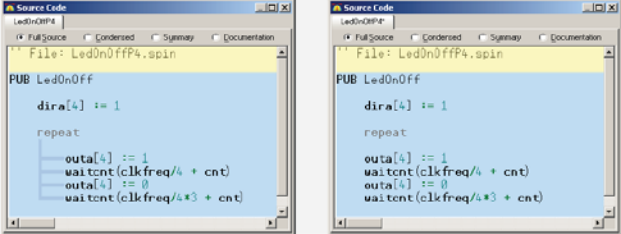
Remember that indentation is important! Figure 4-2 shows a common mistake that can cause unexpected results. On the left, all four lines below the repeat command are indented further than repeat. This means they are nested in the repeat command, and all four commands will be repeated. On the right, the lines below repeat are not indented. They are at the same level as the repeat command. In that case, the program never gets to them because the repeat loop does nothing over and over again instead!

Notice the faint lines that connect the "r" in repeat to the commands below it. These lines indicate the commands in the block that repeat operates on.

To enable this feature in the Propeller Tool software, click *Edit* and select *Preferences*. Under the *Appearance* tab, click the checkmark box next to *Show Block Group Indicators*. Or, use the shortcut key **Ctrl+I**.


Figure 4-2: Repeat Code Block
This repeat loop repeats four commands

The commands below repeat are not indented further, so they are not part of the repeat loop.



```
File: LedOnOffP4.spin
PUB LedOnOff
  dira[4] := 1
  repeat
    outa[4] := 1
    waitcnt(clkfreq/4 + cnt)
    outa[4] := 0
    waitcnt(clkfreq/4*3 + cnt)
```

```
File: LedOnOffP4.spin
PUB LedOnOff
  dira[4] := 1
  repeat
  outa[4] := 1
  waitcnt(clkfreq/4 + cnt)
  outa[4] := 0
  waitcnt(clkfreq/4*3 + cnt)
```

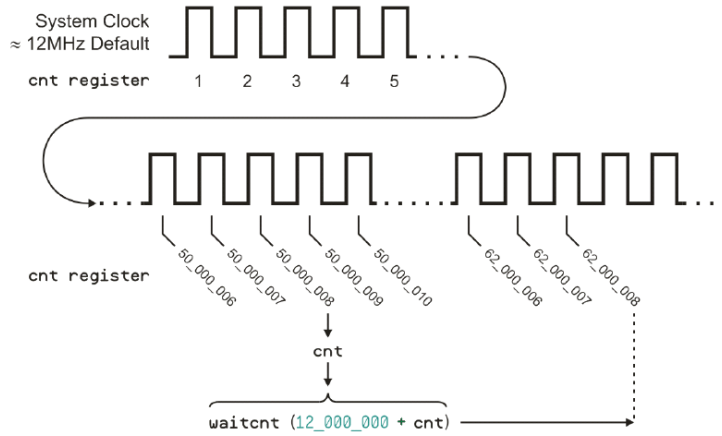
PARALLAX 
www.parallax.com

This is an excerpt from the point in the chapter where code indentation is introduced. It might not have been necessary for this presentation, but if you are reading these notes and are new to the Propeller, take a moment to read the slide....or, download the book PDF from www.parallax.com/go/PEKit, and read it from the source, page 52.

IMPORTANT: Make sure to download [Propeller Education Kit Labs: Fundamentals Book v1.2](http://www.parallax.com/go/PEKit) (.pdf)

Lab 4: I/O and Timing

The WAITCNT Command



From PE Kit Labs, page 53

This is another example of the architecture + programming language explanations in the PE Kit Labs book. It illustrates how the Propeller chip's cnt register increments with every system clock tick. It also shows how a target value for the cnt register can be established in the waitcnt command. The waitcnt command waits for the target value, which is the current value of the cnt register, plus some additional number of clock ticks. The waitcnt command waits for that target value before allowing a Spin program to move on to the next command.

Lab 4: I/O and Timing

Blinking LED

```
'' File: ConstantBlinkRate.spin
CON
    _xinfreq = 5_000_000
    _clkmode = xtal1 + pll16x
PUB LedOnOff
    dira[4] := 1
    repeat
        outa[4] := 1
        waitcnt(clkfreq/2 + cnt)
        outa[4] := 0
        waitcnt(clkfreq/2 + cnt)
```



From PE Kit Labs, page 54

After introducing the cnt register and waitcnt command, clkfreq and the clock settings in the CON block are introduced for more precise timing of an LED on/off loop. Clkfreq is the number of clock ticks in one second, and the system clock settings in the CON block are more examples of how the native spin language is very close to the Propeller chip's architecture.

Lab 4: I/O and Timing

Synchronized Delays - This is important!

''File: TimekeepingBad.spin

```
CON
    _xinfreq = 5_000_000
    _clkmode = xtal1 + pll1x

VAR
    long seconds

PUB BadTimeCount
    dira[4]~~

    repeat
        waitcnt(clkfreq + cnt)
        seconds ++
        ! outa[4]
```

''File: TimekeepingGood.spin

```
CON
    _xinfreq = 5_000_000
    _clkmode = xtal1 + pll1x

VAR
    long seconds, dT, T

PUB GoodTimeCount
    dira[9..4]~~

    dT := clkfreq
    T := cnt

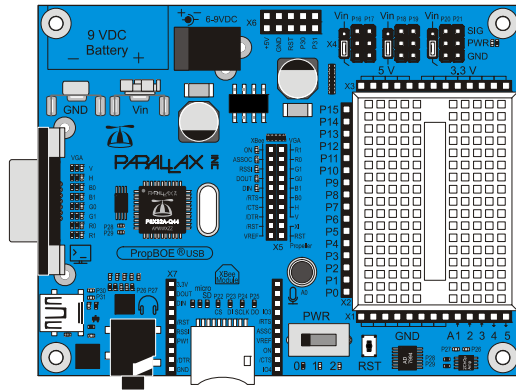
    repeat
        T += dT
        waitcnt(T)
        seconds ++
        outa[9..4] := seconds
```

From PE Kit Labs, page 64

Near the end of the chapter, synchronized delays are introduced. Synchronized delays make it possible to add commands to a loop without delaying the timing of certain events. Synchronized delays accomplish this by marking the cnt register's value at some point in time and then supplying the waitcnt command with a calculated value, which is typically the marked time plus a certain number of clock ticks. TimekeepingGood.spin uses $T += dT$, and it provides a target value that moves ahead by fixed increments. Code before the waitcnt command in TimekeepingGood.spin can take variable amounts of time to execute, but the code immediately after the waitcnt command occurs at a fixed interval.

