Propeller Microcontroller
And the Propeller Education Kit
Why learn microcontrollers?

Because they are the “onboard computers” built into devices that interact with their circuits. Examples include:

- Digital alarm clock
- Microwave oven
- Alarm system
- Cell phone
- Automobile
- Computer keyboard
- Heart monitor
- Factory equipment subsystems
- Etc…
The Propeller Microcontroller?

- 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
Why the Propeller Microcontroller?

Conventional approach to multitasking with a single core microcontroller.

Single processor alternates between tasks.
Why the Propeller Microcontroller?

Multicore Propeller approach.

Read Propeller Education Kit Labs: Fundamentals, chapter 1 to find out more.
Propeller Education Kit (PE kit)
40-Pin DIP Version

Propeller Education Kit – 40 Pin DIP
PE Kit Labs: Fundamentals Book

- For those new to the Propeller
- Guided tutorial
- Accompanies both versions of the PE Kit
- Prerequisites
  - Familiarity with electricity & electronics
  - Familiarity with a high level programming language (C++, Java, BASIC, Pascal, Python, etc.)
  - OR complete What’s a Microcontroller
Programming Languages

• Native high level Spin language and low level Propeller Assembly Language have the best documentation support.
• Many other languages are available, including C, BASIC, Forth, Java, and more…

Propeller Tool Software Help

• Get Started: Propeller Education Kit Labs: Fundamentals
• Reference: Propeller Manual
• Objects: Propeller Object Exchange – obex.parallax.com
• Propeller Forums: forums.parallax.com -> Propeller
• Coding Aid: Propeller Chip Quick Reference
Chapter 1
Propeller Microcontroller & PE Kit Labs Overview
A Cog Interpreting Spin

(a) Interpreter loaded into cog from Main Memory’s ROM through Hub

(b) Cog fetches token from Main Memory’s RAM

(c) Cog executes token. Examples include RAM, I/O or config read/write, or ROM read

From PE Kit Labs, Ch 1
A Cog Executing ASM

From PE Kit Labs, Ch 1
Cogs Communicating

Main (Hub) Memory

- Configuration
- Application
- Stack + VAR

ROM
- Character Set
- Log, Antilog, & Sine Tables
- Boot Loader Interpreter

RAM

COG

COG
Cogs Launching Cogs
Application Design

- Top Object File
  - Following Robot.spin
    - Ir Detector.spin
    - PID Algorithm.spin
      - Launches a cog
        - Spin code only
        - Servo Control.spin
    - Square Wave.spin
      - Launches a cog
        - Spin + ASM
          - Float32.spin
          - FloatString.spin
Program Loading & Storage
Propeller Education Kit Labs: Fundamentals

Chapter 2
Software, Documentation, and Resources
Follow these Instructions!

**Download Software and Documentation**

In these labs, you will make use of the Propeller Tool programming software, the Parallax Serial Terminal, and the Propeller Manual reference documentation. These items along with the Propeller microcontroller datasheet are all on a single page at www.parallax.com.

- Go to [www.parallax.com/Propeller](http://www.parallax.com/Propeller) → Downloads & Articles.
- Download the following items and place them in a convenient folder.
  - Propeller Tool Software v1.2 or newer. System requirements: Windows 2K/XP/Vista and an available USB port.
  - Parallax Serial Terminal Software
  - Source Code – for the Propeller Education Kit Labs: Fundamentals (.zip)
  - Book – Propeller Education Kit Labs: Fundamentals (.pdf)
  - Propeller Manual
  - Propeller Datasheet
  - If you are using the PropStick USB version of the PE Kit, be sure to locate its separate Setup and Testing Lab PDF file.
Software

Figure 2-1: Propeller Tool and Parallax Serial Terminal

From PE Kit Labs, page 17
During Software Installation
Make sure to leave the Driver Install checkbox checked!

Leave this checkbox checked! You may see this step during the Propeller Tool software installation. The "optional driver" is required for these labs. It is necessary for the serial-over-USB circuit built into the Propeller Plug and PropStick USB.
Resources
Bookmark ‘em!

Useful Web Sites
In addition to www.parallax.com/Propeller, there are a couple of other web sites where you can get answers to questions as well as objects to reduce your development time on Propeller projects.

