PropIRC
A Propeller IRC Client

October 21, 2008
Harrison Pham <harrison@harrisonpham.com>

2008 Propeller Design Contest
Project Number PC173

Table of Contents

Project Number

PC173

Project Description

Introduction

Internet Relay Chat, also known as IRC, is a very popular protocol for multi user chat and collaboration. It is commonly used in gaming communities and open source project collaboration.

Many IRC users tend to 'idle' in channels. This allows them to keep logs of chats so that they don't miss anything. Keeping a desktop PC on just to idle in an IRC channel can be very expensive. Current alternative solutions are not easy to configure and tend to be frowned upon.

This is where PropIRC comes in. PropIRC implements a minimal client that directly connects to remote IRC servers via a standard Ethernet connection. The only required external devices are a standard VGA monitor and a PS2 keyboard. This minimizes power consumption costs and provides a quick and easy method of accessing IRC without having to resort back to using a desktop PC.

Features

- User Interface
  - Text based with a clean color scheme
  - Simple intuitive keyboard interface
  - Built in web server for remote access
  - Intuitive key / value settings for configuration
- Hardware
  - 10mbps Ethernet Connection
  - 1024x768 VGA Output
  - PS2 Keyboard Input
  - Configuration stored on code EEPROM
  - Tiny 2"x2" PCB
- Software
  - Modular design using objects
  - Abstracted TCP/IP stack for internet connectivity
- Supported Commands
  - /nick – Change nickname
  - /join – Join channel
  - /part – Leave channel
  - /quit – Disconnect from the server
  - /msg – Send private message
  - /set – Modify / view current settings
  - Many other built in IRC server commands via direct / pass thru

User Interface

The user interface is one of the most important things when designing any sort of client application. This presents unique problems since the Propeller is memory constrained, unlike modern desktop PCs. In order to efficiently use the memory, a text only user interface was chosen.

Text based interfaces are generally looked down upon these days. Luckily, there are still a few IRC clients out there that have excellent interfaces. One such client is Irssi. Irssi is a text based client that is usually the preferred client of choice on linux machines. Irssi has a simple interface that can be partially mimicked on the Propeller.

The final UI design was based loosely on Irssi's interface and color scheme. The pale colored text on a black background provides a nice soft feel. The command structure consists of the most used IRC functions. Settings are achieved through a key/value based system, similar to the one used in Irssi.

Modifying Settings without Recompilation

Most PropIRC settings can be changed without recompilation and reprogramming. This simplifies client usage, and greatly reduces sources of error when the end user is using the client.

Viewing Current Settings

```
[00:11:55] #propeller proptcp: /set
[00:11:55] server=140.211.166.3:6667
[00:11:55] user=username
[00:11:55] pass=password
[00:11:55] nick=proptcp
[00:11:55] channel=#propeller
[00:11:55] tzoff=-18000
[00:11:55] Notice: You must do /save and /reboot to commit and load settings.
```

Users can view settings by entering '/say'. All supported settings are displayed.

Modifying a Setting

```
[00:21:53] #propeller proptcp: /set tzoff -21600
[23:21:53] Notice: You must do /save and /reboot to commit and load settings.
```

Modifying a setting is extremely easy. Settings are not committed immediately to ensure the user has time to verify changes.

Reloading Settings

```
[23:24:59] #propeller proptcp: /load
[00:24:59] Your settings have been reloaded.
```

Incorrect settings can be easily cleared by reloading previous settings from EEPROM.

Chatting

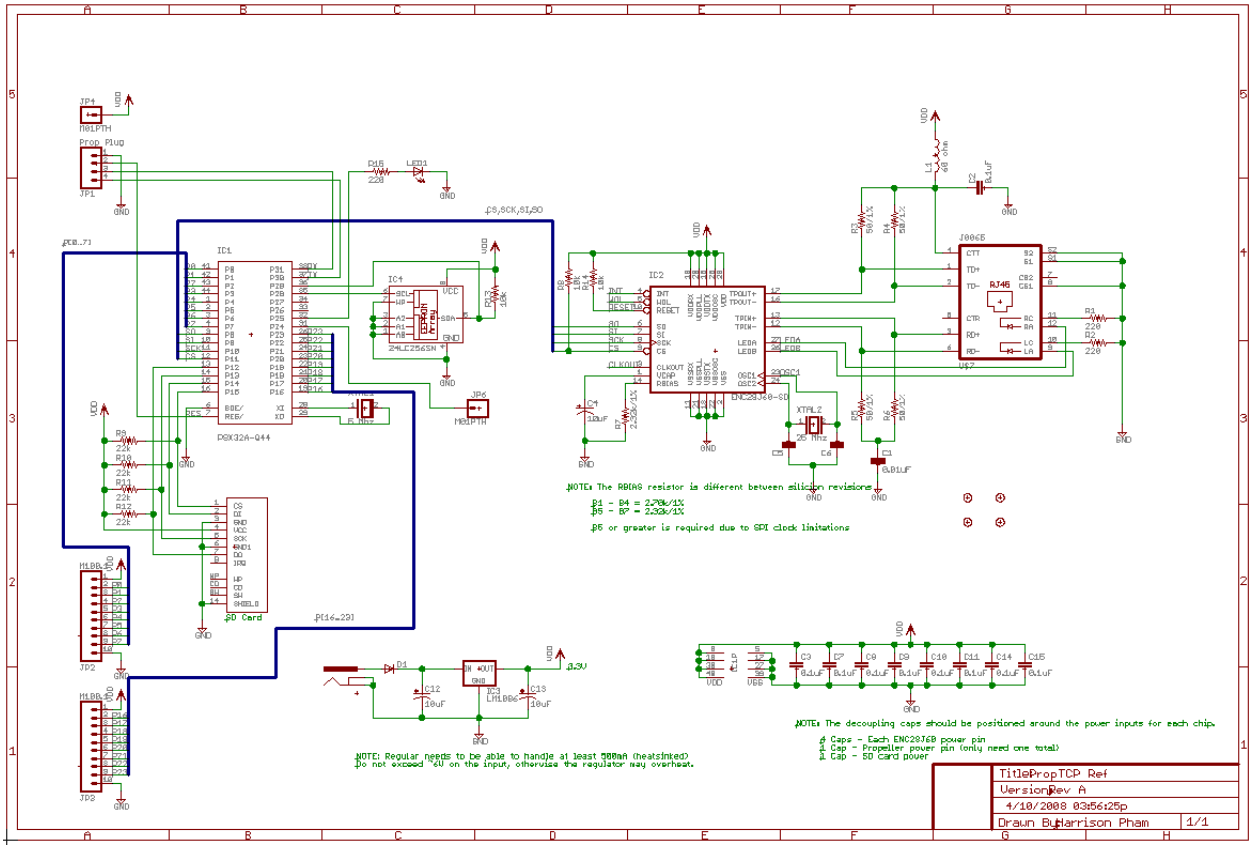
Chatting with PropIRC is extremely simple. It functions like the majority of IRC clients out there. Common commands are supported, which reduces the learning curve.

A Chat Session

```
[23:23:14] #propeller harrison_: hi proptcp!
[23:23:23] #propeller proptcp: sup!
[23:23:25] #propeller RinksCustoms: night
[23:23:43] #propeller proptcp: night
[23:24:10] RinksCustoms has left #propeller
[00:08:16] _BradC has joined #propeller
[00:11:46] #propeller harrison_: why hello there
[00:11:55] #propeller proptcp: /set
[00:11:55] server=140.211.166.3:6667
[00:11:55] user=username
[00:11:55] pass=password
[00:11:55] nick=proptcp
[00:11:55] channel=#propeller
[00:11:55] tzoff=-18000
[00:11:55] Notice: You must do /save and /reboot to commit and load settings.
[00:15:27] #propeller _BradC: heya
[00:21:53] #propeller proptcp: /set tzoff -21600
[23:21:53] Notice: You must do /save and /reboot to commit and load settings.
[23:21:57] #propeller proptcp: /set
[23:21:57] server=140.211.166.3:6667
[23:21:57] user=username
[23:21:57] pass=password
[23:21:57] nick=proptcp
[23:21:57] channel=#propeller
[23:21:57] tzoff=-21600
[23:21:57] Notice: You must do /save and /reboot to commit and load settings.
[23:24:59] #propeller proptcp: /load
[00:24:59] Your settings have been reloaded.
[00:25:14] #propeller _BradC: right.. time to go to work..
[00:26:54] #propeller proptcp: have fun :)
```

The message format is self explanatory. A time stamp, message source, nickname, and the actual message are all displayed for easy viewing. Multiple channels are supported and all displayed together on one screen.

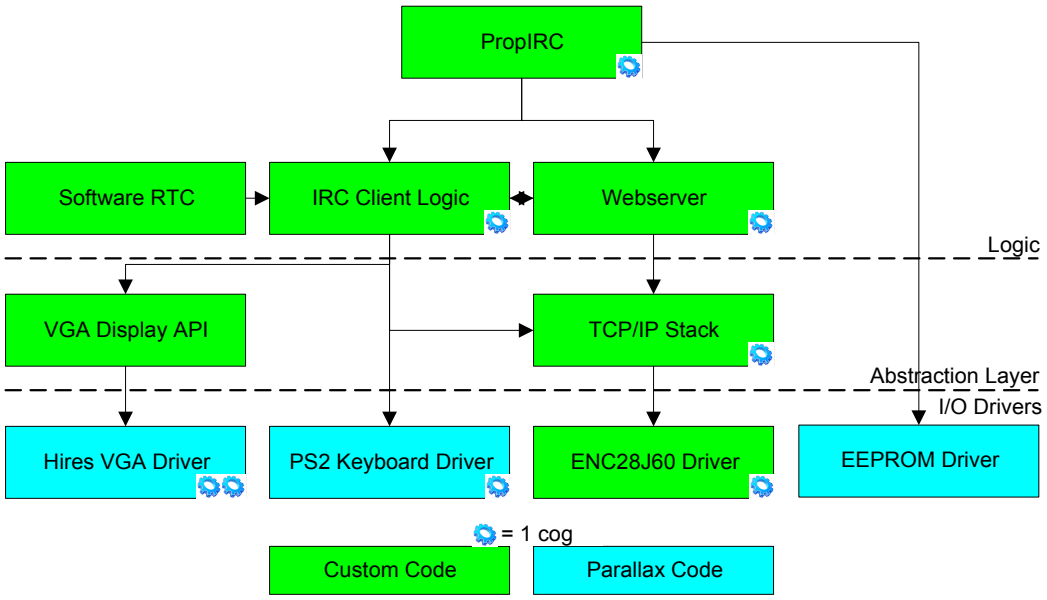
Schematic



The VGA signal is output through P16-P23 using the standard schematic used by the demo board. The PS2 keyboard is connected to P6 and P7, also using the standard demo board schematic.

