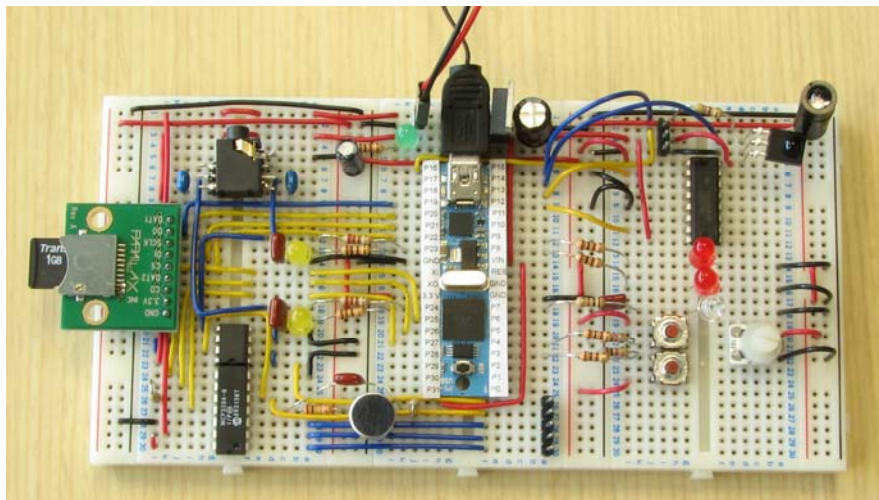


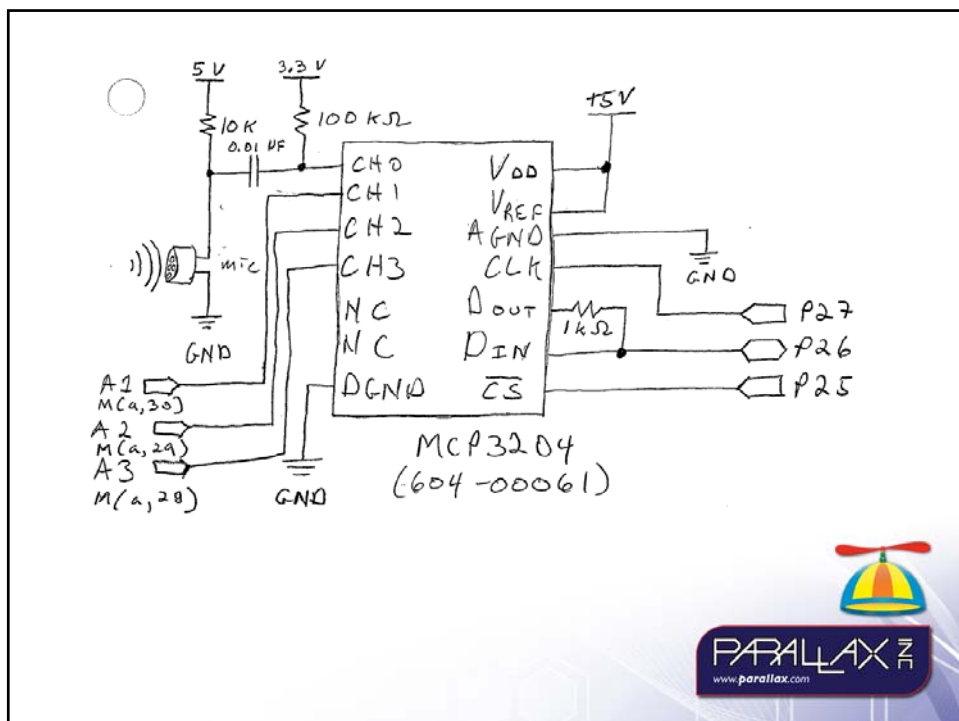
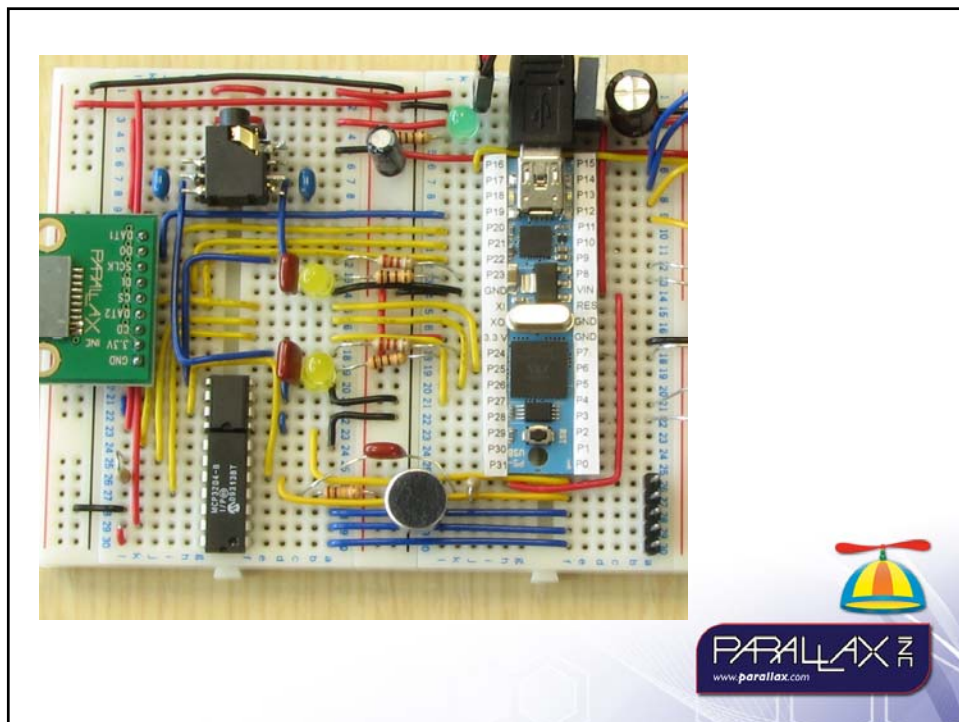
Spin Tip