- Object exchange: http://obex.parallax.com
- Propeller Chip forum: http://forums.parallax.com → Propeller

Tech Support Resources
Parallax Inc. offers several avenues for free technical support services:
- Email: support@parallax.com
- Fax: (916) 624-8003
- Telephone: Toll free in the U.S: (888) 99-STAMP; or (916) 624-8333. Please call between the hours of 7:00 am and 5:00 pm Pacific time, or leave us a message.
- Forums: http://forums.parallax.com/forums/. Here you will find an active forum dedicated to the Propeller chip, frequented by both Parallax customers and employees.
Chapter 3
Setup and Testing
Figure 3-1: PE Kit Platform (40-Pin DIP version)
PE Platform
Schematic
PE Platform Wiring
One of the intermediate steps...
From PE Kit Labs, page 31
PE Platform Testing

Test Circuits

From PE Kit Labs, page 33 & 34
PE Platform Testing

Test Code

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>clkmode</code></td>
<td><code>xtall + pll16x</code> Feedback and PLL multiplier</td>
</tr>
<tr>
<td></td>
<td><code>5_000_000</code> External oscillator = 5 MHz</td>
</tr>
<tr>
<td><code>LEDs_START</code></td>
<td>0 Start of I/O pin group for on/off signals</td>
</tr>
<tr>
<td><code>LEDs_END</code></td>
<td>15 End of I/O pin group for on/off signals</td>
</tr>
<tr>
<td><code>PUSHBUTTON</code></td>
<td>18 Pushbutton Input Pin</td>
</tr>
</tbody>
</table>

**Main Method**

```
; Sends on/off (3.3 V / 0 V) signals at approximately 2 Hz.

; Set entire pin group to output
; Endless loop
; Change the state of pin group
; If pushbutton pressed
; Wait 1/4 second -> 2 Hz
; If pushbutton not pressed
; Wait 1/20 second -> 10 Hz
```

From PE Kit Labs, page 35
Propeller Education Kit Labs: Fundamentals

Chapter 4
I/O and Timing
Lab 4: I/O and Timing

Circuit for this lab

From PE Kit Labs, page 44
Lab 4: I/O and Timing

Emulate a Wire

```
' File: ButtonToLed.spin
  Led mirrors pushbutton state.

PUB ButtonLed

  dira[6] := 1
  dira[21] := 0

  repeat

  ' Pushbutton/Led Method
  ' P6 → output
  ' P21 → input (this command is redundant)
  ' Endless loop
  ' Copy P21 input to P6 output
```
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 checkbox 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.
Lab 4: I/O and Timing
The WAITCNT 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)
```
Lab 4: I/O and Timing
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
```
Propeller Education Kit Labs: Fundamentals

Lab 5
Methods and Cogs
Lab 5: Methods & Cogs

Calling a Method

```
' CallBlink.spin

PUB Main

    repeat
        repeat until ina[23]
        outa[9] := 0
        Blink
            waitcnt(clkfreq/2*3 + cnt)
    Next

PUB Blink | pin, rate, reps

    pin := 4
    rate := clkfreq/3
    reps := 9

    dira[pin]~
    outa[pin]~

    repeat reps * 2
        waitcnt(rate/2 + cnt)
        !outa[pin]
```

From PE Kit Labs, page 69
Lab 5: Methods & Cogs

Passing Parameters

```plaintext
'' BlinkWithParams.spin

PUB BlinkTest

    Blink(4, clkfreq/3, 9)

PUB Blink( pin, rate, reps)

    dira[pin]~
    outa[pin]~

    repeat reps * 2
        waitcnt(rate/2 + cnt)
        !outa[pin]
```

From PE Kit Labs, page 70
Lab 5: Methods & Cogs

Method Return Value

(Step 1) ButtonTime method call passes 23 to ButtonTime's pin parameter

(Step 2) ButtonTime method defines the result variable and returns this value to the method call

(Step 3) ButtonTime method's result value is assigned to the Main method's time variable

(Step 4) time is used in the Blink method call

(Step 5) Blink method receives time as the value to use in its rate parameter

```
'' ButtonBlink.spin

PUB Main | time

Repeat

time := ButtonTime(23)
Blink(4, time, 10)

PUB Blink(pin, rate, reps)

dira[pin] ~
outa[pin] ~
repeat reps * 2
waitcnt(rate/2 + cnt)
!outa[pin]

PUB ButtonTime(pin) | t1, t2

repeat until ina[pin]
t1 := cnt
repeat while ina[pin]
t2 := cnt
result := t2 - t1
```
Lab 5: Methods & Cogs