A full version is included in “propirc-schematic.pdf”.

Block Diagram



Block Descriptions

- **Software RTC** – Provides a network synchronized real time clock. This clock keeps track of time as a unix timestamp and provides the time that is used to display the timestamp by each IRC message.
- **IRC Client Logic** – Implements all of the IRC logic and parsing. Most of this is implemented as one large method that performs large amounts of string manipulation. It also handles all user input and output.
- **Webserver** – Implements a basic HTTP webserver. This webserver is used for remote access to the IRC client. The web page it provides mimics the VGA display and keyboard input as closely as possible in order to maintain a consistent user experience.
- **VGA Display API** – Provides an abstraction layer between the hires VGA driver and the IRC client code. It provides numerous methods for printing to the chat area and the debug area. It also handles the color theme.
- **TCP/IP Stack** – This is a 100% custom designed object that runs in a separate cog that implements a TCP/IP IPv4 stack. It provides multiple sockets to the application code. Application code can access the TCP/IP stack through an API that is similar to FullDuplexSerial.
- **Hires VGA Driver** – Performs the complex art of generating the 1024x768 VGA text display.
- **PS2 Keyboard Driver** – Interfaces with the PS2 keyboard, providing easy access to key presses.
- **ENC28J60 Driver** – Implements a low level API for accessing the ENC28J60 network interface chip. It also contains a custom written PASM SPI driver.
- **EEPROM Driver** – Provides an easy method of saving and restoring variables to and from the code EEPROM.

Source Code

Design

PropIRC is partitioned into multiple objects. This allows code reuse and simplifies many mundane tasks, such as string manipulation.

- **api_telnet_serial.spin** – Provides a FullDuplexSerial-like API. This is nearly a drop in replacement for FullDuplexSerial, and greatly simplifies sockets programming.
- **driver_socket.spin** – A 100% custom IPv4 TCP/IP stack implementation. It supports multiple sockets and uses circular buffers to communicate with your application. This simplifies application coding greatly. It implements IP, ARP, and TCP. Other protocols (such as ICMP) are excluded to reduce code size.
- **driver_enc28j60.spin** – Interfaces with the ENC28J60 ethernet NIC chip. It utilizes a custom assembly SPI driver that performs block reads and writes to the ENC28J60 SRAM in order to boost transfer speeds. This object also provides an abstraction layer for accessing the ENC28J60 registers, settings, and SRAM.
- **propirc-vgatext.spin** – Provides a simple API for outputting text to the VGA monitor. There are 4 regions of text: a title bar, chat region, typing region, and a debug section. These are all accessed independently in code which makes things easier for modification.
- **propirc-vga_hires_text.spin** – Basically a renamed stock Propeller Object that implements a 1024x768 VGA output. This was renamed to reduce the reliance on objects from the default library which could change in future releases.
- **Keyboard.spin** – An I/O driver and SPIN API for accessing PS2 keyboards.
- **util_strings.spin** – Provides some singleton methods for manipulating and mutating strings. The most useful function is indexOf(…), which finds the starting index of a 'needle' inside a 'haystack' string.
- **soft_rtc.spin** – Implements a software real time clock using both cog counters. It synchronizes with NIST time servers in order to provide the most accurate time possible. The time / date is kept internally as a unix timestamp for simplicity.
- **date_time_epoch.spin** – A third party object obtained from the object exchange. It provides methods for converting date and time to epoch times. This is used by the RTC to convert to and from the unix timestamp.
- **propirc-eepromvar.spin** – A Propeller Education Kit Lab object that provides simple methods for saving and restoring variables to and from the code EEPROM. This is used to save settings.

Obtaining the Source Code

All source code is attached in the PropIRC source code zip archive named "propirc-sources.zip". The source is also provided at the end of this document (after the pictures) due to the sheer length.

Bill of Materials

Parts

Qty	Part	Description
2	C5, C6	18pF Ceramic
1	D1	1N4001 Diode
1	LED1	Green LED
1	C1	0.01uF Ceramic Cap
9	C2, C3, C7, C8, C9, C10, C11, C14, C15	0.1uF Ceramic Bypass Cap
1	R7	2.32k/1% Resistor
1	XTAL1	5 Mhz Crystal
3	R8, R13, R14	10k Resistor
3	C4, C12, C13	10uF Electrolytic Capacitor
4	R9, R10, R11, R12	22k Resistor
1	IC4	24LC256SN
1	XTAL2	25 Mhz Crystal
4	R3, R4, R5, R6	50/1% Resistor
1	L1	Ferrite Bead
3	R1, R2, R15	220 ohm Resistor
1	IC2	ENC28J60-SO
1	U$7	J00-0065NL RJ45 w/built-in magnetics
1	IC3	LM2937ET-3.3
2	JP2, JP3	10 pin header
1	IC1	P8X32A-Q44
1	J2	2.1mm Barrel Jack
1	JP1	4 pin header
1	U$1	SD Card Socket
4	U$2, U$3, U$4, U$5	Nylon standoff + screw

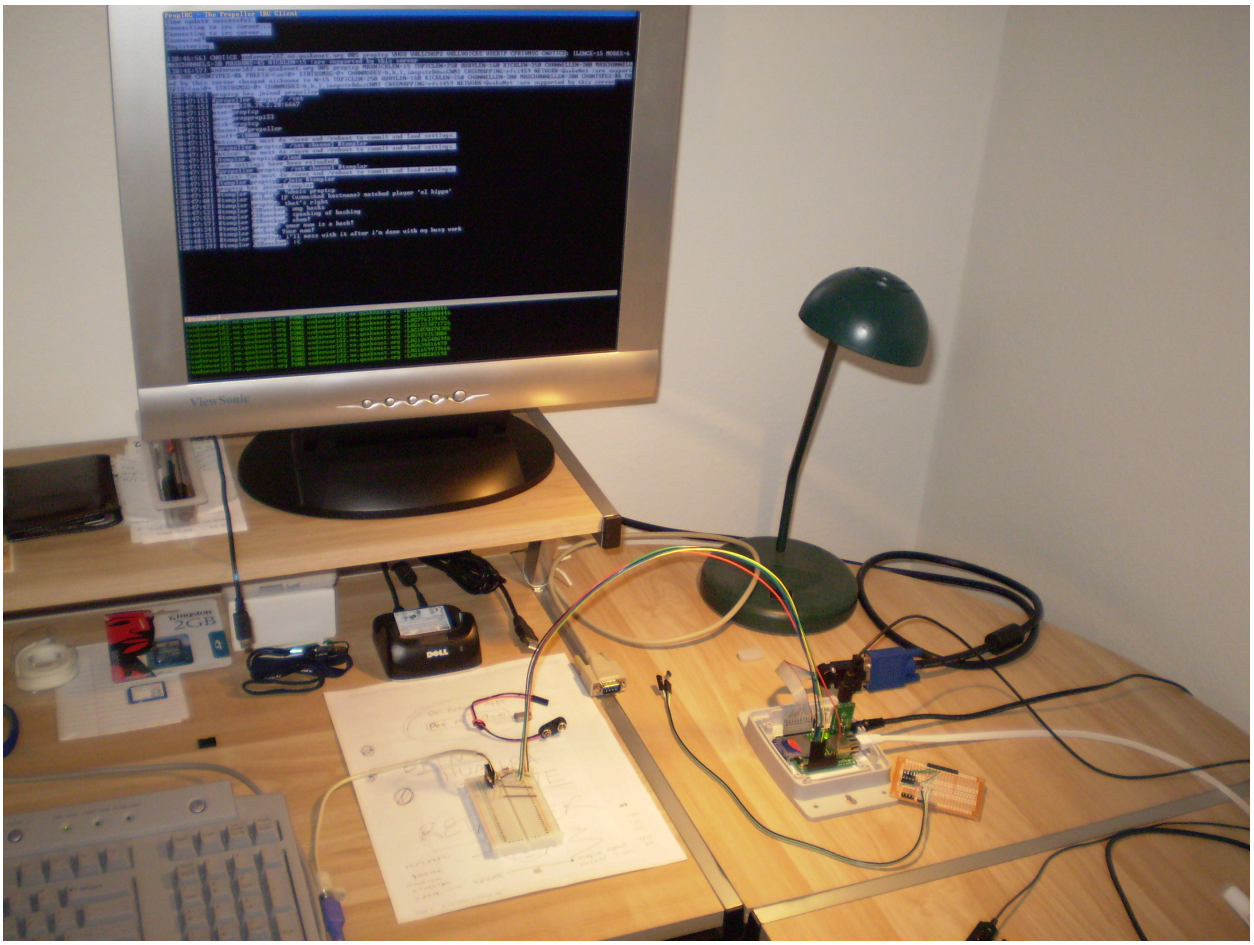
Miscellaneous Parts

The following parts are used as interface cables for the monitor and keyboard connections. They are not used if only the web based remote access mode is used.