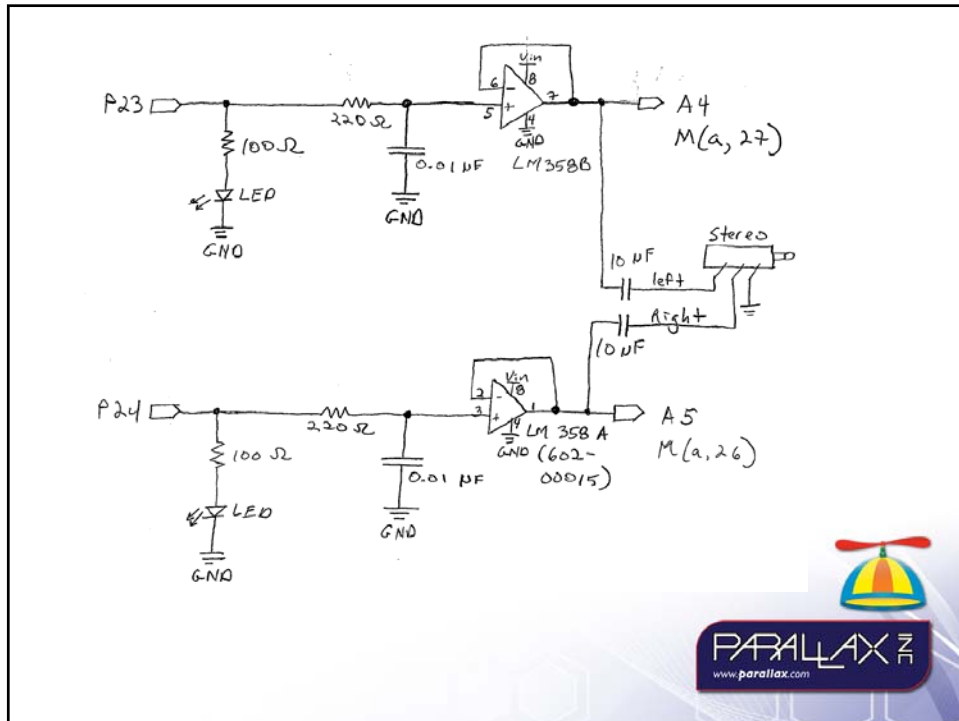
A/D and D/A for the Classroom



Andy's Prototyping Board







Extended Spin Tip from:
Propeller Education Kit Labs: Fundamentals

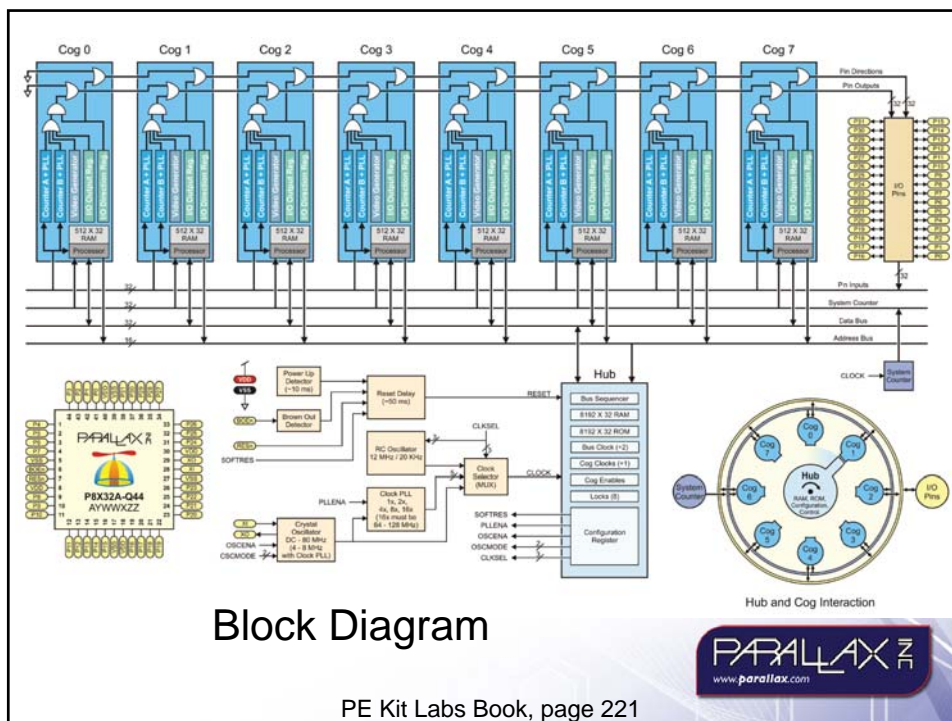
Lab 7

Counter Modules & Circuit Applications



Counter Module App Examples

- Parallax.com/Propeller
 - Propeller Q&A
- parallax.com/go/PEKit
 - Propeller <-> PC Communication
 - Measure Resistance and Capacitance
 - Transmit Square Wave Frequencies
 - Servo Control



Lab 7: Counters Modules...

What's a Counter Module?

Introduction

Each Propeller cog has two *counter modules*, and each counter module can be configured to independently perform repetitive tasks. So, not only does the Propeller chip have the ability to execute code simultaneously in separate cogs, each cog can also orchestrate up to two additional processes with counter modules while the cog continues executing program commands. Counters can provide a cog with a variety of services; here are some examples:

- Measure pulse and decay durations
- Count signal cycles and measure frequency
- Send numerically-controlled oscillator (NCO) signals, i.e. square waves
- Send phase-locked loop (PLL) signals, which can be useful for higher frequency square waves
- Signal edge detection
- Digital to analog (D/A) conversion
- Analog to digital (A/D) conversion
- Provide internal signals for video generation



From PE Kit Labs, page 121

Lab 7: Counter Modules...