Launching Methods into Cogs

```
VAR
    long stack[30]

PUB LaunchBlinkerCogs

    cognew(Blink(4, clkfreq/3, 9), @stack[0])
    cognew(Blink(5, clkfreq/7, 21), @stack[10])
    cognew(Blink(6, clkfreq/11, 39), @stack[20])
```

PUB Blink(p, r, s)

    dira[p]~
    outa[p]~

    repeat s * 2
        waitcnt(r/2 + cnt)
        !outa[p]
```

Cog 0
LaunchBlinkerCogs commands

Cog 1
Blinker(4, clkfreq/3, 9)
RAM @stack[0]

Cog 2
Blinker(5, clkfreq/7, 21)
RAM @stack[10]

Cog 3
Blinker(6, clkfreq/11, 39)
RAM @stack[20]
Lab 5: Methods & Cogs
Memory Map and Stack Space

First unused RAM address for Cog 0's stack

From PE Kit Labs, page 72
Lab 5: Methods & Cogs
Calculating Stack Space

- 2 – return address
- 1 – return result
- number of method parameters
- number of local variables
- workspace for intermediate expression calculations

```plaintext
repeat reps * 2
  waitcnt(rate/2 + cnt)
```

- 2 – return address
- 1 – result variable (every method has this built-in, whether or not a return value is specified. This will be introduced in the next section.)
- 3 – pin, freq, and reps parameters
- 1 – time local variable
- 3 – workspace for calculations.

- 10 – Total
Propeller Education Kit Labs: Fundamentals

Lab 6
Objects
Lab 6: Objects
Calling methods in another object with “dot notation”

```
'File: DotNotationExample.spin

OBJ
  PbLed : ButtonAndBlink

PUB Main | time
  repeat
    time := PbLed.ButtonTime(23)
    PbLed.Blink(4, time, 20)

Method calls with ObjectNickname.MethodName

'File: ButtonAndBlink.spin

  Example object with two methods

  PUB ButtonTime(pin): delta | time1, time2
    repeat until ina[pin] == 1
    time1 := cnt
    repeat until ina[pin] == 0
    time2 := cnt
    delta := time2 - time1

  PUB Blink( pin, rate, reps)
    dira[pin]~
    outa[pin]~
    repeat reps * 2
      waitcnt(rate/2 + cnt)
      !outa[pin]
```

From PE Kit Labs, page 84
Lab 6: Objects

Object View Pane (Upper-Left)

```
OBJ

PbLed : "ButtonAndBlink"

PUB Main | time

repeat

time := PbLed.ButtonTime(23)
PbLed.Blink(4, time, 20)
```
Objects that Launch Cogs

{{{ Top File: CogObjectExample.spin
Blinks an LED circuit for 20 repetitions. The LED
blink period is determined by how long the P23 pushbutton
is pressed and held.
}}}

OBJ

    Blinker : "Blinker"
    Button : "Button"

PUB ButtonBlinkTime | time

    repeat
        time := Button.Time(23)
        Blinker.Start(4, time, 20)
Lab 6: Objects

Documentation View – Schematic & Object Interface

File: Blinker.spin
Example cog manager for a blinking LED process.

SCHEMATIC

100 Ω

pin

LED

GND

Object "Blinker" Interface:

PUB Start(pin, rate, reps) : success
PUB Stop
PUB Blink(pin, rate, reps)

Program: 19 Longs
Variable: 11 Longs
Lab 6: Objects

Documentation View – Methods

PUB Start(pin, rate, reps) : success

Start new blinking process in new cog; return True if successful.

Parameters:
  pin - the I/O connected to the LED circuit + see schematic
  rate - On/off cycle time is defined by the number of clock ticks
  reps - the number of on/off cycles

PUB Stop

Stop blinking process, if any.

PUB Blink(pin, rate, reps)

Blink an LED circuit connected to pin at a given rate for reps repetitions.
Lab 6: Objects
Character Map
Lab 6: Objects
Schematics in Your Documentation Comments

```{language=cpp}
Top File: CogObjectExampleWithSchematic.spin
Functionally similar to ButtonBlink and DotNotatio
can interrupt the button blinking with a new rate
blinks. That’s because the Bliinker object launche
P4 LED.

