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

PE Kit Tools: Transmit Square Wave Frequencies

Contents

Introduction

Each Propeller microcontroller cog can configure either or both of its counter modules for transmitting square waves in the background. This approach is especially useful since it allows the Spin code to move on to other tasks while the counter module transmits the square wave.

The SquareWave object has methods that reduce all the counter module configuration legwork to a few simple method calls. These methods can configure a given cog's counter module(s) to transmit either single-ended or differential square waves at integer frequencies up to 128 MHz. An application that uses more than one cog to send square waves typically only needs a single instance of the SquareWave object, because the cog that calls the SquareWave object's frequency methods gets its own counter modules configured for the task.

This article includes a few circuit and test code examples that demonstrate using the Square wave object to transmit square waves in the subsonic, sonic, and ultrasonic ranges, and it also has a 50+ MHz example. Square wave modulation examples include single ended, differential, and pulse controlled. Propeller Education Kit circuits include: indicator lights, piezospeaker tones, infrared transmitter and receiver, and metal detector.

Square Wave Applications – Simple Indicator Light Tests

For this first application example, the code sends square waves at frequencies that are convenient for viewing with light emitting diodes included in the Propeller Education Kit:

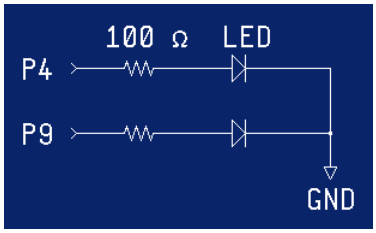


Figure 1: LEDs circuits for square waves at indicator light frequencies

The SquareWave object really easy to use. Just declare the object, and then pass pin, channel, and frequency values to its Freq method. Here is an example from "Test SquareWave with LEDs.spin" that sends two different square waves on two different I/O pins.

```
''Test SquareWave with LEDs.spin

OBJ

  sqw : "SquareWave"          ' Declare Square wave object

PUB go

  sqw.Freq(4, 0, 10)          ' P4, channel 0, 10 Hz
  sqw.Freq(9, 1, 25)          ' P9, channel 1, 25 Hz

    ' Square waves start, code moves on to other tasks.
```

After declaring the SquareWave.spin object and giving it the nickname sqw, a call to the Freq method with I/O pin (0 to 31), channel (0 or 1) and the frequency (0 to 128_000_000) is all it takes to make an I/O pin start transmitting a square wave. The method call starts the square wave and returns almost immediately so that the code can move on to other tasks.

Although each square wave is an independent process, it is not actually transmitted by another cog. Instead, it is transmitted by one of the two counter modules built into each cog. A counter module is a state machine that can perform a variety of repetitive tasks for the cog, and transmitting square wave signals is one of those tasks. To find out more about counter modules and lots of examples of objects that utilize them, see 7: Counter Modules and Circuit Applications Lab in *Propeller Education Kit Labs: Fundamentals (.pdf)*.

The "Test SquareWave with LEDs.spin" object is part of the "PE Kit Tools – Square Waves.zip" package, and after launching two square wave processes, it waits for seven seconds, then shuts them back down again using the SquareWave object's End method. Try it out:

- ✓ Build the Figure 1 Schematic.
- ✓ Download and unzip "PE Kit Tools – Square Waves.zip".
- ✓ Open "Test SquareWave with LEDs.spin" with the Propeller Tool
- ✓ Load "Test SquareWave with LEDs.spin" into the Propeller chip, and verify that it makes the P4 LED circuit blink 10 Hz and the P9 LED circuit blink at 25 Hz. Both LEDs should blink for 7 seconds, then stop.

Square Waves in Differential Mode

Differential mode sends opposite square wave on two I/O pins. When one I/O pin is high, the other is low, and vise versa. This can be useful for sending signals over transmission lines. For example, Ethernet communication depends on differential signals.