How do They Work?

How Counter Modules Work

Each cog has two counter modules, Counter A and Counter B. Each cog also has three 32-bit special purpose registers for each of its counter modules. The Counter A special purpose registers are *phsa*, *frqa*, *ctra*, and Counter B's are *phsb*, *frqb* and *ctrb*. Note that each counter name is also a reserved word in Spin and Propeller assembly. If this lab is referring to a register generally, but it doesn't matter whether it's for Counter A or Counter B, it will use the generic names PHS, FRQ, and CTR.

Here is how each of the three registers works in a counter module:

- PHS – the “phase” register gets updated every clock tick. A counter module can also be configured make certain PHS register bits affect certain I/O pins.
- FRQ – the “frequency” register gets conditionally added to the PHS register every clock tick. The counter module's mode determines what conditions cause FRQ to get added to PHS. Mode options include “always”, “never”, and conditional options based on I/O pin states or transitions.
- CTR – the “control” register configures both the counter module's mode and the I/O pin(s) that get monitored and/or controlled by the counter module. Each counter module has 32 different modes, and depending on the mode, can monitor and/or control up to two I/O pins.



From PE Kit Labs, page 122

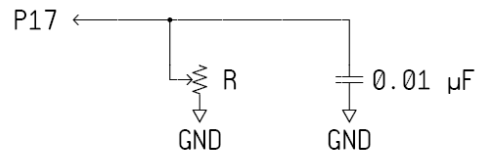
Lab 7: Counter Modules...

Circuit Application: Measure RC Decay

Parts List

- (1) Potentiometer 10 k Ω
- (1) Capacitor - 0.01 μ F
- (misc) Jumper wires

Schematic

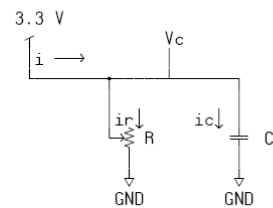


From PE Kit Labs, page 122

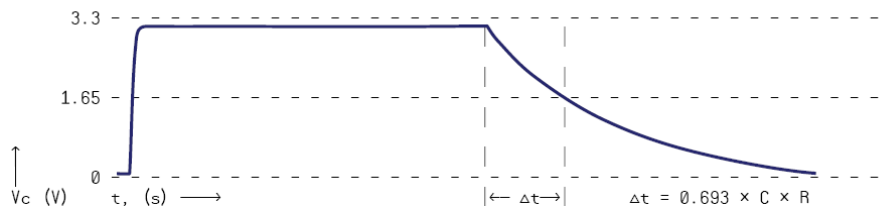
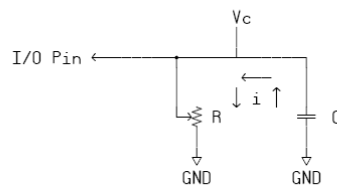
Lab 7: Counter Modules...

RC Decay Circuit Steps

Charge Circuit
(I/O pin = output-high)



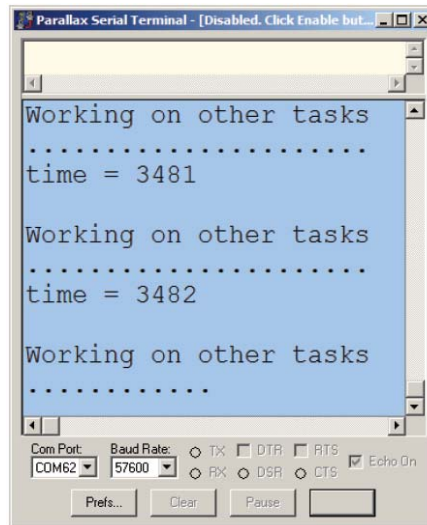
Decay Circuit
(I/O pin = input)



From PE Kit Labs, page 123

Lab 7: Counter Modules...

Doing other Things While Counter Measures RC Decay



From PE Kit Labs, page 127



Lab 7: Counter Modules...

CTRMODE	Description	Accumulate FRQ to PHS	APIN output*	BPIN output*
%00000	Counter disabled (off)	0 (never)	0 (none)	0 (none)
...				
%01000	POS detector	A^1	0	0
%01001	POS detector w/feedback	A^1	0	$!A^1$
%01010	POSEDGE detector	$A^1 \& !A^2$	0	0
%01011	POSEDGE detector w/feedback	$A^1 \& !A^2$	0	$!A^1$
...				
%11111	LOGIC always	1	0	0

* must set corresponding DIR bit to affect pin

A^1 = APIN input delayed by 1 clock
 A^2 = APIN input delayed by 2 clocks
 B^1 = BPIN input delayed by 1 clock

bits	31	30..26	25..23	22..15	14..9	8..6	5..0
Name	—	CTRMODE	PLLDIV	—	BPIN	—	APIN

From PE Kit Labs, page 124



Lab 7: Counter Modules...

Counter Setup Steps for RC Decay

- 1) Store %01000 in the CTR register's CTRMODE bit field:

```
ctr[30..26] := %01000
```

- 2) Store the I/O pin number that you want monitored in the CTR register's APIN bit field:

```
ctr[5..0] := 17
```

- 3) Store 1 in the FRQ register so that the `phsa` register will get 1 added to it for every clock tick that P17 is high:

```
frqa := 1
```



From PE Kit Labs, page 125

Lab 7: Counter Modules...

from TestRcDecay.spin

```
' Charge RC circuit.
dira[17] := outa[17] := 1          ' Set pin to output-high
waitcnt(clkfreq/100_000 + cnt)     ' Wait for circuit to charge

' Start RC decay measurement.  It's automatic after this...

phsa~                               ' Clear the phsa register
dira[17]~                           ' Pin to input stops charging circuit

' Optional - do other things during the measurement.
Debug.str(String(CR, CR, "Working on other tasks", CR))
repeat 22
  Debug.tx(".")
  waitcnt(clkfreq/60 + cnt)

' Measurement has been ready for a while.  Adjust ticks between phsa~ & dira[17]~.
time := (phsa - 624) #> 0

' Display Result
Debug.Str(String(13, "time = "))
Debug.Dec(time)
waitcnt(clkfreq/2 + cnt)
```



From PE Kit Labs, page

Lab 7: Counter Modules...