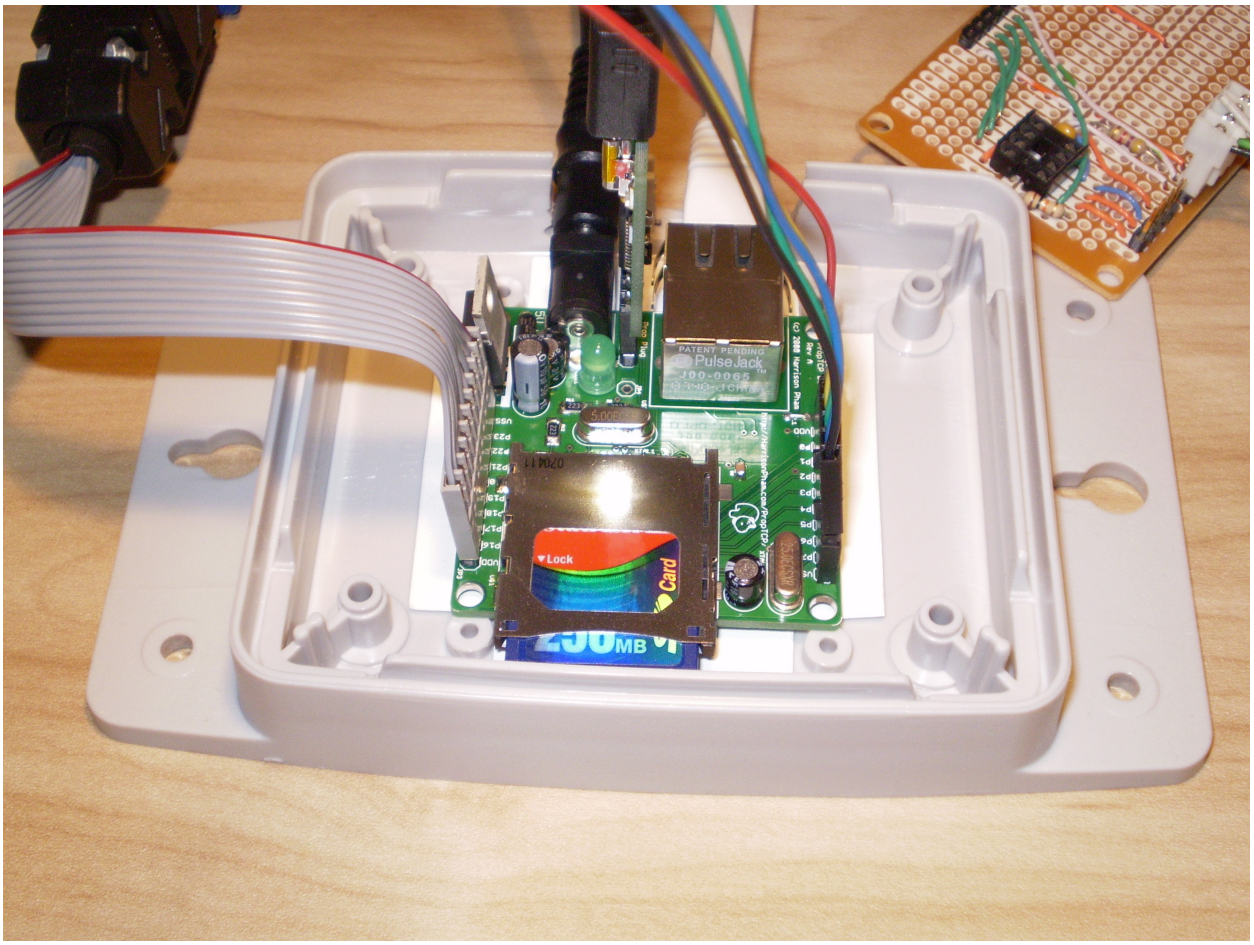
Qty	Description
1	DB-15HD Female
1	10 conductor ribbon cable
5	240 ohm resistor
3	370 ohm resistor
1	6 pin female mini-din socket
2	100 ohm resistor
2	10K resistor

Pictures

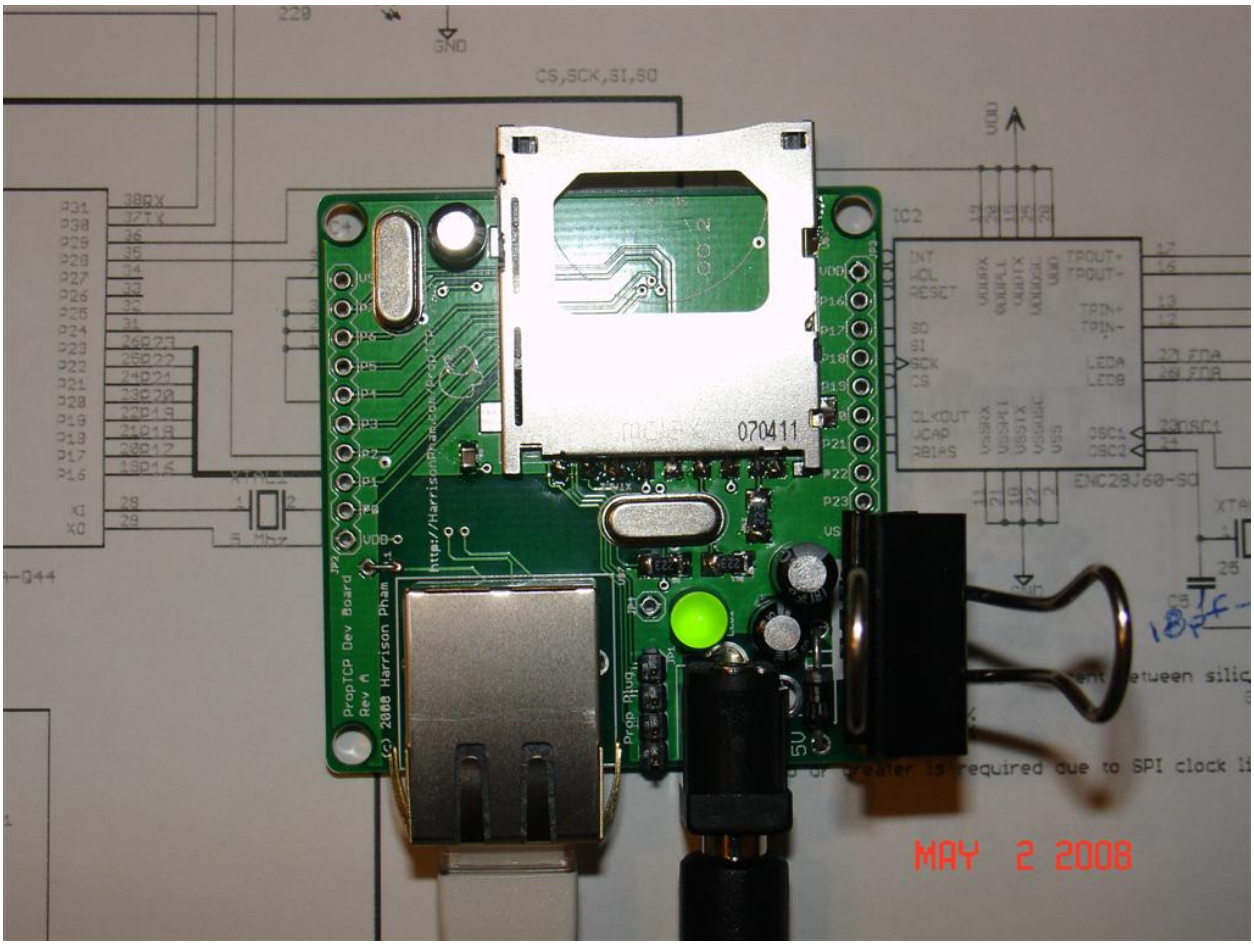
Picture 1: The designer holding the PropIRC PCB.



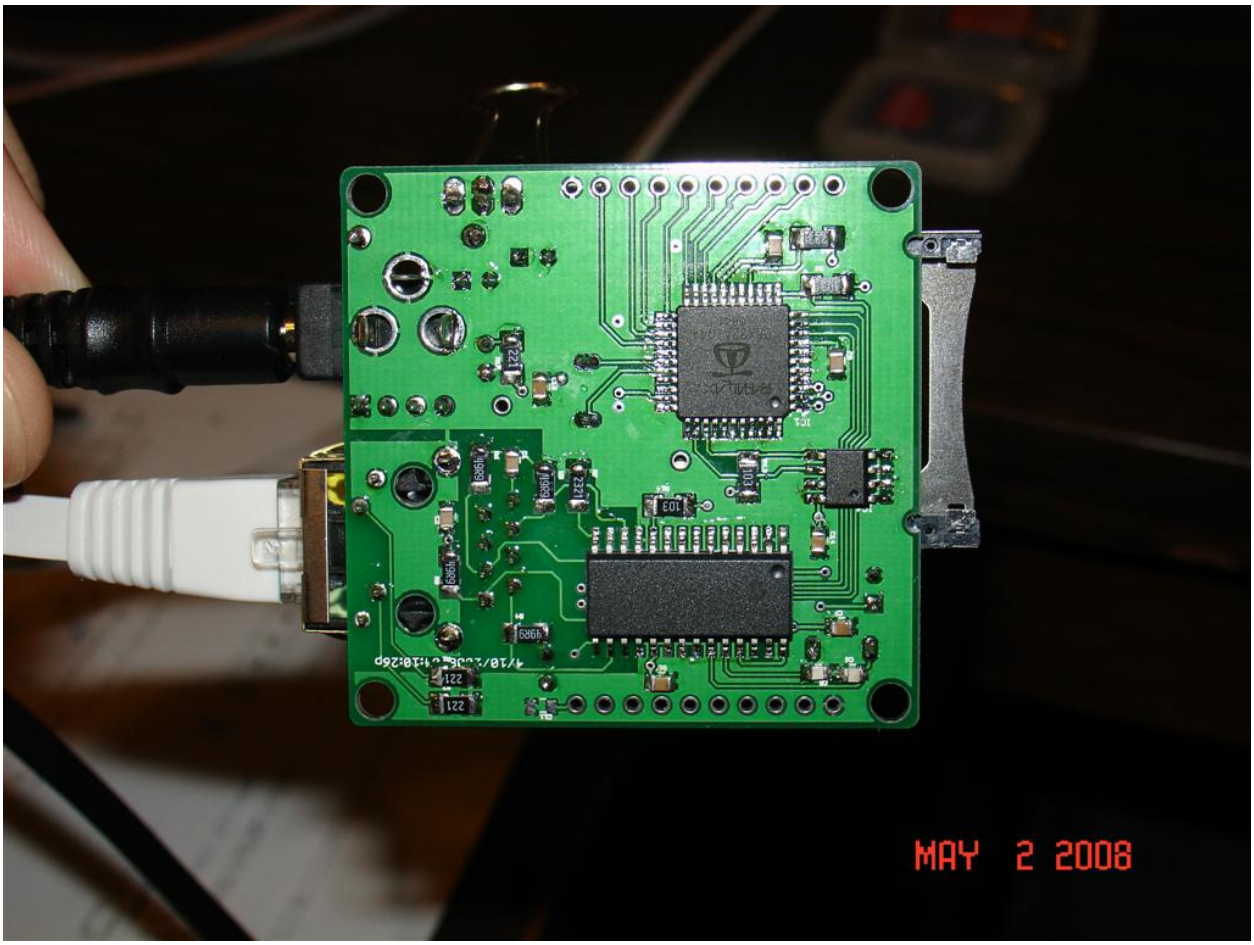
Picture 2: PropIRC with a LCD VGA monitor and a PS2 keyboard. The PropIRC PCB is located to the bottom right of the picture. The breadboard in the middle is used to hold the 4 resistors required for the PS2 keyboard interface.