LED

<table>
<thead>
<tr>
<th>100 Ω</th>
<th>LED</th>
</tr>
</thead>
</table>
P4  ↓   |
|       |
| GND   |

Pushbutton

<table>
<thead>
<tr>
<th>3.3 V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Pushbutton</td>
</tr>
<tr>
<td>P23 \</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>$10 kΩ</td>
</tr>
</tbody>
</table>

From PE Kit Labs, page 93
Lab 6: Objects
Object Arrays

"Top File: MultiCogObjectExample.spin"

OBJ

Button : "Button"

PUB ButtonBlinkTime | time, index

repeat

repeat index from 0 to 5
    time := Button.Time(23)
    Blinker[index].Start(index + 4, time, 1_000_000)

repeat index from 5 to 0
    Button.Time(23)
    Blinker[index].Stop

From PE Kit Labs, page 95
Lab 6: Objects
Displaying Messages with FullDuplexSerial
Lab 6: Objects
Declaring & Using the FullDuplexSerial Object

```
'HelloFullDuplexSerial.spin
'Test message to Parallax Serial Terminal.

CON

_clkmode = xtal1 + pll16x
_xinfreq = 5_000_000

OBJ

Debug: "FullDuplexSerial"

PUB TestMessages

'Send test messages to Parallax Serial Terminal.

Debug.start(31, 30, 0, 57600)

repeat
    Debug.str(string("This is a test message!", 13))
    waitcnt(clkfreq + cnt)
```

From PE Kit Labs, page 99
Lab 6: Objects

FullDuplexSerial’s Documentation Comments

---

Full-Duplex Serial Driver v1.1
(C) 2006 Parallax, Inc.

Object “FullDuplexSerial” Interface:

- PUB start(rxpin, txpin, mode, baudrate) : okay
- PUB stop
- PUB rxfush
- PUB rxcheck : rxbyte
- PUB rxtime(ms) : rxbyte
- PUB rx : rxbyte
- PUB tx(txbyte)
- PUB str(stringptr)
- PUB dec(value)
- PUB hex(value, digits)
- PUB bin(value, digits)

Program: 188 Longs
Variable: 18 Longs

PUB start(rxpin, txpin, mode, baudrate) : okay

Start serial driver - starts a cog
returns false if no cog available

Mode bit 0 = invert rx
Mode bit 1 = invert tx
Mode bit 2 = open-drain/source tx
Mode bit 3 = ignore tx echo on rx

---

Parallax Inc.

From PE Kit Labs, page 101
Lab 6: Objects
Parallax Serial Terminal Application Examples

From PE Kit Labs, page 103, 105, 107, 113
Propeller Education Kit Labs: Fundamentals

Lab 7
Counter Modules & Circuit Applications
Block Diagram

PE Kit Labs Book, page 221
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
Counter Module Modes

• POS/NEG detector
  – Measure Pulse, RC, Duty cycle
• w/Feedback
  – ADC
• Edge Detect
  – Frequency measurements, count events
• LOGIC
  – Detect conditions

• NCO
  – Square waves, clock signals, PWM, duty Cycle
• PLL
  – High frequency square waves and clock signals, video mode
• DUTY
  – DAC
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 to 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.
Lab 7: Counter Modules…
Circuit Application: Measure RC Decay

Parts List
(1) Potentiometer 10 kΩ
(1) Capacitor - 0.01 μF
(misc) Jumper wires

Schematic
P17
R
GND

0.01 μF
GND

From PE Kit Labs, page 122
Lab 7: Counter Modules…

RC Decay Circuit Steps

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

3.3 V

\[ V_c \]

\[ i \rightarrow \]

\[ i \downarrow \]

\[ R \]

\[ GND \]

\[ i_c \downarrow \]

\[ C \]

\[ GND \]

Decay Circuit (I/O pin = input)

\[ V_c \]

\[ \text{I/O Pin} \leftarrow \]

\[ \downarrow i \uparrow \]

\[ R \]

\[ GND \]

\[ C \]

\[ GND \]

\[ \Delta t = 0.693 \times C \times R \]

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...