NCO Mode – for Square Waves & Pulses

Parts List

(2) Piezospeakers
(misc) Jumper wires

Schematic

Piezospakers



$$\text{FRQ register} = \text{PHS bit 31 frequency} \times \frac{2^{32}}{\text{clkfreq}}$$

$$\text{PHS bit 31 frequency} = \frac{\text{clkfreq} \times \text{FRQ register}}{2^{32}}$$



From PE Kit Labs, page 137, 138

Lab 7: Counter Modules...

Example: Calculating FRQ Register Values

What value does `frqa` have to store to make the counter module transmit a 2093 Hz square wave if the system clock is running at 80 MHz? (If this were a sine wave, it would be a C7, a C note in the 7th octave.)

For the solution, start with Eq. 2. Substitute 80,000,000 for `clkfreq` and 2093 for `frequency`.

$$\text{frqa} = 2,093 \times 2^{32} \div 80,000,000$$

$$\text{frqa} = 2,093 \times 53.687$$

$$\text{frqa} = 112,367$$

Table 7-1: Notes, Frequencies, and FRQA/B Register Values for 80 MHz

Note	Frequency (Hz)	FRQA/B Register	Note	Frequency (Hz)	FRQA/B Register
C6	1046.5	56_184	G6	1568.0	84_181
C6#	1107.8		G6#	1661.2	
D6	1174.7	63_066	A6	1760.0	94_489
D6#	1244.5		A6#	1864.7	
E6	1318.5	70_786	B6	1975.5	105_629
F6	1396.9	74_995	C7	2093.0	112_367
F6#	1480.0				



From PE Kit Labs, page 138

Lab 7: Counter Modules...

NCO CTRMODE

CTRMODE	Description	Accumulate FRQ to PHS	APIN output*	BPIN output*
%00000	Counter disabled (off)	0 (never)	0 (none)	0 (none)
.
%00100	NCO/PWM single-ended	1	PHS[31]	0
%00101	NCO/PWM differential	1	PHS[31]	!PHS[31]
.
%11111	LOGIC always	1	0	0



From PE Kit Labs, page 139

Lab 7: Counter Modules...

How to Set Up NCO Mode

Here are the steps for configuring a counter module to NCO mode:

- (1) Configure the CTRA/B register
- (2) Set the FRQA/B register
- (3) Set the I/O pin to output

(1) *Configure the CTRA/B register:* Here is an example that sets Counter A to “NCO single-ended” mode, with the signal transmitted on P27. To do this, set `ctra[30..26]` to `%00100`, and `ctra[5..0]` to 27.

```
ctra[30..26] := %00100
ctra[5..0] := 27
```

(2) *Set the FRQA/B register:* Here is an example for the square wave version of the C7 note:

```
frqa := 112_367
```

(3) *Set the I/O pin to output:* Since it's P27 that's sending the signal, make it an output:

```
dira[27]~~
```



From PE Kit Labs, page 139

Lab 7: Counter Modules...

Square Wave Test

```

''SquareWaveTest.spin
''Send 2093 Hz square wave to P27 for 1 s with counter module.

CON

    _clkmode = xtal1 + pll16x          ' Set up clkfreq = 80 MHz.
    _xinfreq = 5_000_000

PUB TestFrequency

    'Configure ctra module
    ctra[30..26] := %00100            ' Set ctra for "NCO single-ended"
    ctra[5..0] := 27                  ' Set APIN to P27
    frqa := 112_367                   ' Set frqa for 2093 Hz (C7 note) using:
                                      ' FRQA/B = frequency × (232 ÷ clkfreq)

    'Broadcast the signal for 1 s
    dira[27]~~                        ' Set P27 to output
    waitcnt(clkfreq + cnt)            ' Wait for tone to play for 1 s

```



From PE Kit Labs, page 140

Lab 7: Counter Modules...

Stacatto - Example of CTR (OR) Output

```

''Staccato.spin
''Send 2093 Hz beeps in rapid succession (15 Hz for 1 s).

CON

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

PUB TestFrequency

    'Configure ctra module
    ctra[30..26] := %00100            ' Set ctra for "NCO single-ended"
    ctra[5..0] := 27                  ' Set APIN to P27
    frqa := 112_367                   ' Set frqa for 2093 Hz (C7 note):

    'Ten beeps on/off cycles in 1 second.
    repeat 30
        !dira[27]                    ' Set P27 to output
        waitcnt(clkfreq/30 + cnt)    ' Wait for tone to play for 1 s

    'Program ends, which also stops the counter module.

```



From PE Kit Labs, page 141

Lab 7: Counter Modules...

Musical Notes

```

DoReMi.spin
Play C6, D6, E6, F6, G6, A6, B6, C7 as quarter notes quarter stops between.

CON
_clkmode = xtal1 + pll16x          ' System clock = 80 MHz
_winfreq = 5_000_000

PUB TestFrequency | index
'Configure ctra module
ctr[30..26] := x00100              ' Set ctra for "NCO single-ended"
ctr[5..0] := 27                    ' Set APIN to P27
freq := 0                          ' Don't play any notes yet

repeat index from 0 to 7

    freq := long[notes][index]      ' Set the frequency.
    'Broadcast the signal for 1/4 s
    dira[27]~
    waitcnt(clkfreq/4 + cnt)        ' Set P27 to output
    'Wait for tone to play for 1/4 s

    dira[27]~
    waitcnt(clkfreq/4 + cnt)        ' 1/4 s stop

DAT
'80 MHz freq values for square wave musical note approximations with the counter module
'configured to NCO:
      C6      D6      E6      F6      G6      A6      B6      C7
notes long 56_184, 63_066, 70_786, 74_995, 84_181, 94_489, 105_629, 112_528

```



From PE Kit Labs, page 142

Lab 7: Counter Modules...