- ✓ Build the circuit shown in Figure 2

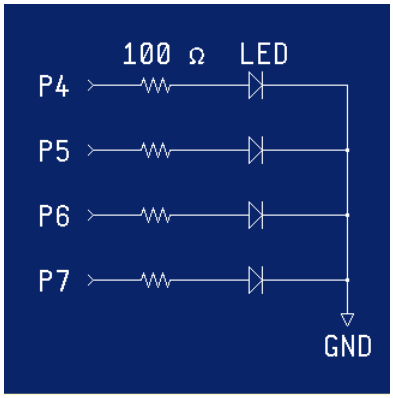


Figure 2: LEDs circuits for differential and multi-cog square waves

The SquareWave object has a FreqDiff method for transmitting square waves in differential mode. The difference between Freq and FreqDiff is that FreqDiff has a second pin argument for the I/O pin that sends the NOT of the signal on the first pin.

```
' 'Test SquareWave with LEDs (Differential).spin

OBJ

  sqw : "SquareWave"          ' Declare Square wave object

PUB go

  ' Transmit a 3 Hz differential signal on P4 and P9 using channel 0.
  sqw.FreqDiff(4, 7, 0, 3)

  ' Code moves on to other tasks...
  repeat
```

- ✓ Load "'Test SquareWave with LEDs (Differential).spin" into the Propeller chip, and verify that P4 and P7 blink at the same rate with opposing states (while one is off the other is on…).
- ✓ Try adding a second FreqDiff method call to make P5 and P6 blink differential at a different rate. Make sure to use channel 1 instead of channel 0 for the second FreqDiff method call.

As with frequencies, each channel's mode (single ended or differential) can also be configured independently.

- ✓ Try changing one of the two FreqDiff method calls to Freq. You'll have to remove one of the I/O pin arguments.
- ✓ Verify that the modified program sends one single-ended and one differential square wave.

More Square Waves? More Cogs!

Each cog has two counter modules, so a given cog can transmit up to two square waves. A cog's counter modules can also be used for A/D conversion, D/A conversion, RC decay measurements, and a variety of other tasks. Some of these tasks occur in the background while others require periodic intervention by the code the cog is executing. So if you your application needs to transmit more than two square waves, or other counter module applications, make sure to launch your code into a new cog before starting the next square wave or other process.

The nice thing about this arrangement is that your cog is still free to perform other tasks in the foreground. As soon as the Spin code has started the square wave, it can move on and take care of other things. Also, most applications only need one instance of the Square Wave object, regardless of

how many cogs have to transmit square waves. The SquareWave object automatically updates the counter modules in the cog that called its methods.

"Test SquareWave with LEDs (Two Cogs).spin" uses two cogs to send four different square wave signals. This example program also uses the SquareWave object's FreqUpdate method to change the frequency of the first cog's channel 0 square wave to 5 Hz.