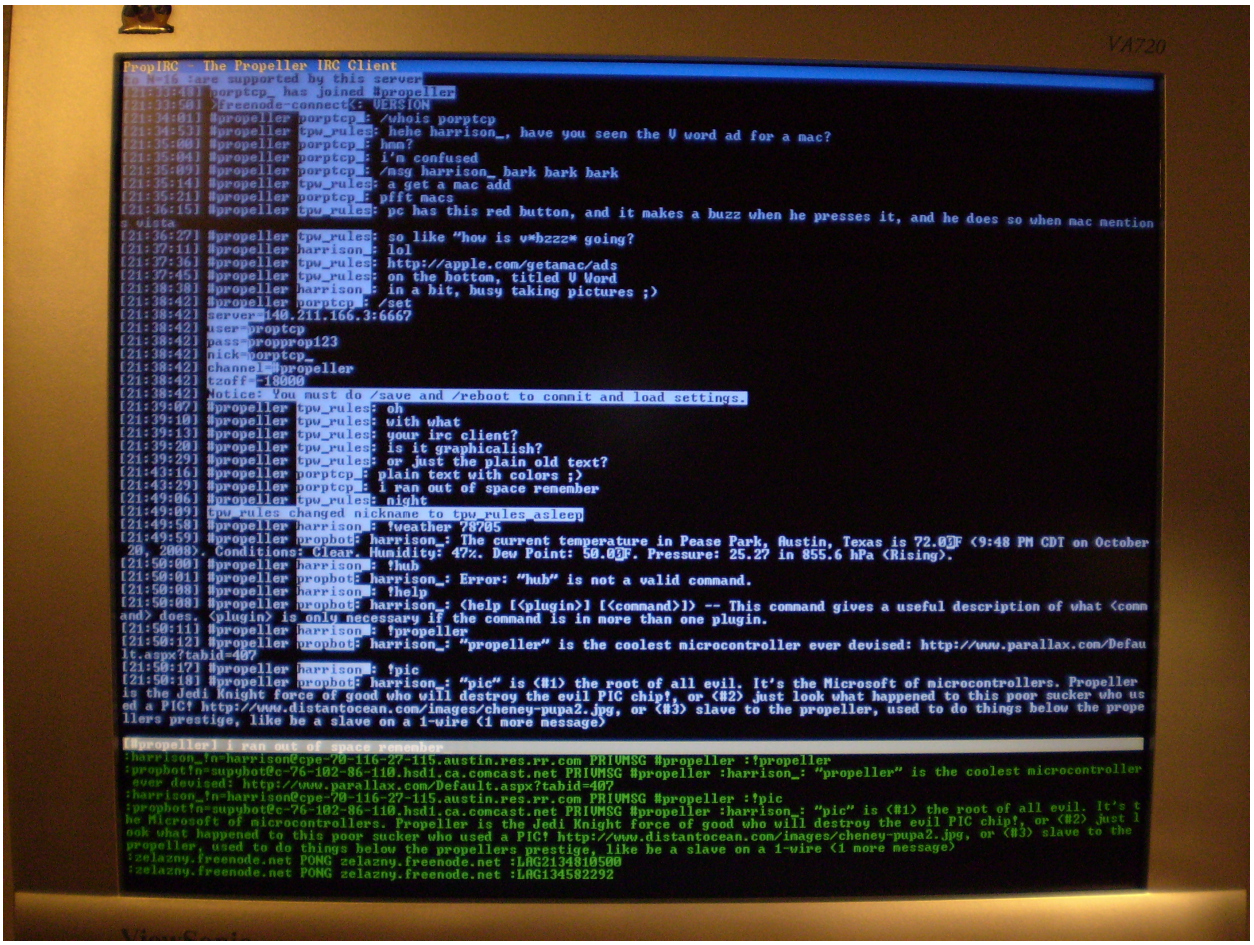
Picture 3: The board sitting in its plastic case. The VGA cable connects to the left 10 pin header and the PS2 keyboard connects to the right 10 pin header. Ethernet and power are connected in the back. The PropPlug programming interface is accessed in between the Ethernet and the power plugs.



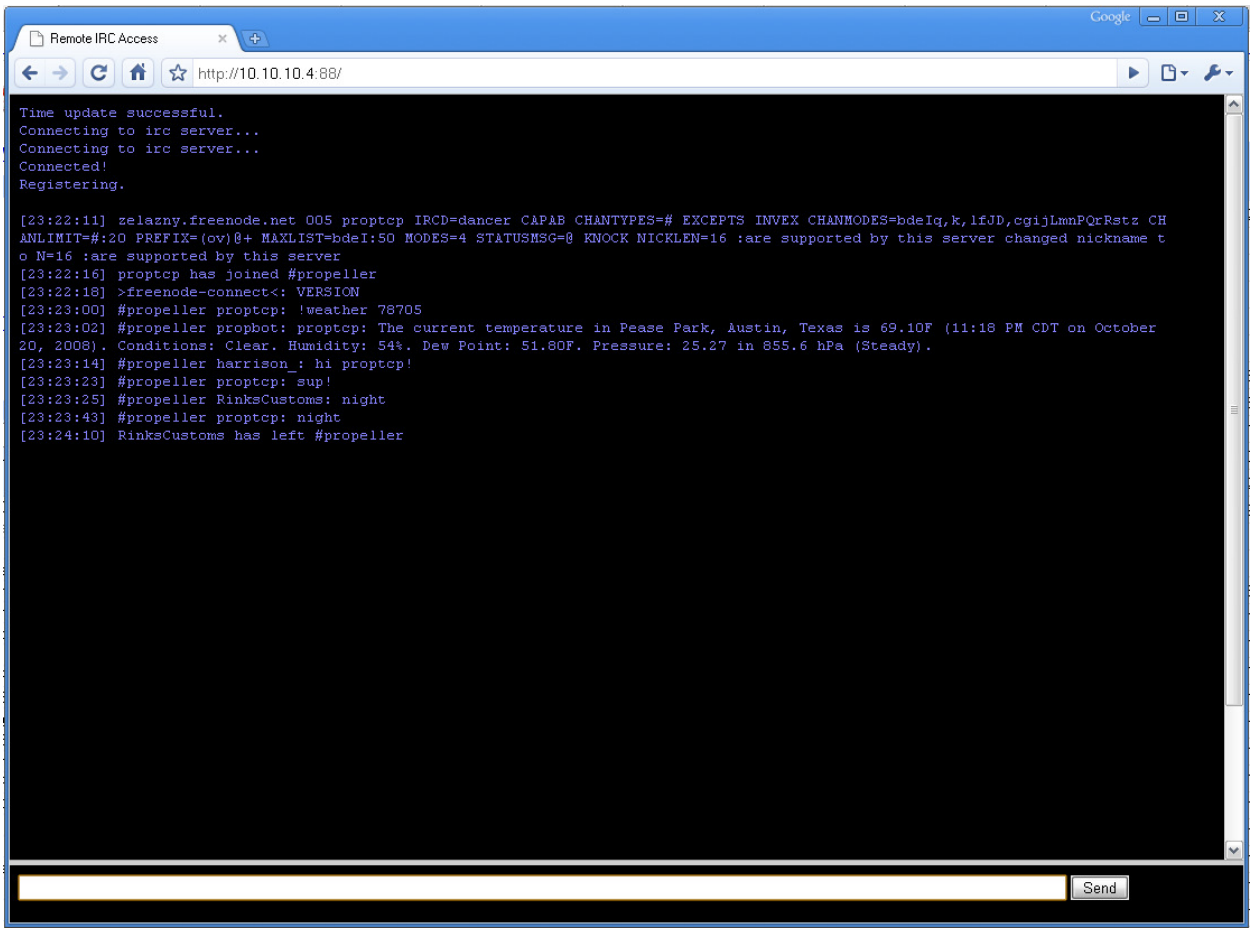
Picture 4: Top view of the PCB.



Picture 5: Bottom view of the PCB.



Picture 6: A screen shot of the IRC display while connected to #propeller on irc.freenode.net.



Picture 7: PropIRC's Remote Access mode. You can connect to the board from anywhere in the world by using any standards compliant browser.

Picture 8: The standard VGA interface and the remote access web interface shown at the same time. Both interfaces can be used at the same time.