Inside NCO Mode

Counter NCO Mode Example with bit 3 Instead of bit 31

In NCO mode, the I/O pin's output state is controlled by bit 31 of the PHS register. However, the on/off frequency for any bit in a variable or register can be calculated using Eq. 4 and assuming a value is repeatedly added to it at a given rate:

$$\text{frequency} = (\text{value} \times \text{rate}) \div 2^{\text{bit} + 1} \quad \text{Eq. 4}$$

Next is an example that can be done on scratch paper that may help clarify how this works.

Bit 3 Example: At what frequency does bit 3 in a variable toggle if you add 4 to it eight times every second? Here, **value** is 4, **rate** is 8 Hz, and **bit** is 3, so

$$\begin{aligned}
 \text{frequency} &= (\text{value} \times \text{rate}) \div 2^{\text{bit} + 1} \\
 &= (4 \times 8 \text{ Hz}) \div 23 + 1 \\
 &= 32 \text{ Hz} \div 16 \\
 &= 2 \text{ Hz}
 \end{aligned}$$



From PE Kit Labs, page 142

Lab 7: Counter Modules...

Inside NCO Mode

Table 7-2: Bit 3 Example

Time (s)	Value	Variable	Bit 3 in Variable							
			7	6	5	4	3	2	1	0
0.000		0	0	0	0	0	0	0	0	0
0.125	4	4	0	0	0	0	0	1	0	0
0.250	4	8	0	0	0	0	1	0	0	0
0.375	4	12	0	0	0	0	1	1	0	0
0.500	4	16	0	0	0	1	0	0	0	0
0.625	4	20	0	0	0	1	0	1	0	0
0.750	4	24	0	0	0	1	1	0	0	0
0.875	4	28	0	0	0	1	1	1	0	0
1.000	4	32	0	0	1	0	0	0	0	0
1.125	4	36	0	0	1	0	0	1	0	0
1.250	4	40	0	0	1	0	1	0	0	0
1.375	4	44	0	0	1	0	1	1	0	0
1.500	4	48	0	0	1	1	0	0	0	0
1.625	4	52	0	0	1	1	0	1	0	0
1.750	4	56	0	0	1	1	1	0	0	0
1.875	4	60	0	0	1	1	1	1	0	0

From PE Kit Labs, page 142



Lab 7: Counter Modules...

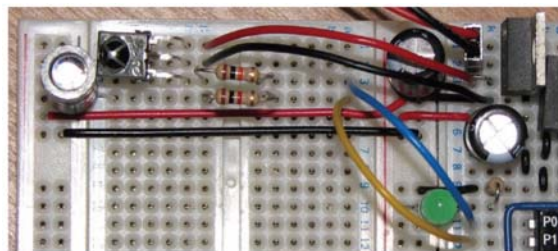
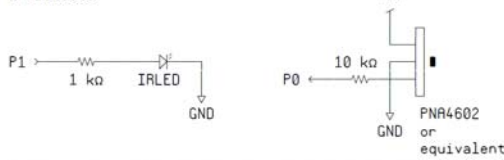
NCO for IR Object Detection

Parts List

- (1) Resistor 1 kΩ
- (1) Resistor 10 kΩ
- (1) IR LED
- (1) IR detector
- (1) LED shield
- (1) LED standoff
- (misc) Jumper wires

Schematic

IR Detector



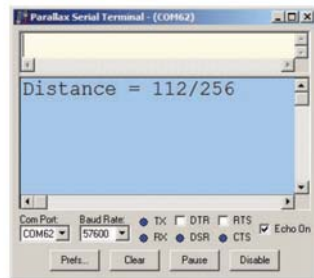
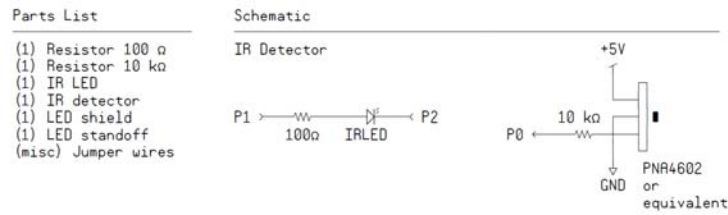
IR Object Detection.spin



From PE Kit Labs, page 146

Lab 7: Counter Modules...

NCO for IR Distance Detection



TestIrDutyDistanceDetector.spin



From PE Kit Labs, page 148-150

Lab 7: Counter Modules...

Transition Counting

CTRMODE	Description	Accumulate FRQ to PHS	APIN output*	BPIN output*
%00000	Counter disabled (off)	0 (never)	0 (none)	0 (none)
%01010	POSEDGE detector	$A^1 \& !A^2$	0	0
%01110	NEGEDGE detector	$!A^1 \& A^2$	0	0
%11111	LOGIC always	1	0	0

CountEdgeTest.spin
BetterCountEdges.spin



From PE Kit Labs, page 151-153

Lab 7: Counter Modules...

PWM with NCO mode

```
''SinglePulseWithCounter.spin
''Send a high pulse to the P4 LED that lasts exactly 80_000_000 clock ticks.

CON

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

PUB TestPwm | tc, tHa, tHb, ti, t

  ctra[30..26] := %00100            ' Configure Counter A to NCO
  ctra[8..0] := 4
  frqa := 1
  dira[4]~~~

  phsa := - clkfreq                  ' Send the pulse

  ' Keep the program running so the pulse has time to finish.
  repeat
```



From PE Kit Labs, page 155

Lab 7: Counter Modules...