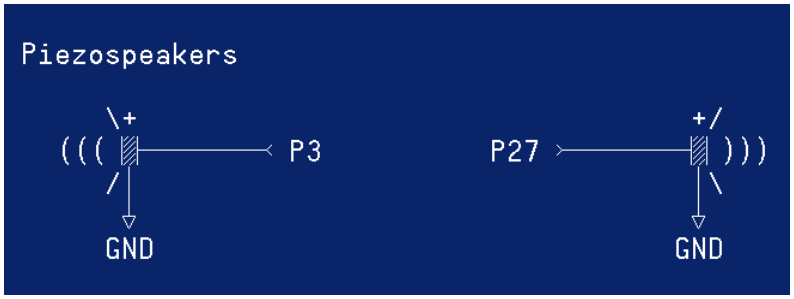
- ✓ Examine the program. How long from the start of the program will it take for the second cog starts transmitting square waves? How long will the second cog transmit? How long after the second cog starts transmitting will it take for the first cog to update its channel 0 square wave frequency?
- ✓ Load the program into the Propeller chip and reconcile any differences between your predictions and the way the program behaved.

```
'' Test SquareWave with LEDs (Two Cogs).spin
''Generate four square waves running in the background of two cogs.

VAR

  long stack[50]                                ' Stack for SecondProcess

OBJ

  sqw : "SquareWave"                            ' Declare Square wave object

PUB go

  'clkgen.Freq(I/O pin, module=0 or 1, frequency in Hz)

  sqw.Freq(4, 0, 10)                            ' P4, channel 0, 10 Hz
  sqw.Freq(5, 1, 25)                            ' P9, channel 1, 25 Hz

  'Cog moves on to other jobs and lets counter modules blink the lights.
  waitcnt(clkfreq*3+cnt)                        ' Blink for 10 seconds

  cognew(SecondProcess, @stack)                 ' Launch another cog

  waitcnt(clkfreq*2+cnt)                        ' 4 square waves, 2 more seconds

  sqw.FreqUpdate(0,5)                           ' Channel 0 freq to 5 Hz

  waitcnt(clkfreq*2+cnt)                        ' 4 square waves, 2 more seconds

  sqw.End(0)                                    ' End Channel 0 transmission
  sqw.End(1)                                    ' End Channel 1 transmission

PUB SecondProcess

  ' This second cog uses the same square wave object to configure its counter
  ' modules to create two more square waves.  As with the other cog, this cog's
  ' does not need to do anything to keep the square waves going, so it can move
  ' on to other tasks.

  sqw.Freq(6, 0, 11)                            ' P6, channel 0, 17 Hz
  sqw.Freq(7, 1, 23)                            ' P7, channel 1, 25 Hz

  waitcnt(clkfreq*5+cnt)                        ' Send for about 5 seconds

  sqw.End(0)                                    ' End Channel 0 transmission
  sqw.End(1)                                    ' End Channel 1 transmission
```

Square Wave Applications – Audio (20 to 20 KHz)

Some sonic applications for square waves include indicator tones, simple musical notes, sound effects, alarm tones.

Figure 3: Piezospeakers for two audio frequency square waves



Here is some code from "Test Audio Square Waves.spin" that plays C6 and E6 notes on the piezospeakers, first separately, and then simultaneously. First the P3 speaker plays the C6 note for 1 second. After a 1 second pause, the P27 speaker plays the slightly higher E6 note. After another 1 second pause, both speakers play the notes simultaneously. Note how the repeat loop uses the index variable to select successive f0 and f1 frequencies from the lookup tables for use in the SquareWave.Freq method calls. Note also that the waitcnt command at the end of the loop is what gives the notes time to play. Again, this is because the SquareWaves object uses a cog's counter modules to transmit the square waves, and once started, they operate independently, so the program has to decide when to start and stop the square waves.

```
' ' Test Audio Square Waves.spin

CON

  _clkmode = xtal1 + pll16x                     ' Crystal and PLL settings
  _xinfreq = 5_000_000                          ' 5 MHz crystal

OBJ

  sqw : "SquareWave"                            ' Square wave object

PUB go | index, f0, f1

  repeat index from 1 to 6                      ' Indexed loop

    ' Lookup tables copy frequencies into f0 and f1 variables
    f0 := lookup(index: 1047,  0,     0,  0,  1047,  0)
    f1 := lookup(index:    0,  0,  1319,  0,  1319,  0)

    sqw.Freq(3, 0, f0)                          ' Transmit f0 on P3
    sqw.Freq(27, 1, f1)                         ' Transmit f1 on P27
    waitcnt(clkfreq + cnt)                      ' Wait 1 second
```