Side note: The code between the marked time and the waitcnt command should not be allowed to take too long. The term "too long" applies if the code takes so long that the value of the cnt register exceeds the value the waitcnt command is waiting for. In the example on the right (above), if the code exceeds 1 second's worth of clock ticks, it would qualify as taking too long. If this happens, the waitcnt command has to wait for an additional 2^{32} clock ticks before it gets back to the target value. When running at 80 MHz this translates to a 53.7 second delay, during which time, it seems like the cog stops responding. Keep this "gotcha" in mind when working with synchronized delays. If the cog appears to stop responding, it could be because your extra code exceeded the target value that you supplied to the waitcnt command.

For Beginners & Lower Grade Levels



In the Works:

- Propeller Board of Education
- C language
- Objects that do not require in-depth understanding of the Propeller Chip's architecture

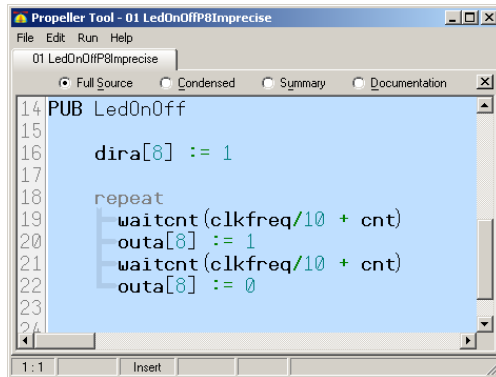


Parallax Propeller Education resources will soon be available to augment high school and middle school STEM courses. STEM stands for Science, Technology, Engineering and Mathematics. These resources are part of a program that will utilize a Propeller Board of Education, a simple C language variant that can seamlessly incorporate Spin building block objects, and a suite of objects that simplify common coding tasks.

The programs in “Spin Tips 2011.01.27.zip” provide examples of the kind of abstractions that will be utilized to simplify the coding. Please keep in mind that the building block objects we are looking at today (Input Output Pins.spin, Timer.Spinn and Simple Motor.spin) are all works in progress. The reason they are included in this presentation is because they dovetailed so nicely with the request for a generic input/output object that simplifies I/O access for the Spin Tips segment of this meeting. (The January 27th, 2011 Propeller Users Group meeting.)

There's a discussion about this board on the Parallax General Discussion forum. See: <http://forums.parallax.com/showthread.php?127226-New-Prop-BOE>

01 LedOnOffP8Imprecise.spin



```
14 PUB LedOnOff
15
16   dira[8] := 1
17
18   repeat
19     waitcnt(clkfreq/10 + cnt)
20     outa[8] := 1
21     waitcnt(clkfreq/10 + cnt)
22     outa[8] := 0
23
24
```

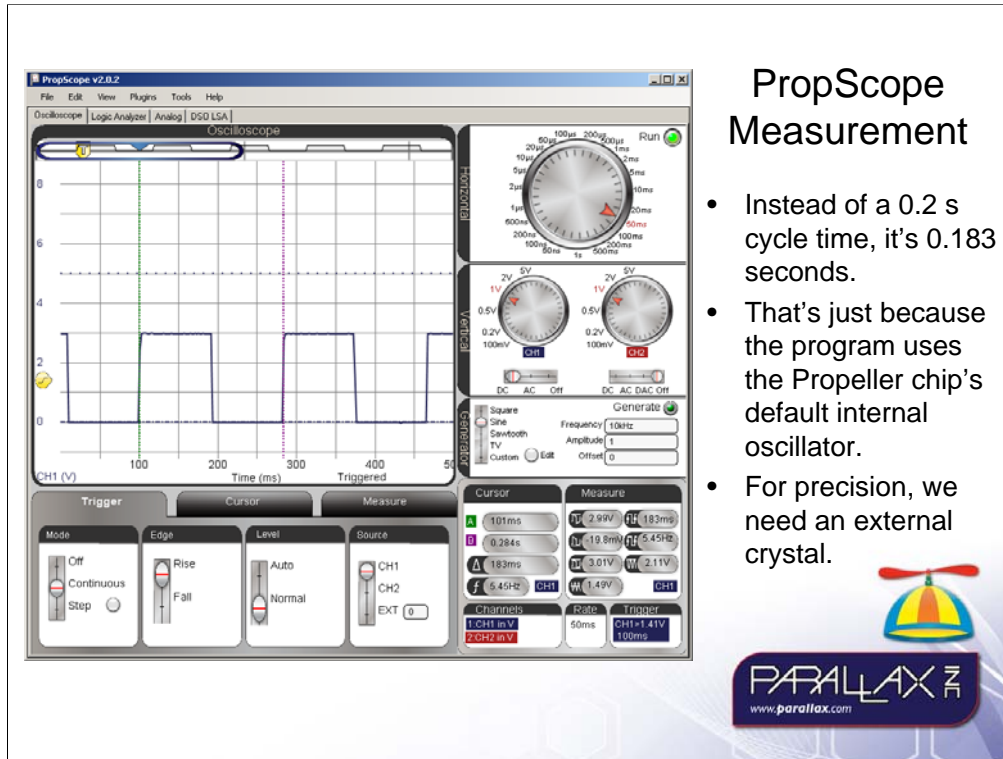
- Each high/low and low/high transition is separated by 0.1 s pause.
- The program looks like the cycle time should be 0.2 s.
- Since there are no system clock settings in the program, the Propeller uses its internal oscillator, which is not precise.
- Let's see what happens...



This example program Demonstrates a PE Kit style first introduction to I/O and timing with a simple blinking light. In the Propeller Users Group presentation on January 27th, the PropScope was used to measure the signal's timing, which was a little off because the program relies on the Propeller's internal oscillator, which is not designed for precision.