Appendix: Source Code

propirc.spin

```
{{
  Propeller Based IRC Client w/ VGA and Keyboard
  ----------------------------------------------

  Copyright (C) 2006-2008 Harrison Pham

  This file is part of PropIRC.

  PropIRC is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 3 of the License, or
  (at your option) any later version.

  PropIRC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
}}

CON

  _clkmode = xtal1+pll16x
  _xinfreq = 5_000_000

OBJ

  sock    : "api_telnet_serial"
  web     : "api_telnet_serial"
  vga     : "propirc-vgatext"
  key     : "keyboard"
  strings : "util_strings"
  rtc     : "softrtc"
  dt      : "date_time_epoch"
  ee      : "propirc-eepromvar"
  'num     : "Numbers"

VAR

  long keyStack[64]
  byte keyBuffer[256]

  long webStack[64]

  byte buffer[512]
  byte bSend                    ' boolean flags between the keyboard cog and the irc client cog
  byte bQuit
  byte bReboot

  ' settings
  long settingsHash             ' hash used to validate configuration variables
  long myServerIP               ' the server IP, packed in a 32 bit number
  word myServerPort             ' server port number
  byte myUser[16]               ' user name
  byte myPass[16]               ' password
  byte myNick[16]               ' default nickname
  byte myChannel[32]            ' default channel
  long myTimeOffset             ' timezone offset (in seconds)

DAT
  mac_addr      byte    $10, $00, $00, $00, $00, $01

  ip_addr       byte    10, 10, 10, 4                       ' device's ip address
  ip_subnet     byte    255, 255, 255, 0                    ' network subnet
  ip_gateway    byte    10, 10, 10, 254                     ' network gateway (router)
  ip_dns        byte    10, 10, 10, 254                     ' network dns

PUB start | port
```

```
  key.start(6,7)

  sock.start(11,10,9,8,-1,-1,@mac_addr,@ip_addr)         '19,18,17,16

  vga.start(16)

  delay_ms(150)

  cognew(keyHandler, @keyStack)

  cognew(webServer, @webStack)

  restoreSettings

  if ((myServerIP + myServerPort + myUser[1] + myPass[2] + myNick[3] + myChannel[4]) <> settingsHash) or (myServerIP
  == 0)
    ' hash check failed, init default settings
    myServerIP := constant((140 << 24) + (211 << 16) + (166 << 8) + 3)
    myServerPort := 6667
    setSettingStr(@myUser, string("username"))
    setSettingStr(@myPass, string("password"))
    setSettingStr(@myNick, string("proptcp"))
    setSettingStr(@myChannel, string("#propeller"))
    myTimeOffset := constant(60 * 60 * -5)              ' CDT (GMT - 5 hours)
    commitSettings
    vga.invstr(string("Notice: Default settings were loaded and saved.",13))

  rtc.start(25)
  if rtc.update => 0
    vga.invstr(string("Time update successful.",13))
  else
    vga.invstr(string("Failed to update time.",13))

  repeat

    if \ircClient(port) < 0
      printTime
      vga.invstr(string(13,"Error: TCP socket terminated by remote host",13))

    sock.close
    delay_ms(500)

PRI commitSettings
  ' saves user settings to the system eeprom
  ee.VarBackup(@myServerIP, @myServerIP + 3)
  ee.VarBackup(@myServerPort, @myServerPort + 1)
  ee.VarBackup(@myUser, @myUser[15])
  ee.VarBackup(@myPass, @myPass[15])
  ee.VarBackup(@myNick, @myNick[15])
  ee.VarBackup(@myChannel, @myChannel[31])
  ee.VarBackup(@myTimeOffset, @myTimeOffset + 3)

  settingsHash := myServerIP + myServerPort + myUser[1] + myPass[2] + myNick[3] + myChannel[4]
  ee.VarBackup(@settingsHash, @settingsHash + 3)

PRI restoreSettings
  ' saves user settings to the system eeprom
  ee.VarRestore(@myServerIP, @myServerIP + 3)
  ee.VarRestore(@myServerPort, @myServerPort + 1)
  ee.VarRestore(@myUser, @myUser[15])
  ee.VarRestore(@myPass, @myPass[15])
  ee.VarRestore(@myNick, @myNick[15])
  ee.VarRestore(@myChannel, @myChannel[31])
  ee.VarRestore(@myTimeOffset, @myTimeOffset + 3)
  ee.VarRestore(@settingsHash, @settingsHash + 3)

PRI setSettingStr(destptr, strptr)
  ' saves a string setting to RAM
  bytemove(destptr, strptr, strsize(strptr) + 1)

PUB webServer
  ' web server (runs in a seperate cog)
  \web.listen(88)
  repeat
    \web.rxcheck
    \web.resetBuffers
    if web.isConnected
```

```
      if \webServerThread == 0
        \web.txflush
      \web.close

DAT
http200       byte      "HTTP/1.0 200 OK",13,10,13,10,0
framepg       byte      "<html><head><title>Remote IRC Access</title></head><frameset rows=",34,"100%,50",34,"><frame
  src=",34,"i",34,"><frame src=",34,"t",34,"></frameset></html>",0
textboxpg     byte      "<html><body bgcolor=",34,"#000000",34,"
  onload=",34,"document.forms[0].elements[0].focus();",34,"><form method=",34,"GET",34," action=",34,"g",34,"><input
  type=",34,"text",34," name=",34,"c",34," size=",34,"150",34," autocomplete=",34,"off",34,"> <input
  type=",34,"submit",34," value=",34,"Send",34,"></form></body></html>",0
iframepg      byte      "<html><body bgcolor=",34,"#000000",34," text=",34,"#8585ff",34,"><div
  id=",34,"t",34,"></div><iframe src=",34,"c",34," style=",34,"display:none",34,"></iframe></body></html>",0
chathead      byte      "<html><head><meta http-equiv=",34,"refresh",34," content=",34,"3",34,"></head><body><div
  id=",34,"t",34,"><pre>",0
chattail      byte      "</pre></div><script>var s=document.getElementById('t').innerHTML;var o='';var
  i;for(i=0;i<s.length;i++){var c=s.charCodeAt(i);o+=''+String.fromCharCode((c<128)?(c):(c-
  128));}parent.document.getElementById('t').innerHTML=o;</script></body></html>",0
textgetpg     byte      "HTTP/1.0 302 Found",13,10,"Location: t",13,10,13,10,0
'chathead      byte      "<html><head><meta http-equiv=",34,"refresh",34," content=",34,"5",34,"></head><body><div
  id=",34,"t",34," style=",34,"visibility:hidden",34,"><pre>",0
'chattail      byte      "</pre></div><script>var e=document.getElementById('t');var s=e.innerHTML;var o='';var
  i;for(i=0;i<s.length;i++){var c=s.charCodeAt(i);o+=''+String.fromCharCode((c<128)?(c):(c-
  128));}e.innerHTML=o;e.style.visibility='visible';</script></body></html>",0
crlf          byte      13,10,0

PRI webServerThread | ptr, row, c, i, n, fname
  ' handles a single client connection

  ' match GET request
  ' we don't have much memory so we only support 1 character args
  c := webRead
  if c == "G"
    repeat 4                    ' GET /
      webRead
  elseif c == "P"
    repeat 5                    ' POST /
      webRead
  else
    return -1                   ' unrecognized method

  fname := webRead              ' single char filename

  ' scan for Auth header

  if fname == "g"
    if webRead == "?"
      ' we got an arg following
      repeat 2
        webRead                 ' c=
      i := 0
      repeat 255
        if (c := webRead) == " "
          quit
        if c == "+"
          keyBuffer[i++] := " "
        elseif c == "%"
          c := webRead
          if c => "A"
            c -= 7
          n := (c - "0") << 4
          c := webRead
          if c => "A"
            c -= 7
          n += c - "0"
          keyBuffer[i++] := n
        else
          keyBuffer[i++] := c
      keyBuffer[i] := 0

      bSend := true

    web.str(@textgetpg)

    return 0

  web.str(@http200)
```

```
  if fname == "c"

    web.str(@chathead)

    ptr := vga.getScreenPtr

    repeat row from vga#topfirstrow to constant(vga#toplastrow - 1)
      web.txdata(ptr + (row * vga#cols), vga#cols)
      web.str(@crlf)

    web.str(@chattail)

  elseif fname == "t"
    ' we got possible args to read!

    web.str(@textboxpg)

  elseif fname == "i"

    web.str(@iframepg)

  else

    web.str(@framepg)

  return 0

PRI webRead | c

  c := web.rxtime(500)

  if c == -1
    abort -1

  return c

PUB keyHandler | i, in
  ' this method runs in a seperate cog to handle keystrokes
  ' it also updates the typing window in the vga driver

  bSend := false
  bQuit := false
  bReboot := false

  i := 0
  keyBuffer[0] := 0

  vga.printChatStr(string("(status)"), @keyBuffer)

  repeat
    in := key.key
    if in
      if in == $CB                ' ESC = quit
        bQuit := true
        next
      if in == $C8                ' backspace
        if i > 0
          --i
      elseif in == $0D            ' enter
        ' send message
        bSend := true
        repeat while bSend        ' wait until sent
        i := 0                    ' empty text buffer
      elseif i < 254
        keyBuffer[i++] := in
      keyBuffer[i] := 0

    vga.printChatStr(@myChannel, @keyBuffer)

PUB ircClient(port) | inlen, i, j, nickstr, chanstr, msgstr, keepalive

  vga.invstr(string("Connecting to irc server...", 13))
  sock.connect(myServerIP, myServerPort)

  sock.resetBuffers

  sock.waitConnectTimeout(2000)
```

```
  if sock.isConnected

    vga.invstr(string("Connected!", 13))

    delay_ms(5000)

    vga.invstr(string("Registering.", 13))

    sock.str(string("PASS "))
    sock.str(@myPass)
    sock.str(string(13,10))

    reinitJoin

    vga.out(13)

    delay_ms(1000)

    keepalive := cnt
    repeat

      inlen := rxLine(@buffer, 512)
      if inlen > -1
        ' we received something from the server
        if (i := strings.indexOf(@buffer, string("PRIVMSG"))) <> -1
          ' chat string
          chanstr := @buffer + i + 8                                    ' seek past PRIVMSG
          j := strings.indexOf(chanstr, string(" "))
          byte[chanstr][j] := 0

          ' message string
          msgstr := chanstr + strsize(chanstr) + 2

          ' nick string
          nickstr := @buffer + 1
          i := strings.indexOf(nickstr, string("!"))
          byte[nickstr][i] := 0

          ' check for CTCP
          i := strsize(msgstr)
          if byte[msgstr] == 1 AND byte[msgstr][i - 1] == 1
            ' it's a CTCP msg
            byte[msgstr][i - 1] := 0                                    ' move string end up one spot
            msgstr++                                                    ' seek past the CTCP byte