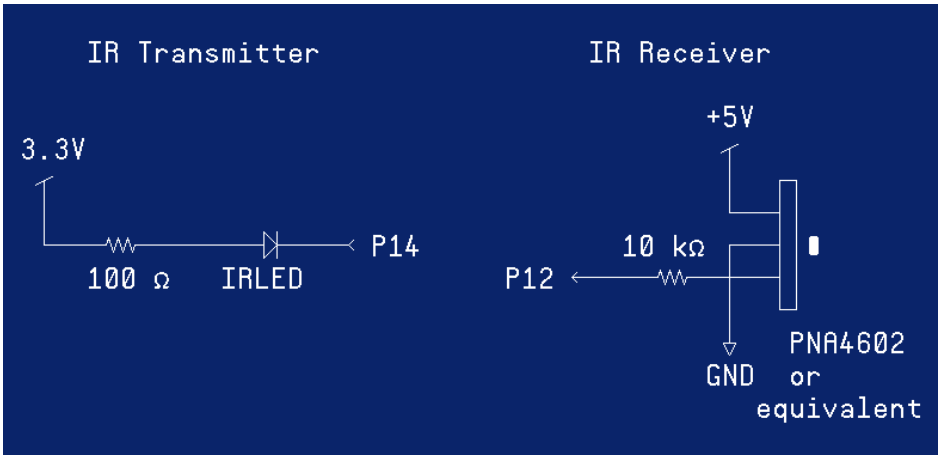
- ✓ Build the Figure 3 Schematic.
- ✓ Open "Test Audio Square Waves.spin" with the Propeller Tool and load it into the Propeller chip
- ✓ Verify that the P3 and P27 speakers play the notes in the sequence described before the code example.
- ✓ Try expanding the lookup tables and composing a tune.

Square waves do not mix as well as sine waves. if you need dual tones on a single speaker, use an object that adds sine waves. (Coming soon to the PE Kit Tools section.)

Square Wave Applications – Ultrasonic (>20 kHz)

Early TV remotes used ultrasound in the 30 – 40 kHz range to control televisions, and their infrared successors used the same frequencies. The IR LED from your TV remote along with the IR receiver inside the TV can be used together for object detection. The foreground code can also modulate the signal for communication the same way the remote does. The cog's code and counter module's I/O pin control goes through an OR gate, so anything sending a high signal overrides any low signals. So the circuit shown in Figure 4 is especially makes coding for modulated signals simple. Other ultrasonic applications include object and distance detection with ultrasonic transducers (speaker and receiver).

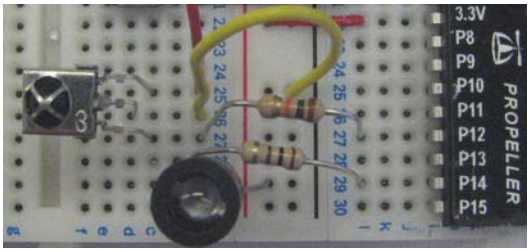
Figure 4: Circuit for infrared object detection or communication

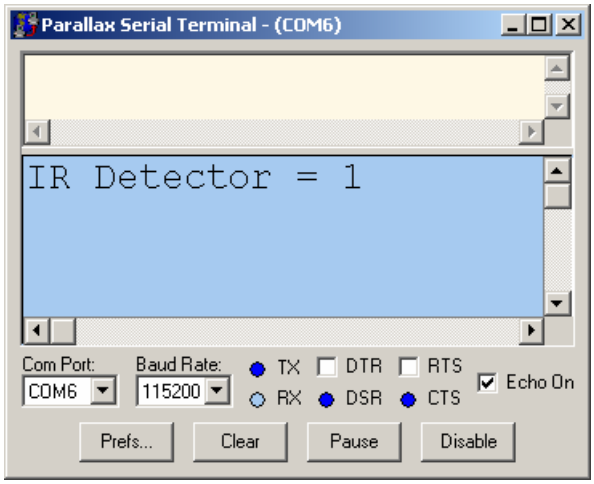


Object Detection Example

Figure 5 shows the wiring for the IR transmitter/receiver so that objects above the board can be detected. Figure 5 also shows the Parallax Serial Terminal displaying the receiver's active-low output. With an active-low output, the receiver indicates that an object is not detected if it sends the Propeller I/O pin a high (1) signal or an object is detected if it sends a low (0) signal.

Figure 5: IR circuit wiring and detection in the Parallax Serial Terminal





This excerpt from "Test IR Object Detection.spin" makes the IR LED transmit a 38 kHz signal for 1 ms, and just before it stops, the code checks the IR receiver's output. It then displays the output (1 = not detected, 0 = detected) in the Parallax Serial Terminal.