<table>
<thead>
<tr>
<th>CTRMODE</th>
<th>Description</th>
<th>Accumulate FRQ to PHS</th>
<th>APIN output*</th>
<th>BPIN output*</th>
</tr>
</thead>
<tbody>
<tr>
<td>%000000</td>
<td>Counter disabled (off)</td>
<td>0 (never)</td>
<td>0 (none)</td>
<td>0 (none)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CTRMODE</th>
<th>Description</th>
<th>APIN output*</th>
<th>BPIN output*</th>
</tr>
</thead>
<tbody>
<tr>
<td>%010000</td>
<td>POS detector</td>
<td>A¹</td>
<td>0</td>
</tr>
<tr>
<td>%010001</td>
<td>POS detector w/feedback</td>
<td>A¹</td>
<td>0</td>
</tr>
<tr>
<td>%010100</td>
<td>POSEDGE detector</td>
<td>A¹ &amp; !A²</td>
<td>0</td>
</tr>
<tr>
<td>%010101</td>
<td>POSEDGE detector w/feedback</td>
<td>A¹ &amp; !A²</td>
<td>!A¹</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CTRMODE</th>
<th>Description</th>
<th>APIN output*</th>
<th>BPIN output*</th>
</tr>
</thead>
<tbody>
<tr>
<td>%11111</td>
<td>LOGIC always</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

* must set corresponding DIR bit to affect pin

A¹ = APIN input delayed by 1 clock
A² = APIN input delayed by 2 clocks
B¹ = BPIN input delayed by 1 clock

<table>
<thead>
<tr>
<th>bits</th>
<th>31</th>
<th>30..26</th>
<th>25..23</th>
<th>22..15</th>
<th>14..9</th>
<th>8..6</th>
<th>5..0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>—</td>
<td>CTRMODE</td>
<td>PLLDIV</td>
<td>—</td>
<td>BPIN</td>
<td>—</td>
<td>APIN</td>
</tr>
</tbody>
</table>
Lab 7: Counter Modules…
Counter Setup Steps for RC Decay

1) Store `%01000` in the CTR register’s CTRMODE bit field:

\[
\text{ctr}[30..26] := %01000
\]

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

\[
\text{ctr}[5..0] := 17
\]

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:

\[
\text{frq} := 1
\]
Lab 7: Counter Modules…
from TestRcDecay.spin

' Charge RC circuit.
waitcnt(clkfreq/100_000 + cnt)                ' Wait for circuit to charge

' Start RC decay measurement. It’s automatic after this...
phsa~
dira[17]~                                      ' Clear the phsa register
                                            ' 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)
Lab 7: Counter Modules…

D/A Conversion

**Time Varying D/A and Filtering:** When modulating the value of \( f_{\text{freq}} \) 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.

![Diagram](image.png)
Lab 7: Counter Modules…
Duty Modes from CTR.spin

<table>
<thead>
<tr>
<th>CTRMODE</th>
<th>Description</th>
<th>Accumulate FRQ to PHS</th>
<th>APIN output*</th>
<th>BPIN output*</th>
</tr>
</thead>
<tbody>
<tr>
<td>%00000</td>
<td>Counter disabled (off)</td>
<td>0 (never)</td>
<td>0 (none)</td>
<td>0 (none)</td>
</tr>
<tr>
<td>%00110</td>
<td>DUTY single-ended</td>
<td>1</td>
<td>PHS-Carry</td>
<td>0</td>
</tr>
<tr>
<td>%00111</td>
<td>DUTY differential</td>
<td>1</td>
<td>PHS-Carry</td>
<td>!PHS-Carry</td>
</tr>
<tr>
<td>%11111</td>
<td>LOGIC always</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>bits</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>CTRMODE</td>
</tr>
<tr>
<td>30..26</td>
<td>PLLDIV</td>
</tr>
<tr>
<td>25..23</td>
<td></td>
</tr>
<tr>
<td>22..15</td>
<td></td>
</tr>
<tr>
<td>14..9</td>
<td></td>
</tr>
<tr>
<td>8..6</td>
<td></td>
</tr>
<tr>
<td>5..0</td>
<td></td>
</tr>
</tbody>
</table>

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.
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:

\[ \text{ctr}a[30..26] := \%00110 \]

(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 \text{ctr}a register’s APIN bits to 4, which will control the green LED connected to P4.

\[ \text{ctr}a[5..0] := 4 \]

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.*

\[ \text{dir}a[4] = \text{~} \]

(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 \text{frq}a register to 1_073_741_824 (calculated earlier).

\[ \text{frq}a := 1_073_741_824 \]
Lab 7: Counter Modules...

Metal Detection

Parts List

(1) Capacitor 10 pf
(2) Jumper Wires
(2) Resistors 100 ohm
(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

Time
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

CalibrateMetalDetector.spin

From PE Kit Labs, page 175 - 176