            printCTCPStr(nickstr, msgstr)

            if strcomp(msgstr, string("VERSION"))
              ' version string, reply with our cool version info
              sock.str(string("NOTICE "))
              sock.str(nickstr)
              sock.str(string(" :VERSION PropIRC 1.0.0 [P8X32A/80MHz] <http://harrisonpham.com/PropTCP/>",13,10))

          else
            printChatStr(chanstr, nickstr, msgstr)

        elseif (i := strings.indexOf(@buffer, string("JOIN"))) <> -1
          chanstr := @buffer + i + 6                                    ' seek to message content

          nickstr := @buffer + 1
          i := strings.indexOf(nickstr, string("!"))
          byte[nickstr][i] := 0

          printTime
          vga.invstr(nickstr)
          vga.invstr(string(" has joined "))
          vga.invstr(chanstr)
          vga.out(13)

        elseif (i := strings.indexOf(@buffer, string("PART"))) <> -1
          chanstr := @buffer + i + 5                                    ' seek past PART
          j := strings.indexOf(chanstr, string(" "))
          byte[chanstr][j] := 0

          nickstr := @buffer + 1
          i := strings.indexOf(nickstr, string("!"))
          byte[nickstr][i] := 0

          printTime
```

```
        vga.invstr(nickstr)
        vga.invstr(string(" has left "))
        vga.invstr(chanstr)
        vga.out(13)

      elseif (i := strings.indexOf(@buffer, string("NICK"))) <> -1
        msgstr := @buffer + i + 6                                          ' seek to message content

        nickstr := @buffer + 1
        i := strings.indexOf(nickstr, string("!"))
        byte[nickstr][i] := 0

        printTime
        vga.invstr(nickstr)
        vga.invstr(string(" changed nickname to "))
        vga.invstr(msgstr)
        vga.out(13)

        if strcomp(@myNick, nickstr)
          bytemove(@myNick, msgstr, strsize(msgstr) + 1)

      elseif strings.indexOf(@buffer, string(":Nickname is already in use.")) <> -1
        ' we need a new nickname
        vga.invstr(string("ERROR: Nickname in use.  Trying new nickname.",13))

        i := strsize(@myNick)
        if i == 15
          ' we are out of nickname space
          ' so just quit
          quit
        myNick[i] := "_"
        myNick[i+1] := 0

        reinitJoin

      elseif strings.indexOf(@buffer, string(":Register first.")) <> -1

        reinitJoin

      elseif strings.indexOf(@buffer, string("PING :")) == 0
        ' reply with PONG

        buffer[1] := "O"
        sock.str(@buffer)

    if bSend
      ' check to see if they are trying to set settings
      printChatStr(@myChannel, @myNick, @keyBuffer)

      if strings.indexOf(@keyBuffer, string("/set")) == 0
        ' settings request
        msgstr := @keyBuffer[4]

        if strsize(msgstr) > 0
          msgstr++

        if strsize(msgstr) == 0
          ' no setting key given, print all settings
          showSettingIpPort(string("server="), myServerIp, myServerPort)
          showSetting(string("user="), @myUser)
          showSetting(string("pass="), @myPass)
          showSetting(string("nick="), @myNick)
          showSetting(string("channel="), @myChannel)
          showSettingDec(string("tzoff="), myTimeOffset)

        elseif strings.indexOf(msgstr, string("server")) == 0
          msgstr += strings.indexOf(msgstr, string(" ")) + 1
          if strsize(msgstr) > 8
            strToIpPort(msgstr, @myServerIp, @myServerPort)

        elseif strings.indexOf(msgstr, string("user")) == 0
          msgstr += strings.indexOf(msgstr, string(" ")) + 1
          if strsize(msgstr) < 15
            bytemove(@myUser, msgstr, strsize(msgstr) + 1)

        elseif strings.indexOf(msgstr, string("pass")) == 0
          msgstr += strings.indexOf(msgstr, string(" ")) + 1
          if strsize(msgstr) < 15
```

```
            bytemove(@myPass, msgstr, strsize(msgstr) + 1)

        elseif strings.indexOf(msgstr, string("nick")) == 0
          msgstr += strings.indexOf(msgstr, string(" ")) + 1
          if strsize(msgstr) < 15
            bytemove(@myNick, msgstr, strsize(msgstr) + 1)

        elseif strings.indexOf(msgstr, string("channel")) == 0
          msgstr += strings.indexOf(msgstr, string(" ")) + 1
          if strsize(msgstr) < 31
            bytemove(@myChannel, msgstr, strsize(msgstr) + 1)

        elseif strings.indexOf(msgstr, string("tzoff")) == 0
          msgstr += strings.indexOf(msgstr, string(" ")) + 1
          myTimeOffset := strToDec(msgstr)

        else
          printTime
          vga.invstr(string("Error: Unknown settings key.",13))

        printTime
        vga.invstr(string("Notice: You must do /save and /reboot to commit and load settings.",13))

      elseif strings.indexOf(@keyBuffer, string("/save")) == 0
        commitSettings
        printTime
        vga.invstr(string("Your settings have been saved.  /reboot to load and run settings.",13))

      elseif strings.indexOf(@keyBuffer, string("/load")) == 0
        restoreSettings
        printTime
        vga.invstr(string("Your settings have been reloaded.",13))

      elseif strings.indexOf(@keyBuffer, string("/reboot")) == 0
        bQuit := true
        bReboot := true

      elseif strings.indexOf(@keyBuffer, string("/quit")) == 0
        ' quit / disconnect
        sock.str(string("QUIT :"))
        sock.str(@keyBuffer[5])
        sock.str(@crlf)
        bQuit := true
      elseif strings.indexOf(@keyBuffer, string("/msg")) == 0 AND strsize(@keyBuffer) > 7
        ' directed PRIVMSG
        if (i := strings.indexOf(@keyBuffer[5], string(" "))) <> -1
          keyBuffer[i + 5] := 0
          sock.str(string("PRIVMSG "))
          sock.str(@keyBuffer[5])
          sock.str(string(" :"))
          sock.str(@keyBuffer[i + 6])
          sock.str(@crlf)
      elseif keyBuffer[0] == "/"
        ' IRC commands
        sock.str(@keyBuffer[1])
        sock.str(@crlf)
      else
        ' PRIVMSG message
        sock.str(string("PRIVMSG "))
        sock.str(@myChannel)
        sock.str(string(" :"))
        sock.str(@keyBuffer)
        sock.str(@crlf)
      bSend := false

    if bQuit
      quit

    if (keepalive - cnt) < 0
      sock.str(string("PING LAG"))
      sock.dec(||keepalive)
      sock.str(@crlf)
      keepalive := (clkfreq * 25) + cnt

  vga.invstr(string("Disconnecting...",13))

  sock.str(string("QUIT :Leaving",13,10))
```

```
    delay_ms(1500)

    sock.close

    delay_ms(2000)

    vga.invstr(string("Disconnected",13))

    if bReboot
      reboot

    repeat
      waitcnt(0)                ' sleep

PRI strToIpPort(str, ip, port) | octet
  ' extracts the IP and PORT from a string

  long[ip] := 0
  word[port] := 0
  octet := 3
  repeat while octet => 0
    case byte[str]
      "0".."9":
        byte[ip][octet] := (byte[ip][octet] * 10) + (byte[str] - "0")
      ".":
        octet--
      ":":
        quit
      other:
        return false
    str++
  if octet <> 0
    return false
  if byte[str++] == ":"
    repeat while byte[str] <> 0
      if byte[str] => "0" and byte[str] =< "9"
        word[port] := (word[port] * 10) + (byte[str] - "0")
      else
        return false
      str++

  return true

PRI strToDec(str) : val | isneg
  val := 0
  isneg := false
  if byte[str] == "-"
    isneg := true
    str++
  repeat strsize(str)
    if byte[str] => "0" and byte[str] =< "9"
      val := (val * 10) + (byte[str] - "0")
    else
      quit
    str++
  if isneg
    val *= -1

PRI showSetting(keystr, valuestr)
  printTime
  vga.invstr(keystr)
  vga.str(valuestr)
  vga.out(13)

PRI showSettingIpPort(keystr, ip, port)
  printTime
  vga.invstr(keystr)
  printIpPort(ip, port)
  vga.out(13)

PRI showSettingDec(keystr, value)
  printTime
  vga.invstr(keystr)
  vga.dec(value)
  vga.out(13)

PRI printIpPort(ip, port) | i
  repeat i from 3 to 1
```

```
    vga.dec(byte[@ip][i])
    vga.out("._")
  vga.dec(byte[@ip][0])
  vga.out(":")
  vga.dec(port)

PRI cleanStr(str)
  repeat strsize(str)
    if byte[str] =< 13
      byte[str] := " "
    str++

PRI dec2(val)
  vga.out(val / 10 + "0")
  vga.out(val // 10 + "0")

PRI printTime | dtret
  dtret := dt.timeETV(rtc.getTimestamp + myTimeOffset)

  vga.out("[")
  dec2((dtret & $FF0000) >> 16)
  vga.out(":")
  dec2((dtret & $FF00) >> 8)
  vga.out(":")
  dec2(dtret & $FF)
  vga.str(string("] "))

PRI printChatStr(chanstr, nickstr, msgstr)
  printTime
  cleanStr(msgstr)
  vga.str(chanstr)
  vga.out(" ")
  vga.invstr(nickstr)
  vga.str(string(": "))
  vga.str(msgstr)
  vga.out(13)

PRI printCTCPStr(nickstr, msgstr)
  printTime
  cleanStr(msgstr)
  vga.out(">")
  vga.invstr(nickstr)
  vga.out("<")
  vga.str(string(": "))
  vga.str(msgstr)
  vga.out(13)

PRI reinitJoin

  sock.str(string("NICK "))
  sock.str(@myNick)
  sock.str(string(13,10))

  sock.str(string("USER "))
  sock.str(@myUser)
  sock.tx(" ")
  sock.str(@myUser)
  sock.tx(" ")
  sock.str(@myUser)
  sock.str(string(" :PropTCP-IRC User",13,10))

  sock.str(string("JOIN "))
  sock.str(@myChannel)
  sock.str(string(13,10))

PRI rxcheck : in
  '' receives a character from the tcp socket
  '' non-blocking

  in := sock.rxcheck
  if in > -1
    vga.printDbg(in)

PRI rxtime(ms) : in
  '' receives a character from the tcp socket
  '' blocks for ms milliseconds

  in := sock.rxtime(ms)
```

```
  if in > -1
    vga.printDbg(in)

PRI rx : in
  '' receives / waits for a character from the tcp socket
  '' blocking

  repeat until (in := rxcheck) > -1

PRI rxline(strptr, len) | in, i
  '' receives an entire line (up to a CR), skips LFs
  '' returns -1 on emtpy buffer, or received length
  '' blocking

  in := rxcheck
  if in < 0
    return -1

  len--

  i := 0
  repeat
    if in == 13 or in == -1
      quit
    elseif in <> 10
      byte[strptr][i++] := in
      if i => len
        quit
    in := rxtime(250)

  byte[strptr][i] := 0

  return i

PRI delay_ms(Duration)
  waitcnt(((clkfreq / 1_000 * Duration - 3932)) + cnt)
```