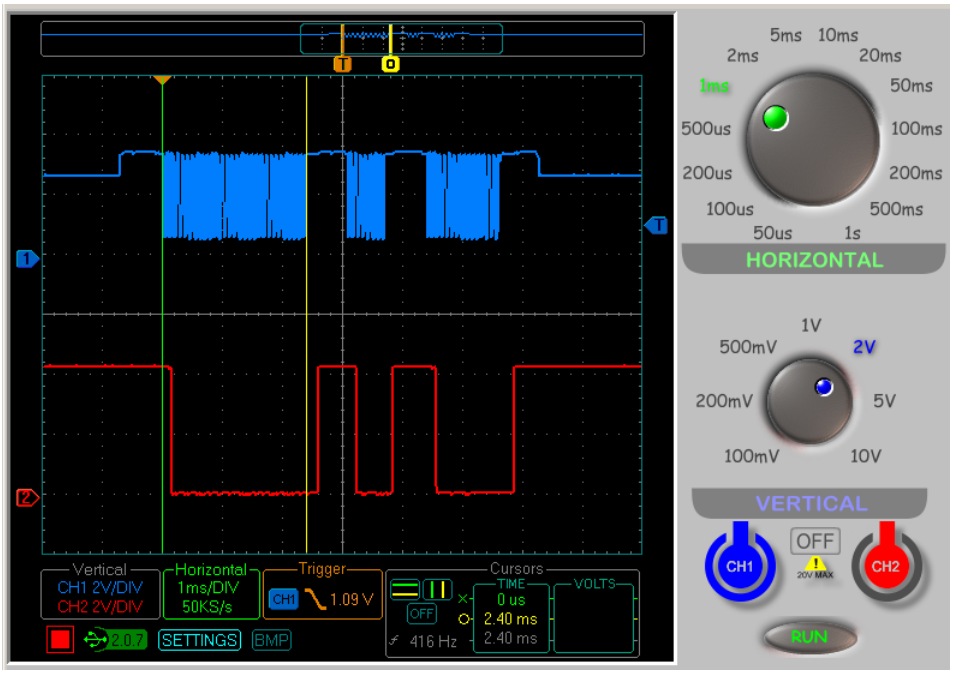
- ✓ If you do not have the Parallax Serial Terminal, complete Propeller <-> PC Terminal Communication before continuing here.
- ✓ Load "Test Ir Object Detection.spin" into the Propeller chip.
- ✓ Display the IR detection status with the Parallax Serial Terminal, and test IR object detection circuit.

```
' 'Test IR Object Detection.spin
' 'Detect 38 kHz IR signal if it reflects off an object.

CON

  _clkmode = xtal1 + pll16x                     ' Crystal and PLL settings
  _xinfreq = 5_000_000                          ' 5 MHz crystal

OBJ

  sqw    : "SquareWave"                         ' Square wave object
  term   : "FullDuplexSerialPlus"               ' Serial communication object

PUB go | detect

  term.StartPST(115200)                         ' Start for Parallax Serial Terminal

  repeat
    sqw.Freq(14, 0, 38000)                      ' IR LED filcker at 38 kHz
    waitcnt(clkfreq/1000+cnt)                   ' Wait 1 ms
    detect:=ina[12]                             ' Check IR receiver
    sqw.Freq(14, 0, 0)                          ' Turn off IR LED

    ' Send cursor to home positon, then display label, display detector value, then
    ' clear to end of line.
    term.str(String(term#HOME, "IR Detector = "))
    term.dec(detect)
    term.tx(term#CLREOL)
```

Modulation Example for Communication – Transmitter

Figure 6 shows a Propeller chip generated example of the first three pulses universal remote sends a SONY receiver (TV set, DVD player, etc.). The channel 1 trace in the upper half of the oscilloscope display is the signal the Propeller chip sends to the IR LED. From left to right, the signal transmits 38 kHz for 2.4 ms, then waits 0.6 ms, then transmits 38 kHz again, this time for 0.6 ms, another 0.6 ms pause, and then 38 kHz for 1.2 ms. With a SONY remote, the 2.4 ms signal indicates the start of a message. The 0.6 ms signal would be a binary-0, and the 1.2 ms signal would be a binary 1. These are the first three pulses of a 13 pulse sequence that sends 12 bits of information to a SONY device.