Duty Cycle Signal with NCO

```
''1Hz25PercentDutyCycle.spin
''Send 1 Hz signal at 25 % duty cycle to P4 LED.

CON

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

PUB TestPwm | tc, tHa, t

  ctra[30..26] := %00100            ' Configure Counter A to NCO
  ctra[8..0] := 4
  frqa := 1
  dira[4]~~~

  tC := clkfreq                      ' Set up cycle and high times
  tHa := clkfreq/4
  t := cnt                            ' Mark counter time
```

1Hz25PercentDutyCycleDiffSig.spin

TestDualPWM.spin

SinglePwmWithTimeIncrements



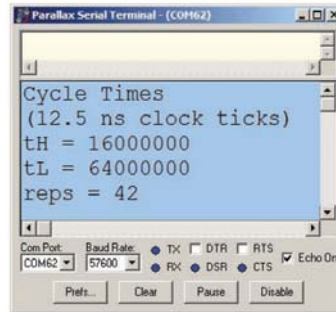
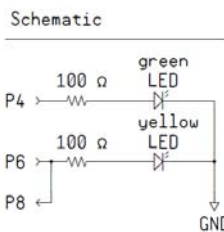
From PE Kit Labs, page 155-157

Lab 7: Counter Modules...

Application: Probe & Display PWM Signal

Figure 7-20: Use P8 to Measure PWM Signal from P6

Parts List
(2) Resistors 100 Ω
(1) LED - green
(1) LED - yellow
(misc) Jumper wires



TestDualPwmWithProbes.spin
MonitorPWM.spin



From PE Kit Labs, page 158-160

Lab 7: Counter Modules...

PLL Modes for High Frequency Applications

CTRMODE	Description	Accumulate FRQ to PHS	APIN output*	BPIN output*
%00000	Counter disabled (off)	0 (never)	0 (none)	0 (none)
...				
%00001	PLL internal (video mode)	1 (always)	0	0
%00010	PLL single-ended	1	PLL	0
%00011	PLL differential	1	PLL	!PLL
...				
%11111	LOGIC always	1	0	0

bits	31	30..26	25..23	22..15	14..9	8..6	5..0
Name	—	CTRMODE	PLLDIV	—	BPIN	—	APIN



From PE Kit Labs, page 165

Lab 7: Counter Modules...

Calculate Frequency from FRQ & PLLDIV

- (1) Calculate the PHS bit 31 frequency:

$$\text{PHS bit 31 frequency} = \frac{\text{clkfreq} \times \text{FRQ register}}{2^{32}}$$

- (2) Use the PHS bit 31 frequency to calculate the VCO frequency:

$$\text{VCO frequency} = 16 \times \text{PHS bit 31 frequency}$$

- (3) Divide the PLLDIV result, which is $2^{7-\text{PLLDIV}}$ into the VCO frequency:

$$\text{PLL frequency} = \frac{\text{VCO frequency}}{2^{7-\text{PLLDIV}}}$$



PARALLAX
www.parallax.com

From PE Kit Labs, page 166

Lab 7: Counter Modules...

Calculate Frequency from FRQ & PLLDIV

Example: Given a system clock frequency (**clkfreq**) of 80 MHz and the code below, calculate the PLL frequency transmitted on I/O Pin P15.

```
'Configure ctra module
ctr[30..26] := %00010
freq := 322_122_547
ctr[25..23] := 2
ctr[5..0] := 15
dira[15]--
```

- (1) Calculate the PHS bit 31 frequency:

$$\begin{aligned} \text{PHS bit 31 frequency} &= \frac{80_000_000 \times 322_122_547}{2^{32}} \\ &= 5_999_999 \end{aligned}$$

- (2) Use the PHS bit 31 frequency to calculate the VCO frequency:

$$\begin{aligned} \text{VCO frequency} &= 16 \times 5_999_999 \\ &= 95_999_984 \end{aligned}$$

- (3) Divide the PLLDIV result ($2^{7-\text{PLLDIV}}$) into the VCO frequency:

$$\begin{aligned} \text{PLL frequency} &= \frac{95_999_984}{2^{7-2}} \\ &= 2_999_999 \text{ MHz} \\ &\approx 3 \text{ MHz} \end{aligned}$$



PARALLAX
www.parallax.com

From PE Kit Labs, page 166

Lab 7: Counter Modules...

Need a Frequency, Calculate FRQ & PLLDIV

- (1) Use the table below to figure out which value to put in the CTR register's PLLDIV bit field based on the frequency you want to transmit.

MHz	PLLDIV	MHz	PLLDIV
0.5 to 1	0	8 to 16	4
1 to 2	1	16 to 32	5
2 to 4	2	32 to 64	6
4 to 8	3	64 to 128	7

- (2) Calculate the VCO frequency with the PLL frequency you want to transmit and the PLL divider, and round down to the next lowest integer.

$$\text{VCO frequency} = \text{PLL frequency} \times 2^{(7-\text{PLLDIV})}$$

- (3) Calculate the PHS bit 31 frequency you'll need for the VCO frequency. It's the VCO frequency divided by 16.

$$\text{PHS bit 31 frequency} = \text{VCO frequency} \div 16$$

- (4) Use the NCO frequency calculations to figure out the FRQ register value for the PHS bit 31 frequency.

$$\text{FRQ register} = \text{PHS bit 31 frequency} \times \frac{2^{32}}{\text{clkfreq}}$$

From PE Kit Labs, page 167

Lab 7: Counter Modules...

FRQ & PLLDIV Calculation Example

Example: `clkfreq` is running at 80 MHz, and you want to generate a 12 MHz signal with PLL. Figure out the FRQ register and PLLDIV bit fields.