Keep in mind that this is the university level introduction, so there are lots of nested concepts in terms of the commands for I/O pin direction and state control, and the waitcnt command with its target cnt register value determined by a fraction of clockfreq added to the cnt register.

This is program 01. All these concepts will be converted to simple method calls for beginners in program 05.



PropScope Measurement

- Instead of a 0.2 s cycle time, it's 0.183 seconds.
- That's just because the program uses the Propeller chip's default internal oscillator.
- For precision, we need an external crystal.



PARALLAX 
www.parallax.com

With the imprecise clock, the code intended to make the LED flash on/off with a 0.2 s cycle time has a 0.183 s cycle time instead.

PropScope



- To learn more about the PropScope USB oscilloscope hardware and software we are using to take these measurements, go to:

www.parallax.com/go/PropScope



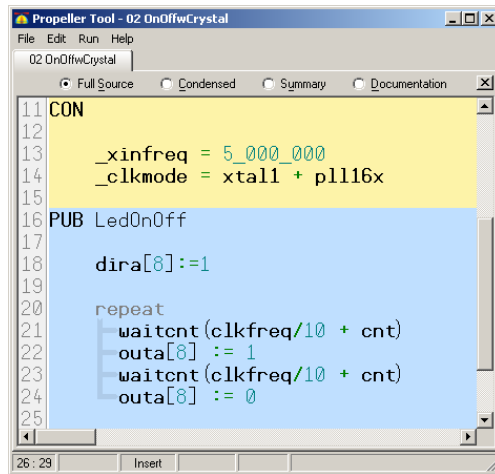
The PropScope is a great little device that takes voltage measurements at up to 25 mega samples per second. It has two oscilloscope channels, a function generator, a 4-channel logic analyzer, spectrum analyzer, xy plotter and more. There's a nice PropScope resource page at:

www.parallax.com/go/PropScope.

You can get to a new Stamps in Class PropScope book draft by going to:

forums.parallax.com -> Forum -> PropScope -> [Understanding Signals with the PropScope Book \(Early Drafts\)](#)

02 OnOffwCrystal.spin



```
11 CON
12
13   _xinfreq = 5_000_000
14   _clkmode = xtal1 + pll16x
15
16 PUB LedOnOff
17
18   dira[8] := 1
19
20   repeat
21     waitcnt (clkfreq/10 + cnt)
22     outa[8] := 1
23     waitcnt (clkfreq/10 + cnt)
24     outa[8] := 0
25
```

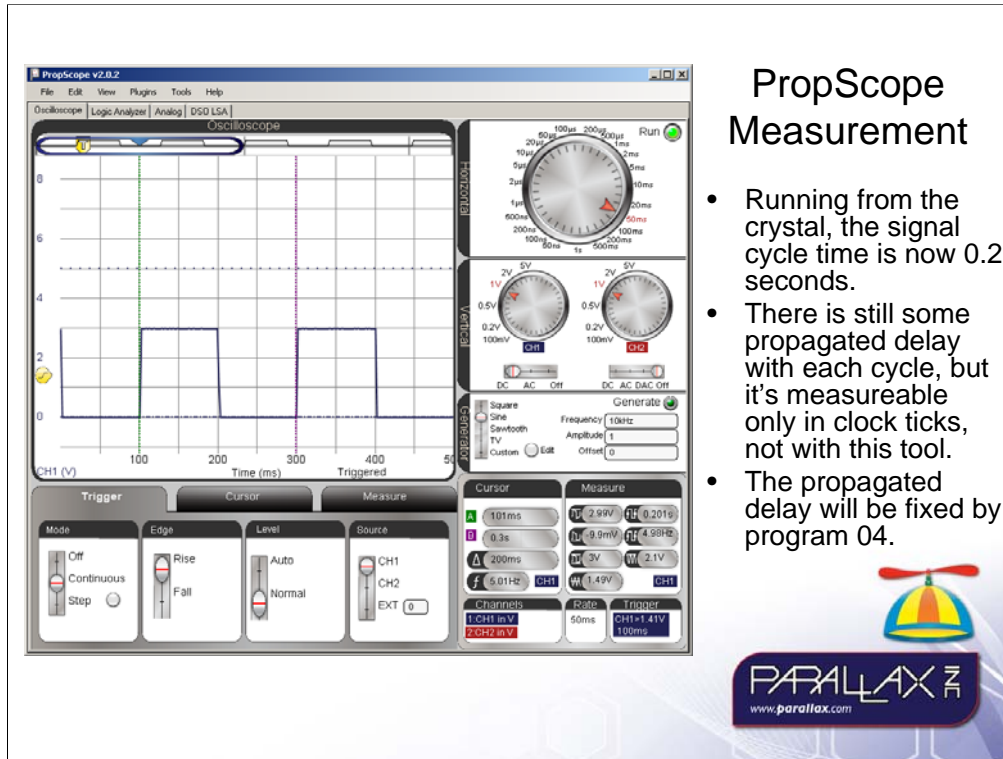
- The first line of defense is to tell the Propeller to use the PE Kit's external oscillator, which is much more precise.
- The CON block tells the Propeller to use its external 5 MHz oscillator.
- With pll16x, the system runs at 80 MHz.
- Let's see what happens now...



By adding a CON block that configures the Propeller to use the PE Kit's external 5 MHz crystal (which is much more precise) it corrects the timing inaccuracies of the previous example program.

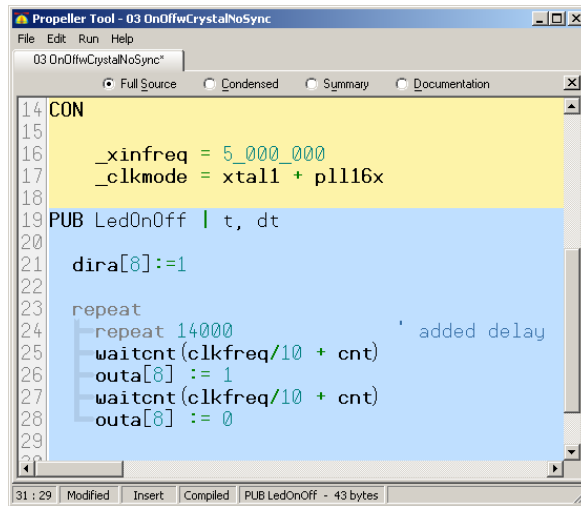
Again, keep in mind that this is the PE Kit coding style, using the CON block to set the constants that affect the system clock. This entire block will be replaced by a single method call with more intuitive names in program 06.