Figure 6: Transmit three SONY remote control signal bits



The channel 2 trace in the lower half of the oscilloscope display is the signal the IR receiver circuit transmits. The IR receiver is the same one you would find inside a TV set, DVD player, etc, and its output typically goes to a microcontroller inside the component for decoding and processing. The IR receiver sends a low signal to the Propeller chip whenever it receives the 38 kHz modulated infrared.

The IR LED circuit in Figure 4 and Figure 5 is active low so that P14 turns the IR LED on by sending a low signal and off by sending a high signal. All cog counter module and processor I/O control goes through OR gates. With OR gates, if any OR gate input is 1, then its output is a 1. With this arrangement, the code can be the counter module's gate keeper. The code can turn the square wave off by sending a high signal, or let it pass to the I/O pin by sending a low signal.

"Test Modulated IR Signals.spin" demonstrates how the three 38 kHz pulses in Figure 6 were transmitted. First, the program sets the P14 high to prevent any low signals from the SquareWave object from passing to the I/O pin. Then, it starts the 38 kHz signal with a call to SquareWave.Freq, but again, the lows of this signal never make it through to the I/O since the cog's processor is holding the pin high. After waiting for 600 µs, the repeat loop looks up each 38 kHz duration, and copies each successive value to the t variable.

The program starts each pulse by sending a low signal on P14 using outa[14]~. Then, waitcnt(t + cnt) waits for the duration of t, which is 2400 µs the first time though the loop, 600 µs the second time through the loop, and 1200 µs the third time through the loop. To turn the 38 kHz signal back off again, the code sets the I/O pin high with outa[14]~~. Then, waitcnt(600*us + cnt) delays for 600 µs before repeating the loop and sending the next 38 kHz pulse.

- ✓ If you have an oscilloscope at your disposal, load "Test Modulated IR Signals.spin" into the Propeller chip and verify the signaling.
- ✓ If you do not have an oscilloscope at your disposal, the values can be modified for testing with a speaker or LED. For example, us := clkfreq/1_000_000 can be changed to ms := clkfreq/1_000_000 and all instances of us can be replaced with ms. Also reduce 38000 to 3800 for playing on a speaker, or 38 for displaying with an LED.