api_telnet_serial.spin

```
{{
  PropTCP Sockets - FullDuplexSerial API Layer
  --------------------------------------------

  Copyright (C) 2007-2008 Harrison Pham

  This file is part of PropTCP.

  PropTCP is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 3 of the License, or
  (at your option) any later version.

  PropTCP is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
}}

OBJ
  tcp : "driver_socket"
  'dbg : "vga_irc_text"
  'dbg : "vga_text"

VAR
  long handle
  word listenport
  byte listening

PUB start(cs, sck, si, so, int, xtalout, macptr, ipconfigptr)

  tcp.start(cs, sck, si, so, int, xtalout, macptr, ipconfigptr)

  'dbg.start(16)

PUB stop

  tcp.stop

PUB trace
  {dbg.str(string(13,"D "))
  dbg.dec(handle)
  dbg.out(" ")
  dbg.dec(tcp.getSocketState(handle))}

PUB connect(ipaddr, remoteport)

  listening := false
  return (handle := tcp.connect(ipaddr, remoteport))

PUB listen(port)

  listenport := port
  listening := true
  handle := tcp.listen(listenport)

  {dbg.str(string(13,"L "))
  dbg.dec(handle)
  dbg.out(" ")
  dbg.dec(tcp.getSocketState(handle))}

  if handle == -1
    'dbg.str(string("Out of sockets!",13))
    abort -1

  'dbg.str(string(13,"socket: "))
  'dbg.dec(handle)
  'dbg.str(string(" "))

  return handle
```

```
PUB isConnected

  return tcp.isConnected(handle)

PUB resetBuffers

  tcp.resetBuffers(handle)

PUB waitConnectTimeout(ms) | t

  t := cnt
  repeat until isConnected or (((cnt - t) / (clkfreq / 1000)) > ms)

PUB close

  'dbg.str(string(13,"FORCE CLOSE",13))
  {dbg.str(string(13,"C "))
  dbg.dec(handle)
  dbg.out(" ")
  dbg.dec(tcp.getSocketState(handle))}

  tcp.close(handle)

PUB rxflush

  repeat while rxcheck => 0

PUB rxcheck

  if listening
    ifnot tcp.isValidHandle(handle)
      'dbg.str(string(13,"DEATH RX",13))
      listen(listenport)
  else
    ifnot tcp.isConnected(handle)
      abort -1

  return tcp.readByteNonBlocking(handle)

PUB rxtime(ms) : rxbyte | t

  t := cnt
  repeat until (rxbyte := rxcheck) => 0 or (cnt - t) / (clkfreq / 1000) > ms

  'if rxbyte == -1
    'dbg.str(string(13,"TIMEOUT",13))

PUB rx : rxbyte

  repeat while (rxbyte := rxcheck) < 0

PUB txflush

  'dbg.str(string(13,"FLUSH",13))

  tcp.flush(handle)

PUB txcheck(txbyte)

  if listening
    ifnot tcp.isValidHandle(handle)
      'dbg.str(string(13,"DEATH TX",13))
      listen(listenport)
  'else
  '  ifnot tcp.isConnected(handle)
  '    abort -1

  ifnot tcp.isConnected(handle)        'TEMP TEMP TEMP!!! FIX THIS MAKE IT BETTER!!!
    abort -1

  return tcp.writeByteNonBlocking(handle, txbyte)

PUB tx(txbyte)

  repeat while txcheck(txbyte) < 0

PUB txdata(ptr, len)
```

```
  tcp.writeData(handle, ptr, len)

PUB str(stringptr)

  {repeat strsize(stringptr)
    tx(byte[stringptr++])}
  txdata(stringptr, strsize(stringptr))

PUB dec(value) | i

'' Print a decimal number

  if value < 0
    -value
    tx("-")

  i := 1_000_000_000

  repeat 10
    if value => i
      tx(value / i + "0")
      value //= i
      result~~
    elseif result or i == 1
      tx("0")
    i /= 10


PUB hex(value, digits)

'' Print a hexadecimal number

  value <<= (8 - digits) << 2
  repeat digits
    tx(lookupz((value <-= 4) & $F : "0"..."9", "A"..."F"))


PUB bin(value, digits)

'' Print a binary number

  value <<= 32 - digits
  repeat digits
    tx((value <-= 1) & 1 + "0")
```