Note: For lower power, configure the system to run slower. For example, pll1x would make the system run at 5 MHz instead of 80.



The more precise crystal definitely improves the crystal timing. Because of the way the delays are calculated, there is still a very slight propagated delay with each loop repetition. With this program, it's not measurable on the PropScope because it only involves the amount of time it takes to execute a few spin commands. The next program has modifications that highlight the delay, and the program after that corrects the problem.

03 OnOffwCrystalNoSync.spin



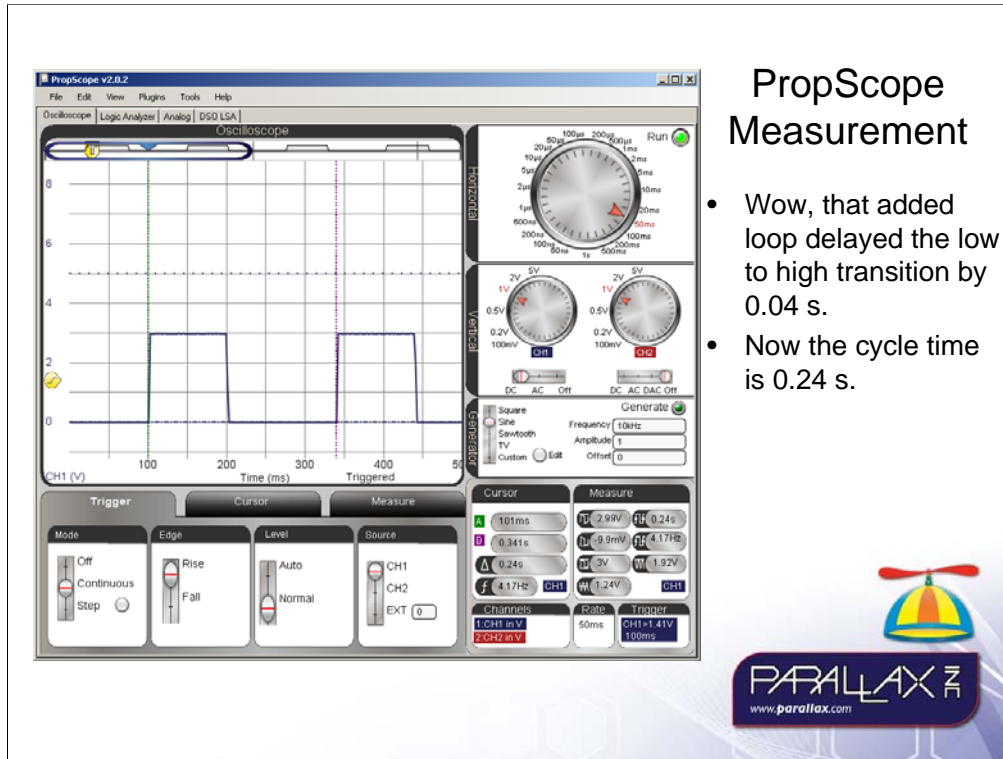
```
14 CON
15
16   _xinfreq = 5_000_000
17   _clkmode = xtal1 + pll16x
18
19 PUB LedOnOff | t, dt
20
21   dira[8] := 1
22
23   repeat
24     repeat 14000 ' added delay
25     waitcnt(clkfreq/10 + cnt)
26     outa[8] := 1
27     waitcnt(clkfreq/10 + cnt)
28     outa[8] := 0
29
30
```

The plot thickens:

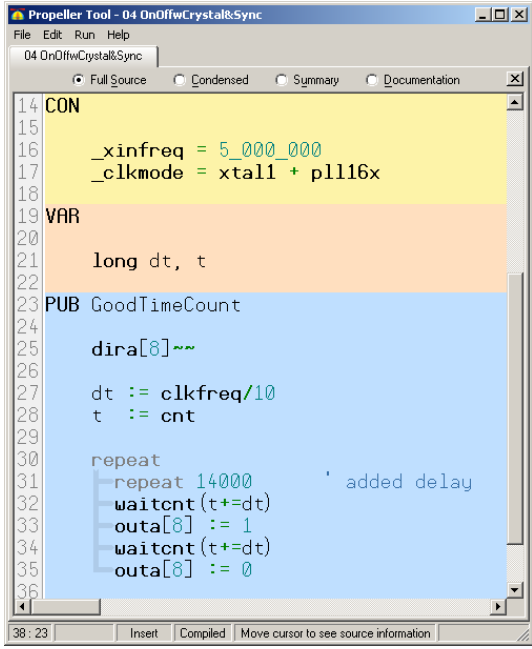
- Here we have a delay added before the I/O pin transitions from low to high.
- It takes time to count to 14,000 in Spin, but the delays are not compensating, so there should be a visible effect on the signal's timing.



This example program highlights an aspect of the repeat loop that can have undesirable effects on code that needs to have precisely timed events. The line with repeat 14000 adds a delay to the code that visibly affects the signals timing when viewed with the PropScope. So, even though the crystal is precise, your code can still propagate errors if you're not careful.