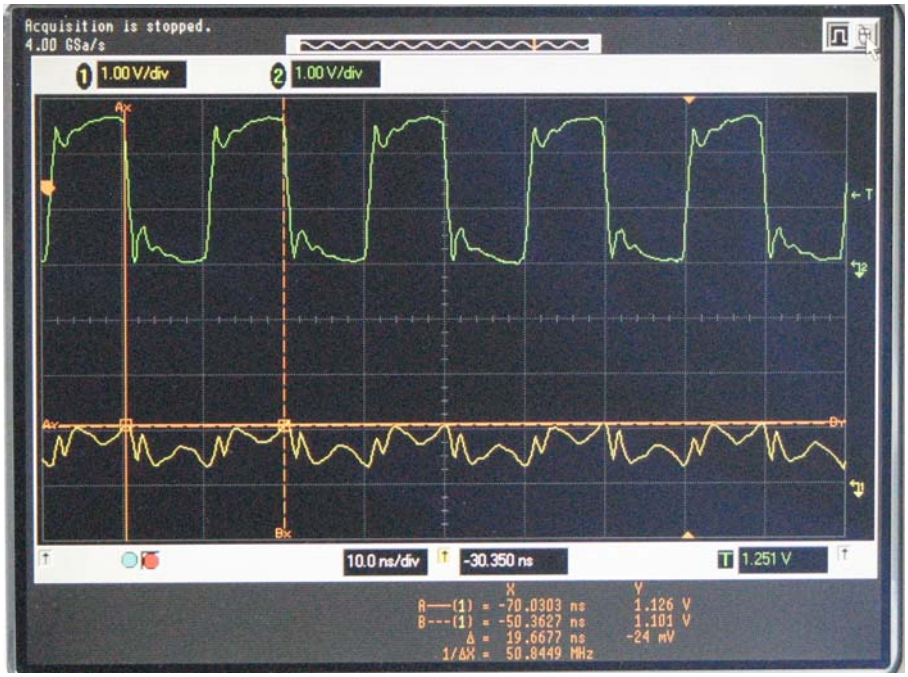
```
''Test Modulate IR Signals.spin
''Use SquareWave.spin to send pulses of modulated infrared with a carrier frequency
''of 38 kHz.

CON

  _clkmode = xtal1 + pll16x                     ' Crystal and PLL settings
  _xinfreq = 5_000_000                          ' 5 MHz crystal

OBJ

  sqw    : "SquareWave"                         ' Square wave object

PUB go | us, index, t

  us := clkfreq/1_000_000

  '  outa[pin]~~ holds P14 high.  The 38 kHz signal will only transmit when
  '  outa[pin] is low.

  outa[14]~~                                    ' Hold P14 high
  sqw.Freq(14, 0, 38000)                        ' Prepeare to send signal
  waitcnt(600*us + cnt)                         ' Don't do anything for 0.6 ms

  repeat index from 1 to 3                      ' Indexed loop

    ' First rep, t = 2400 µs, second rep, t = 600 µs, third rep, t = 1200 µs
    t := lookup(index: 2400*us, 600*us, 1200*us)

    outa[14]~                                   ' Start transmitting square wave
    waitcnt(t + cnt)                            ' Wait for t
    outa[14]~~                                  ' Stop transmitting square wave
    waitcnt(600*us + cnt)                       ' Wait for 600 µs
```

Square Wave Applications – up to 128 MHz

The SquareWave object can transmit square waves up to 128 Mhz. These signals can provide clock signals for high speed peripheral devices, carrier frequencies for transmitting data over radio frequencies, and sensor applications like metal detection. Designs at these frequencies delve into the RF realm of filtering, maximum transmitter power rules, antenna design and so on.

The upper trace in Figure 7 shows a 50.85 Mhz test signal displayed by a high speed oscilloscope, measured with low impedance, active probes. The lower trace is the step response if a notch filter tuned to approximately 50 MHz. The shape and maximum height of this step response changes with frequency, and the PE Kit Labs uses these signals to detect the presence or absence of a coin below a loop of wire.

Figure 7: 50.85 MHz test signal

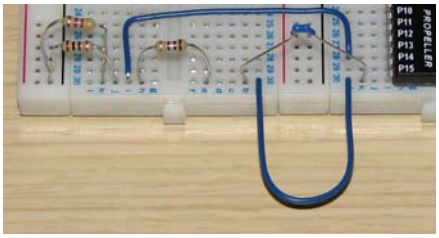


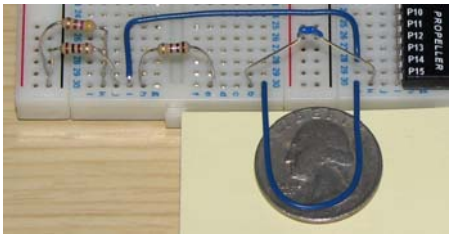
The interesting thing about the code for sending the signal shown in Figure 7 is that it's about the same as the rest of the example programs. The SquareWave does configure the counter module differently for signals faster than 500 kHz, but from the user standpoint, it's still just a matter of passing the pin, channel number and frequency to the SquareWave object's Freq method.