- (1) Use the table to figure out which value to put in the CTR register's PLLDIV bit field:
Since 12 MHz falls in the 4 to 16 MHz range, PLLDIV is 4.7. Round down, and use 4.

- (2) Calculate the VCO frequency with the final PLL frequency and the PLL divider:

$$\begin{aligned} \text{VCO frequency} &= 12 \text{ MHz} \times 2^{(7-4)} \\ &= 12 \text{ MHz} \times 8 \\ &= 96 \text{ MHz} \end{aligned}$$

- (3) Calculate the PHS bit 31 frequency you'll need for the VCO frequency. It's the VCO frequency divided by 16:

$$\begin{aligned} \text{PHS bit 31 frequency} &= 96 \text{ MHz} \div 16 \\ &= 6 \text{ MHz} \end{aligned}$$

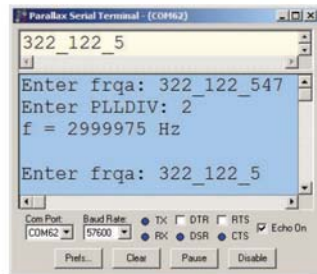
- (4) Use the NCO frequency calculations to figure out the FRQ register value for the PHS bit 31 frequency:

$$\begin{aligned} \text{FRQ register} &= 6 \text{ MHz} \times \frac{2^{32}}{80 \text{ MHz}} \\ &= 322_122_547 \end{aligned}$$

From PE Kit Labs, page 167

Lab 7: Counter Modules...

Test PLL Frequencies



TestPLLParameters.spin

SquareWave.spin



From PE Kit Labs, page 168-169

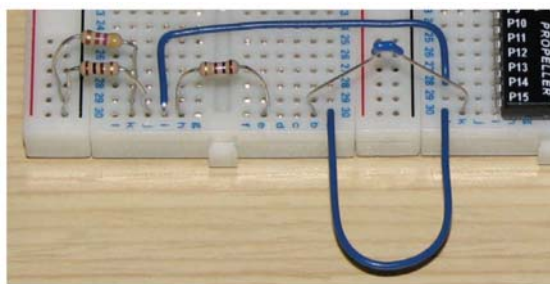
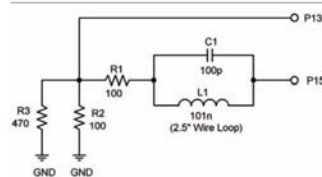
Lab 7: Counter Modules...

Metal Detection

Parts List

- (1) Capacitor 10 pF
- (2) Jumper Wires
- (2) Resistors 100 Ω
- (misc) resistors:
220, 470, 1000,
2000, 10k

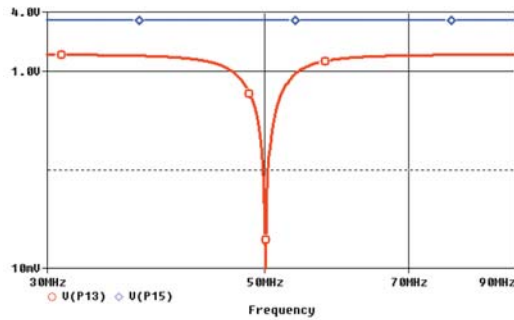
Schematic



From PE Kit Labs, page

Lab 7: Counter Modules...

Notch Filter Resonant Frequency



$$f_R = \frac{1}{2\pi\sqrt{LC}}$$

$$L = \frac{1}{(2\pi f_R)^2 C}$$

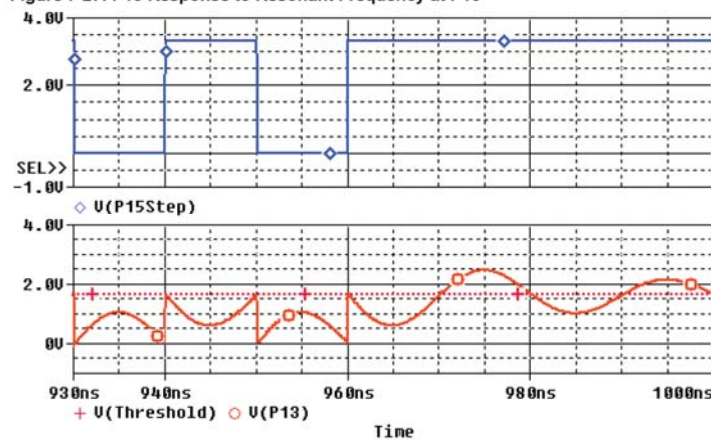


From PE Kit Labs, page 171

Lab 7: Counter Modules...

Step Response of Notch Filter

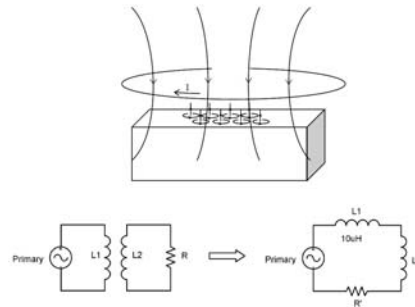
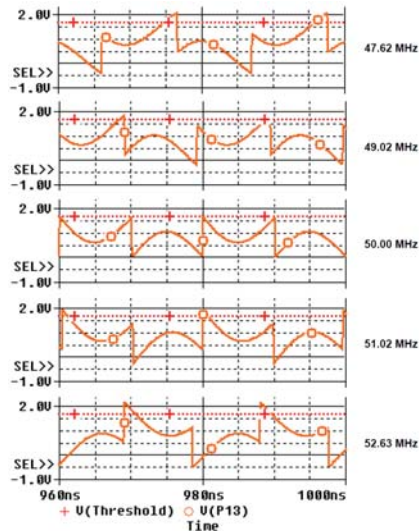
Figure 7-27: P13 Response to Resonant Frequency at P15



From PE Kit Labs, page 172

Lab 7: Counter Modules...