That extra code caused the cycle to take 0.24 s to repeat instead of 0.2 s. Proof positive that this coding approach might be okay for some applications, but we need a better solution for precisely timed loops and events.



```
14 CON
15
16   _xinfreq = 5_000_000
17   _clkmode = xtall + pll16x
18
19 VAR
20
21   long dt, t
22
23 PUB GoodTimeCount
24
25   dira[8]~~~
26
27   dt := clkfreq/10
28   t := cnt
29
30   repeat
31     repeat 14000      ' added delay
32     waitcnt (t+=dt)
33     outa[8] := 1
34     waitcnt (t+=dt)
35     outa[8] := 0
36
```

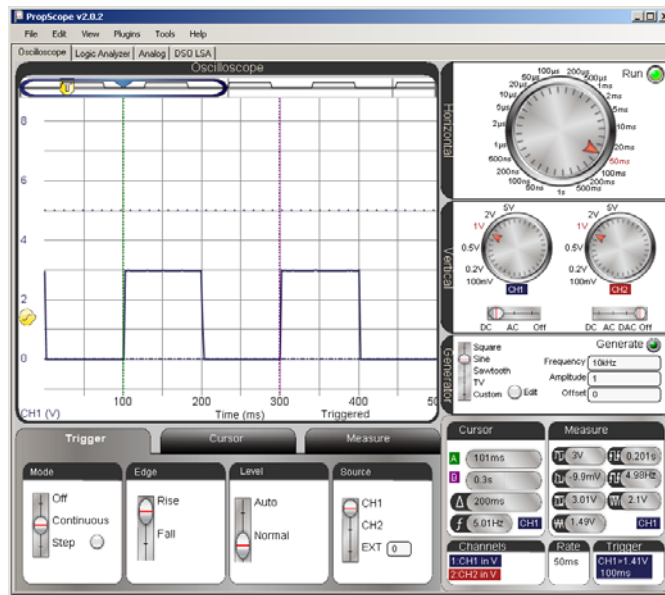
04 OnOffwCrystal &Sync.spin

- The PE Kit's synchronized delay approach should correct the problem.
- Let's take a look...



This program uses an approach called synchronized delays to make the timing of the LED on/off transitions precise, even with the added delay. This approach is introduced in Propeller Education Kit Labs: Fundamentals near the end of Chapter 4, and it is also introduced and explained in the Propeller Manual.

PropScope Measurement



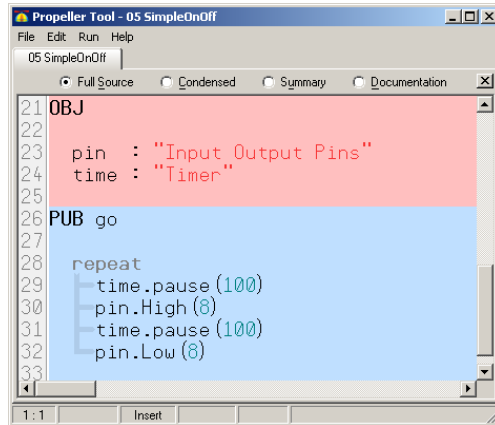
- ...and sure enough, the signal's cycle time is back to 0.2 s.
- With this approach, it's more accurate than the PropScope can show.
- Program 02, which has small amounts of propagated error that can accumulate over time. This program (04) does not.



PARALLAX
www.parallax.com

In this case, the transitions are separated by a fixed number of clock ticks. So in addition to a cycle time that looks about the same as it did in program 02, the timing is actually precise down to the clock tick. Parallax Serial Terminal can be used to display the cnt register values, but due to time constraints, we will have to save that subject of another Spin Tip.

05 SimpleOnOff.spin



```
21 OBJ
22
23   pin : "Input Output Pins"
24   time : "Timer"
25
26 PUB go
27
28   repeat
29     time.pause(100)
30     pin.High(8)
31     time.pause(100)
32     pin.Low(8)
33
```

- Same functionality as 01 LedOnOffP8Imprecise.spin
- Easier to read
- Easier to understand
- Judicious object nicknames that play nice with method names make each call even more self explanatory.
- Method calls are slightly slower than I/O commands, but it's a worthwhile tradeoff for simplicity in beginner examples.



This is the first of several example programs that show how “building block” objects can be used to make Propeller programming more accessible to students who might be closer to, or even at the beginner level.

This program has the same functionality as “01 LedOnOffP8Imprecise.spin”, but it is much easier to understand. This example program uses two objects, Input Output Pins and Timer to control the I/O pins and the delays between switching them from high to low and from low to high again in the repeat loop. The “Input Output Pins” object has methods like High and Low for setting I/O output states, and Pause for delaying. By giving the objects nicknames like pin and time, the method calls read pin.High(8) and time.Pause(100). For a beginner, these method calls are much more approachable than outa[8]:=1, dira[8]:=1, and waitnt(clkfreq/10+cnt).

Keep in mind that this program performs almost identically to “01 LedOnOffP8Imprecise.spin”. So we should expect to see about the same internal clock timing error we saw in slide 13.

Another thing to keep in mind: There is a slightly larger propagation delay with each cycle because it takes more clock ticks to call a method than it does to just directly adjust the value of a bit in the outa register. Obviously, for this type of beginner example, it's a worthwhile tradeoff.

```

1 Object "Input Output Pins" Interface:
2
3 PUB High(pin)
4 PUB Low(pin)
5 PUB Check(pin) : state
6 PUB Dir(pin, direction)
7 PUB Set(pin, state)
8 PUB Toggle(pin)
9 PUB Reverse(pin)
10 PUB CheckGroup(last, first) : states
11 PUB DirGroup(last, first, directions)
12 PUB SetGroup(last, first, states)
13 PUB ToggleGroup(last, first)
14 PUB reverseGroup(last, first)
15
16 Program:      31 Longs
17 Variable:    0 Longs
18
19
20 PUB High(pin)
21
22 Set I/O pin to output-high. Pin will transmit 3.3 V.
23 PARAMETER: pin = I/O pin number
24
25
26 PUB Low(pin)
27
28 Set I/O pin to output-low. Pin will transmit 0 V.
29 PARAMETER: pin = I/O pin number
30

```

Input Output Pins Object

- Building block objects have documentation comments that make them their own syntax manuals.
- Try opening 05 SimpleOnOff.spin. Then press F8 to compile, and double-click the Input Output Pins object. Then, click the Documentation radio button.
- The Object Interface at the beginning of the file gives you a quick overview of the methods.
- Each method's function and parameters are also documented below the Object Interface.