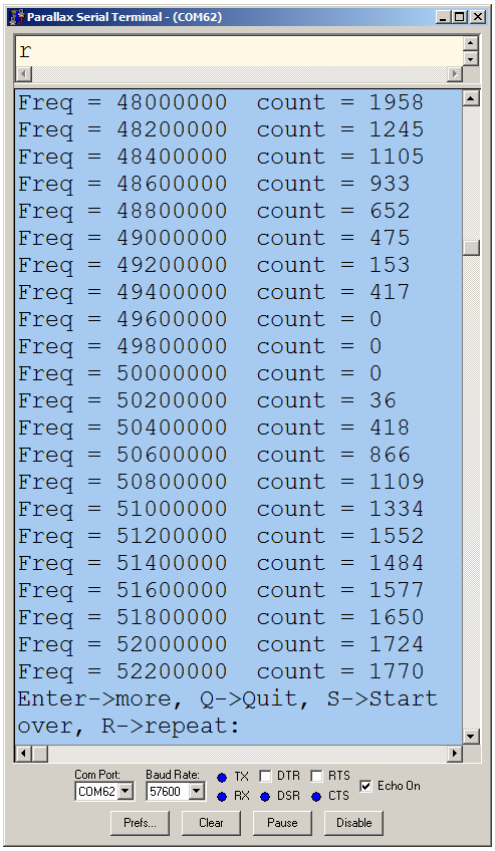
```
''Test High Freq Square Wave.spin
''Send a 50.85 Mhz signal to P15 for 0.1 ms.

CON

  _clkmode = xtal1 + pll16x                     ' Crystal and PLL settings
  _xinfreq = 5_000_000                          ' 5 MHz crystal

OBJ

  sqw    : "SquareWave"                         ' Square wave object

PUB go

  sqw.Freq(15, 0, 50_850_000)                   ' Send 50.85 MHz into notch filter
  waitcnt(clkfreq/10000+cnt)                    ' Wait 0.1 ms
  sqw.End(15, 0, 0)                            ' Stop signal
```

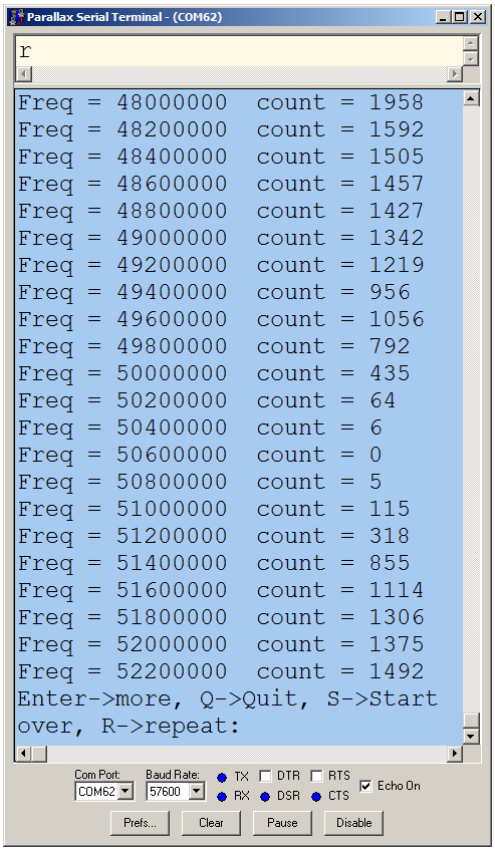
The metal detector application featured in Propeller Education Kit Labs: Fundamentals demonstrates how to use an older revision of the SquareWave object to briefly send a variety of square waves through a filter. The application uses a second counter module to monitor the filter's response. Figure 8 shows how the filter responds differently with and without the coin below the loop of metal.

Figure 8: Metal Detector Application*









* Source: *Propeller Education Kit Labs: Fundamentals*

More Information

For "under the hood" information about how the SquareWave object configures a cog's counter modules, read 7: Counter Modules and Circuit Applications Lab in *Propeller Education Kit Lab: Fundamentals*. Additional information on counter modules can be found in the *Propeller Manual*, and in *AN001 – Propeller Counters (.pdf)*.