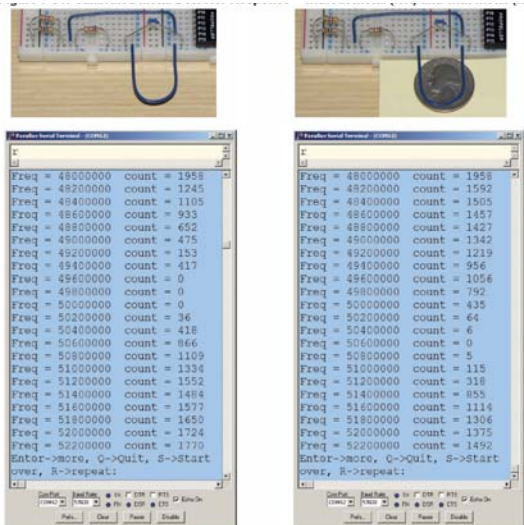
Eddie Currents and Reflected Impedance



From PE Kit Labs, page 173, 174

Lab 7: Counter Modules...

Metal Detection Application – Measure Frequency Response

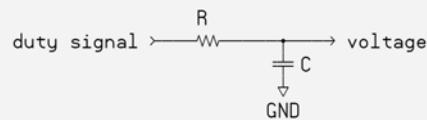


From PE Kit Labs, page 175 - 176

Lab 7: Counter Modules...

D/A Conversion

Time Varying D/A and Filtering: When modulating the value of $frqa$ to send time varying signals, an RC circuit typically filters the duty signal. It's better to use a smaller fraction of the useable duty signal range, say 25% to 75% or 12.5% to 87.5%. By keeping the duty in this middle range, the D/A will be less noisy and smaller resistor R and capacitor C values can be used for faster responses. This is especially important for signals that vary quickly, like audio signals, which will be introduced in a different lab.



From PE Kit Labs, page 132

Lab 7: Counter Modules...

Duty Modes from CTR.spin

CTRMODE	Description	Accumulate FRQ to PHS	APIN output*	BPIN output*
%00000	Counter disabled (off)	0 (never)	0 (none)	0 (none)
...				
%00110	DUTY single-ended	1	PHS-Carry	0
%00111	DUTY differential	1	PHS-Carry	!PHS-Carry
...				
%11111	LOGIC always	1	0	0

bits	31	30..26	25..23	22..15	14..9	8..6	5..0
Name	—	CTRMODE	PLLDIV	—	BPIN	—	APIN



From PE Kit Labs, page 131

Lab 7: Counter Modules...

Steps for Setting up a Duty Mode Signal

Setting up a Duty Signal

Here are the steps for setting a duty signal either with a counter:

- (1) Set the CTR register's CTRMODE bit field to choose duty mode.
- (2) Set the CTR register's APIN bit field to choose the pin.
- (3) If you are using differential DUTY mode, set the CTR register's BPIN field.
- (4) Set the I/O pin(s) to output.
- (5) Set the FRQ register to a value that gives you the percent duty signal you want.



From PE Kit Labs, page 131

Lab 7: Counter Modules...

Duty Mode Example

Example – Send a 25% single-ended duty signal to P4 Using Counter A.

(1) Set the CTR register's CTRMODE bit field to choose a DUTY mode. Remember that bits 30..26 of the CTR register (shown in Figure 7-9) have to be set to the bit pattern selected from the CTRMODE list in Figure 7-8. For example, here's a command that configures the counter module to operate in single-ended DUTY mode:

```
ctr[30..26] := %00110
```

(2) Set the CTR register's APIN bit field to choose the pin. Figure 7-9 indicates that APIN is bits 5..0 in the CTR register. Here's an example that sets the ctr register's APIN bits to 4, which will control the green LED connected to P4.

```
ctr[5..0] := 4
```

We'll skip step (3) since the counter module is getting configured to single-ended DUTY mode and move on to:

(4) Set the I/O pin(s) to output.

```
dira[4] := 1
```

(5) Set the FRQ register to a value that gives you the duty signal you want. For ¼ brightness, use 25% duty. So, set the frqa register to 1_073_741_824 (calculated earlier).

```
frqa := 1_073_741_824
```



From PE Kit Labs, page 132

Lab 7: Counter Modules...

Sweep D/A Output

```

''LedsweepWithSpr.spin
''Cycle P4 and P5 LEDs through off, gradually brighter, brightest at different rates.
CON

scale = 16_777_216          ' 2^24 ÷ 256

PUB TestDuty | apin, duty[2], module

'Configure both counter modules with a repeat loop that indexes SPR elements.

repeat module from 0 to 1          ' 0 is A module, 1 is B.
  apin := lookupz (module: 4, 6)
  spr[8 + module] := (x00110 << 26) + apin
  dira[apin]~~

'Repeat duty sweep indefinitely.

repeat
  repeat duty from 0 to 255          ' Sweep duty from 0 to 255
    duty[1] := duty[0] * 2          ' duty[1] twice as fast
    repeat module from 0 to 1
      spr[10 + module] := duty[module] * scale ' Update frqa register
      waitcnt(clkfreq/128 + cnt)          ' Delay for 1/128th s

```



From PE Kit Labs, page 136

Lab 7: Counter Modules...

How Duty Mode Works (3-bit example)

$$\text{duty} = \frac{\text{pin high time}}{\text{time}} = \frac{\text{frq}}{8} \quad (\text{3-bit example})$$

carry flag set

		1 1	1 1	1 1		1 1 1	1	1 1 1
3-bit frq	011	011	011	011	011	011	011	011
3-bit phs (previous)	+000	+011	+110	+001	+100	+111	+010	+101
3-bit phs (result)	011	110	001	100	111	010	101	000



From PE Kit Labs, page 134