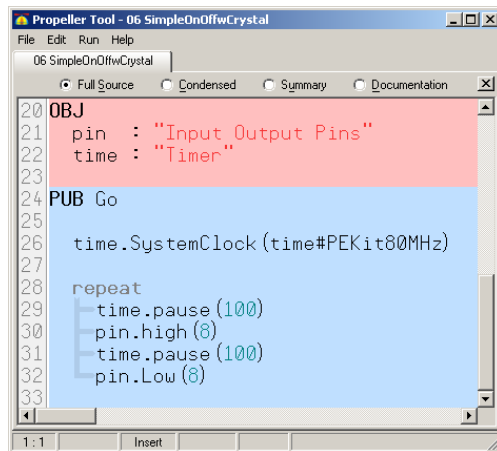
Building block objects are designed to simplify coding tasks. Their documentation comments should make them their own syntax manuals. Provided the documentation comments are properly written, a coder should only have to look at a building block object in Documentation View for the information he/she needs.

For example, here is the Input Output Pins object in documentation mode. At the beginning of the object, it has a list of methods called the Object Interface. The method names were chosen to be brief, yet self explanatory to make this list easier to browse. Each method also has its full documentation below in the same order shown in the list. We can see method documentation for the High and Low methods near the bottom of the excerpt. As you can probably see by the size of the window's scrollbar, there's lots more documentation in the file.

Building block objects usually come zipped in packages with example code that demonstrates how they can be used. Ideally, they have a simple test program, and a more full featured demo program. For example, all the code examples in this slide show are in a zipped package, and the Input Output Pins and Timer Objects have documentation comments. At this point, the zipped package's example programs are of the simple test variety; there aren't full demo programs for Input Output Pins and Timer yet.

Also, simple DC Motor does not yet have documentation comments. Remember, all these objects are works in progress. They were not yet ready to publish at this time, but again, they addressed the Spin Tips request for this meeting perfectly, so here they are, just not in their final states.

06 SimpleOnOffwCrystal.spin



```
20 OBJ
21 pin : "Input Output Pins"
22 time : "Timer"
23
24 PUB Go
25
26   time.SystemClock (time#PEKit80MHz)
27
28   repeat
29     time.pause (100)
30     pin.high (8)
31     time.pause (100)
32     pin.Low (8)
33
```

- Same functionality as 02 OnOffwCrystal.spin.
- Again, much easier to read, and write!
- The SystemClock method call replaces the entire CON block from program 02.
- Signal timing is corrected, same as program 02.
- The SystemClock call uses a constant built into the Timer object to make the PE Kit run at 80 MHz using its external crystal.



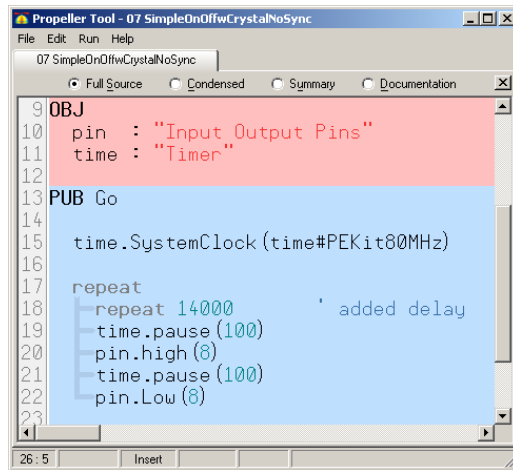
Program 05 had the same weakness as program 01; it did not specify any oscillator settings. As a result, the signal had about the same timing error as program 01.

This example program (06) fixes that problem, but in a much simpler way, with one method call that tells the Timer object that we want to use the PE Kit's crystal oscillator and run the system clock at 80 MHz. The result is the same as the oscilloscope display in slide 16.

In contrast, this program's PE Kit Labs counterpart (02 OnOffwCrystal.spin) used a lot more terminology that's related to the Propeller chip's architecture to do the same job. Examples include: `_xinfreq`, `_clkmode`, and `xtal` and `pll`, all in a CON block. For a beginner, that's a lot of terminology and concepts to wade through to make the same correction that a single method call can make.

IMPORTANT: The Timer object is just a proof of concept for educational applications at this point, and it is still a work in progress. It is not intended to be immediately taken and used in industrial applications. If you want something that behaves similarly but is officially released, use the Clock object. It's in the Propeller Tool software's Propeller Library.

07 SimpleOnOffwCrystalNoSync.spin



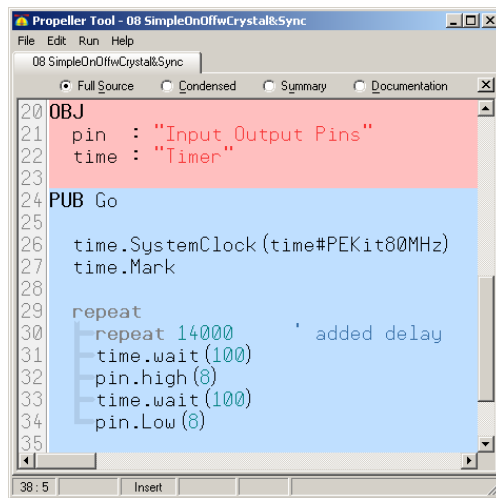
```
9 OBJ
10 pin : "Input Output Pins"
11 time : "Timer"
12
13 PUB Go
14
15 time.SystemClock(time#PEKit80MHz)
16
17 repeat
18 repeat 14000 ' added delay
19 time.pause(100)
20 pin.high(8)
21 time.pause(100)
22 pin.Low(8)
23
```

- Same functionality as 03 OnOffwCrystalNoSync.spin,
- The only change from the previous program is repeat 14000 added to make it add delay to the loop.
- Delay is added to demonstrate how pause is not designed for synchronized delays.
- Cycle time gets increased by 0.4 seconds in the PropScope, just like with program 03.



This demonstrates the same potential coding problem that was first introduced in program 03. The result is an oscilloscope display that's about the same as slide 18. The low signal gets stretched by 0.4 s.

08 SimpleOnOffwCrystal&Sync.spin



```
20 OBJ
21 pin : "Input Output Pins"
22 time : "Timer"
23
24 PUB Go
25
26 time.SystemClock(time#PEKit80MHz)
27 time.Mark
28
29 repeat
30 repeat 14000 ' added delay
31 time.wait(100)
32 pin.high(8)
33 time.wait(100)
34 pin.Low(8)
35
```

- Same functionality as 04 OnOffwCrystal&Sync.spin.
- Instead of all the synchronized delay bookkeeping in program 04, you can use the Timer object's built-in synchronized delay features:
 - Call time.Mark to mark a current time.
 - Call time.Wait(100) to wait for 100 ms from time.Mark.
- The result is the same as program 04, precise signal transitions, regardless of how much time other code in the loop takes (so long as it's less than 100 ms).



This demonstrates the same solution to the potential coding/timing problem that was first shown in program 04. By calling methods in the Timer object, the program solves the problem with a much lighter conceptual load because the Timer object takes care of the cnt register bookkeeping for synchronized delays.

The resulting oscilloscope display is identical to the one in slide 20, with zero clock ticks of propagated error.

Note how there is a call to the Timer object's Mark method to “mark” the time. After that, the first wait method call waits for 100ms from the Mark. It also updates the marked time so that the next wait method call waits an additional 100 ms. All of the bookkeeping is done in terms of clock ticks, so the loop never adds any propagated delays.

```

10 OBJ
11
12   time : "Timer"
13   motor : "Simple DC Motor"
14
15
16 PUB go | speed
17
18   time.SystemClock(time#PropBOE)
19
20   speed := 0
21
22   motor.Start(12, 13, @speed)
23
24   repeat 2
25     repeat speed from 25 to 100
26       time.pause(100)
27     repeat speed from 100 to 25
28       time.pause(100)
29     repeat speed from -25 to -100
30       time.pause(100)
31     repeat speed from -100 to -25
32       time.pause(100)
33
34   speed := 0
35   motor.stop
36

```

09 Test Simple Motor Speed.spin

- This example program uses an object called Simple DC Motor.spin (nicknamed motor) to send a PWM signal to a DC motor to control its speed.
- PWM for motor control typically has a constant cycle time with variable signal high time.
- The motor object uses synchronized delays to maintain the constant cycle time.
- The ratio of signal high time to cycle time is called duty cycle.
- The motor object watches the speed variable in this program (09 Test Simple Motor Speed), and uses its value to set the PWM signal's duty cycle.



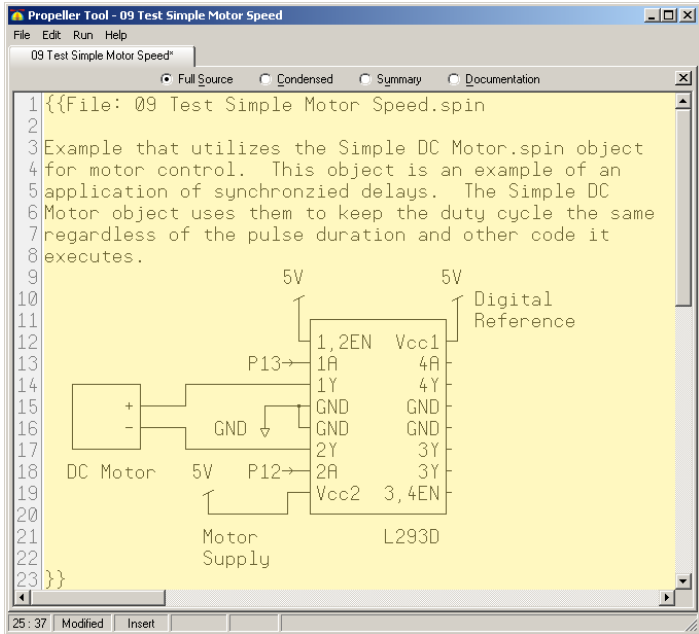
This program is an example of motor dc motor control, which is one of many applications of synchronized delays. The code uses an object named Simple DC Motor to send signals that control a motor with another cog. The motor control signal is a 1 kHz, variable duty cycle signal. The duty cycle of the signal is the ratio of high time to cycle time. Longer high times to a driver chip cause it to allow current through the motor for larger fractions of the time. It's equivalent to applying a higher voltage to the DC motor. Likewise, lower duty cycles (less high time to cycle time) cause the driver chip to allow current through the motor less of the time, which corresponds to lower voltage.

Another interesting thing about this program from the beginner standpoint - it is designed to be simple and approachable. It also support a programming lesson on the virtues of pointers. A call to the object's Start method passes the motor control I/O pins, and the address of a variable named speed. After starting the object, the speed variable can be adjusted to values in the -100 to 100 range, where -100 corresponds to full speed one direction, 0 corresponds to full stop, and 100 corresponds to full speed in the opposite direction.

Since the Simple DC Motor object has the address of the speed variable, it watches speed from another cog and adjusts the motor control signals based on the value the speed variable stores. The coolest thing about this arrangement is that all the example program has to do is change the speed variable, and the object automatically updates the signaling to the motor. The net result: change the variable value, and motor speed changes. Change the sign of the speed variable, and the direction of the motor's rotation changes.

Remember, Simple DC Motor is one of the objects that is currently a work in progress, and also keep in mind that it will be primarily for educational use.

Test Circuit



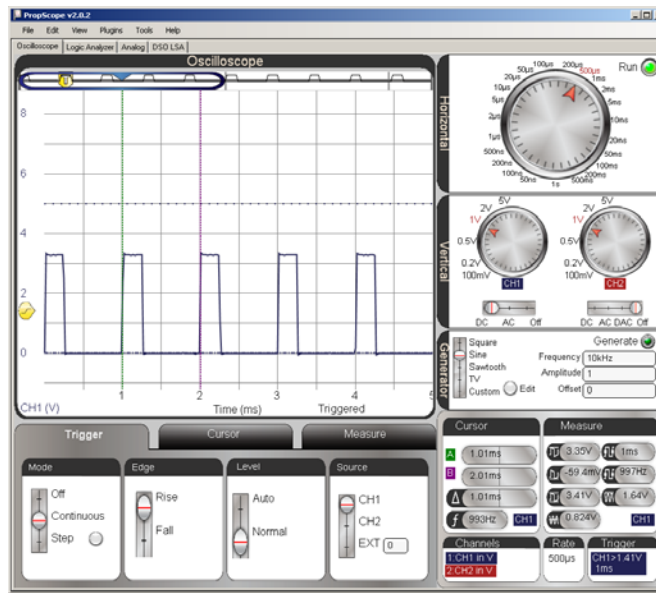
- You can use the Propeller Tool software's Character Chart to draw schematics.
- To open the Character Chart, click Help and select View Character Chart.

If the Simple DC Motor object sends a low signal to P12 and a PWM signal to P13, current passes through the motor from 1Y to a grounded 2Y. If Simple DC Motor instead sends a low signal to P13 and a PWM signal to P12, the chip allows current from 2Y through the motor, to ground in 1Y.

For background information on circuits used to drive DC motors, look up H-bridge on Wikipedia. The L293D chip contains four half H-bridge circuits. One half of an H-bridge can drive a motor in one direction, two halves allow you to drive a motor either direction.

Four half H-bridges give you a lot of options. For example two half bridges make a full H-bridge that can drive one motor two directions. For larger motors, the half bridges can be connected to the motor in parallel to accommodate higher current demands. A single half bridge can drive a motor in a single direction, so you could even control four separate motor speeds, but each one would only have a single direction option.

PropScope Measurement



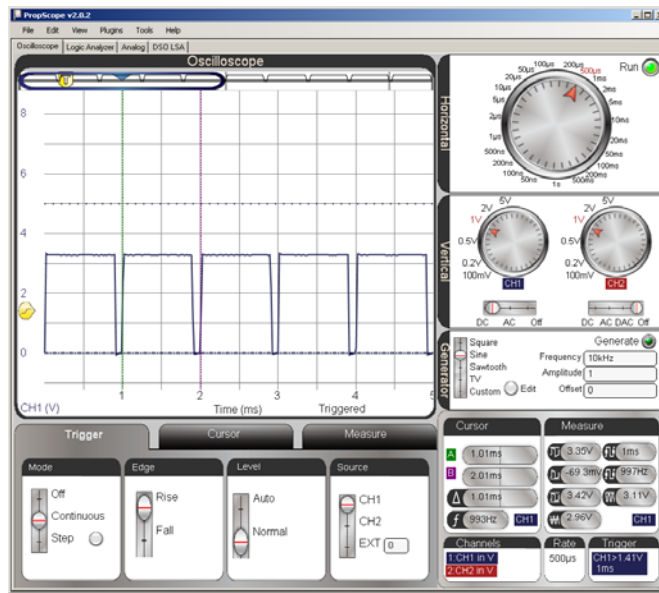
- When the speed variable in 09 Simple Motor Speed.spin is set to 25, we have a 25% duty cycle, so the signal is high 25% of the time.
- The cycle time is 1 ms.
- If the speed variable is a different value, only the high time should change, not the cycle time, which is controlled by synchronized delay code.



PARALLAX
www.parallax.com

This is an example of the PWM motor control signal while the speed variable is set to 25. It makes the 1 kHz PWM that goes to the motor driver chip send a high signal for 25% of the time. Regardless of how long it's high, the synchronized delays in the Simple DC Motor object maintains the 1 ms cycle time. This signal causes the motor to turn slowly.

PropScope Measurement



- Here is an example with the speed variable set to 90.
- Notice that the cycle time is still 1 ms, it has not changed, thanks to synchronized delay code in the Simple DC Motor object.



When the speed variable is changed to 90, the signal stays high 90% of the time, but the frequency remains at 1 kHz (1 ms cycle). Again, that's because of the Simple DC Motor object's synchronized delay code. This signal causes the motor to turn at almost full speed.

```

38
39 t := cnt
40 dt := clkfreq/1000
41
42 repeat
43   speed := long[motorAddr] #> -100 <# 100
44   if speed == 0
45     phsa~
46     frqa~
47     outa[posPin]~
48     outa[negPin]~
49   elseif speed > 0
50     outa[negPin]~
51     ctra[5..0] := posPin
52     frqa := 1
53     phsa := ticks*(-speed)
54   elseif speed < 0
55     outa[posPin]~
56     ctra[5..0] := negPin
57     frqa := 1
58     phsa := ticks*speed
59   waitcnt(t+=dt)
60

```

Simple DC Motor.spin

- The synchronized delay code is the first two lines and last line in this code excerpt.
- The program uses a counter module to set the pulse duration.
- There's more information about counter modules in the PE Kit Labs book (Chapter 7).



This current incarnation of Simple DC Motor uses the PE Kit approach to Spin language motor control. I have not yet simplified it to use the Input Output Pins and Timing building block objects.

The first two lines in the program mark the current time and set the time increment (to the number of clock ticks in 1/1000 of a second). Then, in the repeat loop, the code does its thing, and might take varying amounts of time depending on how many conditions it rejects before getting to a true one. Just before the loop repeats, the waitcnt(t+=dt) command adds dt to the marked cnt register value of t, and then waits until the cnt register gets to that point. By doing that each time, it ensures that the repeat loop repeats every 1 ms worth of clock ticks. For example, if the system clock is configured for 80 MHz, the loop repeats every 80,000 clock ticks because $dt := clkfreq/1000 = 80,000,000/1000 = 80,000$.

Now, if you made the mistake of replacing waitcnt(t+=dt) with waitcnt(clkfreq/1000 + cnt), each loop repetition would take slightly longer than 1 ms to repeat. True, it's not as critical as it would be with actual timekeeping or feedback for a critical process, but it's still nice to keep the process "as advertized", with a 1 kHz PWM signal.

Propeller Education Resources

- www.parallax.com/go/PEKit
- forums.parallax.com -> Forum -> Propeller Chip -> Propeller Education Kit Labs, Tools, and Applications
- forums.parallax.com -> Forum -> PropScope -> Understanding Signals with the PropScope Book (Early Drafts)
- Comments? Contact me on the parallax forums. My username is Andy Lindsay (Parallax).



I hope you enjoyed this Spin Tip. Here is a list of Parallax Educational resources that are currently available for the Propeller chip. Keep in mind that if you have the latest Propeller Tool software, you can just click Help to get to these resources, and A LOT more. If you have any questions or comments, please feel free to drop me a line through the Parallax Forums: forums.parallax.com. My username there is: Andy Lindsay (Parallax).