driver_socket.spin

```
{{
  Ethernet TCP/IP Socket Layer Driver (IPv4)
  ------------------------------------------

  Copyright (C) 2006-2008 Harrison Pham

  This file is part of PropTCP.

  PropTCP is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 3 of the License, or
  (at your option) any later version.

  PropTCP is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
}}

CON
' ****************************************
' **      Versioning Information       **
' ****************************************
  version    = 2                 ' major version
  release    = 0                 ' minor version
  apiversion = 4                 ' api compatibility version

' ****************************************
' **     User Definable Settings       **
' ****************************************
  sNumSockets   = 2              ' number of concurrent sockets
  buffer_length = 256            ' per socket buffer size (2, 4, 8, 16, 32, 64, 128 , ..., 65536)

' *** End of user definable settings, don't edit anything below this line!!!
' *** All IP/MAC settings are defined by calling the start(...) method

OBJ
  nic : "driver_enc28j60"

  'ser : "SerialMirror"
  'stk : "Stack Length"

CON
' ****************************************
' **   Socket Constants and Offsets    **
' ****************************************
' The following is an 'array' that represents all the socket handle data (with respect to the remote host)
' longs first, then words, then bytes (for alignment)
'
'         4 bytes - (1 long ) my sequence number
'         4 bytes - (1 long ) my acknowledgement number
'         4 bytes - (1 long ) src ip
'         4 bytes - (1 long ) socket state timer
'         2 bytes - (1 word ) src port
'         2 bytes - (1 word ) dst port
'         2 bytes - (1 word ) last window
'         1 byte  - (1 byte ) conn state
'         6 bytes - (6 bytes) src mac address
'         1 byte  - (1 byte ) handle index (MUST be the last item in the array)
' total: 30 bytes

  sSocketBytes = 32              ' MUST BE MULTIPLE OF 4 (long aligned) set this to total socket state data size

' Offsets for socket status arrays
  sMySeqNum  = 0
  sMyAckNum  = 4
  sSrcIp     = 8
  sTime      = 12
  sSrcPort   = 16
  sDstPort   = 18
  sLastWin   = 20
```

```
  sConState  = 22
  sSrcMac    = 23
  sSockIndex = 29               ' sSockIndex MUST be the last item in the array, otherwise it won't work right

' Socket states (user should never touch these)
  SCLOSED          = 0          ' closed, handle not used
  SLISTEN          = 1          ' listening, in server mode
  SSYNSENT         = 2          ' SYN sent, server mode, waits for ACK
  SSYNSENTCL       = 3          ' SYN sent, client mode, waits for SYN+ACK
  SESTABLISHED     = 4          ' established connection (either SYN+ACK, or ACK+Data)
  SCLOSING         = 5          ' connection is being forced closed by code
  SCLOSING2        = 6          ' closing, we are waiting for a fin now
  SFORCECLOSE      = 7          ' force connection close (just RSTs, no waiting for FIN or anything)
  SCONNECTINGARP1  = 8          ' connecting, next step: send arp request
  SCONNECTINGARP2  = 9          ' connecting, next step: arp request sent, waiting for response
  SCONNECTINGARP2G = 10         ' connecting, next step: arp request sent, waiting for response [GATEWAY REQUEST]
  SCONNECTING      = 11         ' connecting, next step: got mac address, send SYN

' ****************************************
' **    Circular Buffer Constants      **
' ****************************************
  buffer_mask = buffer_length - 1
' window_size = buffer_length / 8

' ****************************************
' **  TCP State Management Constants   **
' ****************************************
  TIMEOUTMS     = 500               ' (milliseconds) socket operation timeout, to prevent stalled states

  EPHPORTSTART  = 49152             ' ephemeral port start
  EPHPORTEND    = 65535             ' end

  IPMTU         = 1000              ' max transfer unit size (<1500 is good)

DAT
' ****************************************
' **          Global Variables         **
' ****************************************
  cog           long 0                                  ' cog index (for stopping / starting)
  stack         long 0[128]                             ' stack for new cog (currently ~74 longs, using 128 for
  expansion)

  mac_ptr       long 0                                  ' mac address pointer

  pkt           long 0                                  ' memory address of packet start

  pkt_id        long 0                                  ' packet fragmentation id
  pkt_isn       long 0                                  ' packet initial sequence number

  ip_ephport    word 0                                  ' packet ephemeral port number (49152 to 65535)

  pkt_count     byte 0                                  ' packet count

' ****************************************
' **        IP Address Defaults        **
' ****************************************
  ' NOTE: All of the MAC/IP variables here contain default values that will
  '       be used if override values are not provided as parameters in start().
                long                                    ' long alignment for addresses
  ip_addr       byte    10, 10, 1, 4                    ' device's ip address
  ip_subnet     byte    255, 255, 255, 0                ' network subnet
  ip_gateway    byte    10, 10, 1, 254                  ' network gateway (router)
  ip_dns        byte    10, 10, 1, 254                  ' network dns

' ****************************************
' **        Socket Data Arrays         **
' ****************************************
                long                                    ' long align the socket state data
  sSockets      byte      0[sSocketBytes * sNumSockets]   ' socket data array space (pre allocate)

' ****************************************
' **      Circular Buffer Arrays       **
' ****************************************
  rx_head       word      0[sNumSockets]                ' rx head array
  rx_tail       word      0[sNumSockets]                ' rx tail array
  tx_head       word      0[sNumSockets]                ' tx head array
  tx_tail       word      0[sNumSockets]                ' tx tail array
```

```
  tx_buffer     byte      0[buffer_length * sNumSockets]  ' transmit buffer space

  rx_buffer     byte      0[buffer_length * sNumSockets]  ' receive buffer space

PUB start(cs, sck, si, so, int, xtalout, macptr, ipconfigptr) | socketIdx
'' Start the TCP/IP Stack (requires 2 cogs)
'' Only call this once, otherwise you will get conflicts
''    macptr      = HUB memory pointer (address) to 6 contiguous mac address bytes
''    ipconfigptr = HUB memory pointer (address) to ip configuration block (16 bytes)
''                  Must be in order: ip_addr, ip_subnet, ip_gateway, ip_dns

  stop
  'stk.Init(@stack, 128)

  ' zero socket data arrays (clean up any dead stuff from previous instance)
  bytefill(@sSockets, 0, constant(sSocketBytes * sNumSockets))

  ' reset buffer pointers, zeros a contigous set of bytes, starting at rx_head
  wordfill(@rx_head, 0, constant(sNumSockets * 4))

  ' setup pointer address values and indexing values
  repeat socketIdx from 0 to constant(sNumSockets - 1)
    BYTE[@sSockets][(socketIdx * sSocketBytes) + sSockIndex] := socketIdx       ' set socket indicies

  ' start new cog with tcp stack
  cog := cognew(engine(cs, sck, si, so, int, xtalout, macptr, ipconfigptr), @stack) + 1

PUB stop
'' Stop the driver

  if cog
    nic.stop                    ' stop nic driver (kills spi engine)
    cogstop(cog~ - 1)           ' stop the tcp engine

PRI engine(cs, sck, si, so, int, xtalout, macptr, ipconfigptr) | i

  ' Start the ENC28J60 driver in a new cog
  nic.start(cs, sck, si, so, int, xtalout, macptr)                            ' init the nic

  if ipconfigptr > -1                                                         ' init ip configuration
    bytemove(@ip_addr, ipconfigptr, 16)

  mac_ptr := nic.get_mac_pointer                                              ' get the local mac address pointer

  pkt := nic.get_packetpointer                                                ' get the packet pointer

  ip_ephport := EPHPORTSTART                                                  ' set initial ephemeral port number
  (might want to random seed this later)

  i := 0
  nic.banksel(nic#EPKTCNT)                                                    ' select packet count bank
  repeat
    pkt_count := nic.rd_cntlreg(nic#EPKTCNT)
    if pkt_count > 0
      service_packet                                                          ' handle packet
      nic.banksel(nic#EPKTCNT)                                                ' re-select the packet count bank

    ++i
    if i > 10                                                                 ' perform send tick
      tick_tcpsend                                                            '  occurs every 10 cycles, since
  incoming packets more important
      i := 0
      nic.banksel(nic#EPKTCNT)                                                ' re-select the packet count bank

PRI service_packet

  ' lets process this frame
  nic.get_frame

  ' check for arp packet type (highest priority obviously)
  if BYTE[pkt][enetpacketType0] == $08 AND BYTE[pkt][enetpacketType1] == $06
    if BYTE[pkt][constant(arp_hwtype + 1)] == $01 AND BYTE[pkt][arp_prtype] == $08 AND BYTE[pkt][constant(arp_prtype
  + 1)] == $00 AND BYTE[pkt][arp_hwlen] == $06 AND BYTE[pkt][arp_prlen] == $04
      if BYTE[pkt][arp_tipaddr] == ip_addr[0] AND BYTE[pkt][constant(arp_tipaddr + 1)] == ip_addr[1] AND
  BYTE[pkt][constant(arp_tipaddr + 2)] == ip_addr[2] AND BYTE[pkt][constant(arp_tipaddr + 3)] == ip_addr[3]
        case BYTE[pkt][constant(arp_op + 1)]
          $01 : handle_arp
          $02 : handle_arpreply
```

```
        '++count_arp
  else
    if BYTE[pkt][enetpacketType0] == $08 AND BYTE[pkt][enetpacketType1] == $00
      if BYTE[pkt][ip_destaddr] == ip_addr[0] AND BYTE[pkt][constant(ip_destaddr + 1)] == ip_addr[1] AND
  BYTE[pkt][constant(ip_destaddr + 2)] == ip_addr[2] AND BYTE[pkt][constant(ip_destaddr + 3)] == ip_addr[3]
        case BYTE[pkt][ip_proto]
          'PROT_ICMP : 'handle_ping
                       'ser.str(stk.GetLength(0, 0))
                       '++count_ping
           PROT_TCP :  \handle_tcp                                          ' handles abort out of tcp handlers
  (no socket found)
                       '++count_tcp
          'PROT_UDP :  ++count_udp

' ********************************
' ** Protocol Receive Handlers **
' ********************************
PRI handle_arp | i
  nic.start_frame

  ' destination mac address
  repeat i from 0 to 5
    nic.wr_frame(BYTE[pkt][enetpacketSrc0 + i])

  ' source mac address
  repeat i from 0 to 5
    nic.wr_frame(BYTE[mac_ptr][i])

  nic.wr_frame($08)             ' arp packet
  nic.wr_frame($06)

  nic.wr_frame($00)             ' 10mb ethernet
  nic.wr_frame($01)

  nic.wr_frame($08)             ' ip proto
  nic.wr_frame($00)

  nic.wr_frame($06)             ' mac addr len
  nic.wr_frame($04)             ' proto addr len

  nic.wr_frame($00)             ' arp reply
  nic.wr_frame($02)

  ' write ethernet module mac address
  repeat i from 0 to 5
    nic.wr_frame(BYTE[mac_ptr][i])

  ' write ethernet module ip address
  repeat i from 0 to 3
    nic.wr_frame(ip_addr[i])

  ' write remote mac address
  repeat i from 0 to 5
    nic.wr_frame(BYTE[pkt][enetpacketSrc0 + i])

  ' write remote ip address
  repeat i from 0 to 3
    nic.wr_frame(BYTE[pkt][arp_sipaddr + i])

  return nic.send_frame

PRI handle_arpreply | handle, handle_addr, ip, found
  ' Gets arp reply if it is a response to an ip we have

  ip := (BYTE[pkt][arp_sipaddr] << 24) + (BYTE[pkt][constant(arp_sipaddr + 1)] << 16) +
  (BYTE[pkt][constant(arp_sipaddr + 2)] << 8) + (BYTE[pkt][constant(arp_sipaddr + 3)])

  found := false
  if ip == conv_endianlong(LONG[@ip_gateway])
    ' find a handle that wants gateway mac
    repeat handle from 0 to constant(sNumSockets - 1)
      handle_addr := @sSockets + (sSocketBytes * handle)
      if BYTE[handle_addr + sConState] == SCONNECTINGARP2G
        found := true
        quit
  else
    ' find the one that wants this arp
    repeat handle from 0 to constant(sNumSockets - 1)
```

```
      handle_addr := @sSockets + (sSocketBytes * handle)
      if BYTE[handle_addr + sConState] == SCONNECTINGARP2
        if LONG[handle_addr + sSrcIp] == conv_endianlong(ip)
          found := true
          quit

  if found
    bytemove(handle_addr + sSrcMac, pkt + arp_shaddr, 6)
    BYTE[handle_addr + sConState] := SCONNECTING

'PRI handle_ping
  ' Not implemented yet (save on space!)

PRI handle_tcp | i, ptr, handle, handle_addr, srcip, dstport, srcport, datain_len, head
  ' Handles incoming TCP packets

  srcip := BYTE[pkt][ip_srcaddr] << 24 + BYTE[pkt][constant(ip_srcaddr + 1)] << 16 + BYTE[pkt][constant(ip_srcaddr +
  2)] << 8 + BYTE[pkt][constant(ip_srcaddr + 3)]
  dstport := BYTE[pkt][TCP_destport] << 8 + BYTE[pkt][constant(TCP_destport + 1)]
  srcport := BYTE[pkt][TCP_srcport] << 8 + BYTE[pkt][constant(TCP_srcport + 1)]

  handle_addr := find_socket(srcip, dstport, srcport)   ' if no sockets avail, it will abort out of this function

  handle := BYTE[handle_addr + sSockIndex]

  ' at this point we assume we have an active socket, or a socket available to be used
  datain_len := ((BYTE[pkt][ip_pktlen] << 8) + BYTE[pkt][constant(ip_pktlen + 1)]) - ((BYTE[pkt][ip_vers_len] & $0F)
  * 4) - (((BYTE[pkt][TCP_hdrlen] & $F0) >> 4) * 4)

  if (BYTE[handle_addr + sConState] == SSYNSENT OR BYTE[handle_addr + sConState] == SESTABLISHED) AND
  (BYTE[pkt][TCP_hdrflags] & TCP_ACK) > 0 AND datain_len > 0
    ' ACK, without SYN, with data

    ' set socket state, established session
    BYTE[handle_addr + sConState] := SESTABLISHED

    i := BYTE[pkt][constant(TCP_seqnum + 3)] << 24 + BYTE[pkt][constant(TCP_seqnum + 2)] << 16 +
  BYTE[pkt][constant(TCP_seqnum + 1)] << 8 + BYTE[pkt][TCP_seqnum]
    if LONG[handle_addr + sMyAckNum] == i
      if datain_len =< (buffer_mask - ((rx_head[handle] - rx_tail[handle]) & buffer_mask))
        ' we have buffer space
        ptr := @rx_buffer + (handle * buffer_length)
        if (datain_len + rx_head[handle]) > buffer_length
          bytemove(ptr + rx_head[handle], @BYTE[pkt][TCP_data], buffer_length - rx_head[handle])
          bytemove(ptr, @BYTE[pkt][TCP_data] + (buffer_length - rx_head[handle]), datain_len - (buffer_length -
  rx_head[handle]))
        else
          bytemove(ptr + rx_head[handle], @BYTE[pkt][TCP_data], datain_len)
        rx_head[handle] := (rx_head[handle] + datain_len) & buffer_mask
      else
        datain_len := 0
      {' copy data to buffer
      ptr := @rx_buffer + (handle * buffer_length)
      head := WORD[@rx_head][handle]
      repeat i from 0 to datain_len - 1
        if (WORD[@rx_tail][handle] <> (head + 1) & buffer_mask)
          BYTE[ptr][head] := BYTE[pkt][TCP_data + i]
          head := (head + 1) & buffer_mask
        else
          ' so we ran out of buffer space...
          ' don't update the rx_head, just send a dup ack and quit
          datain_len := 0
          quit
      ' goody, we had buffer space, update the head and continue
      WORD[@rx_head][handle] := head}
    else
      ' we had a bad ack number, meaning lost or out of order packet
      ' the remote host will just have to handle it (aka retransmit)
      datain_len := 0

    ' recalculate ack number
    LONG[handle_addr + sMyAckNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMyAckNum]) + datain_len)

    ' ACK response
    build_ipheaderskeleton(handle_addr)
    build_tcpskeleton(handle_addr, TCP_ACK)
    send_tcpfinal(handle_addr, 0)
```

```
elseif (BYTE[handle_addr + sConState] == SSYNSENTCL) AND (BYTE[pkt][TCP_hdrflags] & TCP_SYN) > 0 AND
(BYTE[pkt][TCP_hdrflags] & TCP_ACK) > 0
  ' We got a server response, so we ACK it

  bytemove(handle_addr + sMySeqNum, pkt + TCP_acknum, 4)
  bytemove(handle_addr + sMyAckNum, pkt + TCP_seqnum, 4)

  LONG[handle_addr + sMyAckNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMyAckNum]) + 1)

  ' ACK response
  build_ipheaderskeleton(handle_addr)
  build_tcpskeleton(handle_addr, TCP_ACK)
  send_tcpfinal(handle_addr, 0)

  ' set socket state, established session
  BYTE[handle_addr + sConState] := SESTABLISHED

elseif (BYTE[handle_addr + sConState] == SLISTEN) AND (BYTE[pkt][TCP_hdrflags] & TCP_SYN) > 0
  ' Reply to SYN with SYN + ACK

  ' copy mac address so we don't have to keep an ARP table
  bytemove(handle_addr + sSrcMac, pkt + enetpacketSrc0, 6)

  ' copy ip, port data
  bytemove(handle_addr + sSrcIp, pkt + ip_srcaddr, 4)
  bytemove(handle_addr + sSrcPort, pkt + TCP_srcport, 2)
  bytemove(handle_addr + sDstPort, pkt + TCP_destport, 2)

  ' get updated ack numbers
  bytemove(handle_addr + sMyAckNum, pkt + TCP_seqnum, 4)

  LONG[handle_addr + sMyAckNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMyAckNum]) + 1)
  LONG[handle_addr + sMySeqNum] := conv_endianlong(++pkt_isn)               ' Initial seq num (random)

  build_ipheaderskeleton(handle_addr)
  build_tcpskeleton(handle_addr, constant(TCP_SYN | TCP_ACK))
  send_tcpfinal(handle_addr, 0)

  ' incremement the sequence number for the next packet (it will be for an established connection)
  LONG[handle_addr + sMySeqNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMySeqNum]) + 1)

  ' set socket state, waiting for establish
  LONG[handle_addr + sTime] := cnt
  BYTE[handle_addr + sConState] := SSYNSENT

elseif (BYTE[handle_addr + sConState] == SESTABLISHED OR BYTE[handle_addr + sConState] == SCLOSING2) AND
(BYTE[pkt][TCP_hdrflags] & TCP_FIN) > 0
  ' Reply to FIN with RST

  ' get updated sequence and ack numbers (gaurantee we have correct ones to kill connection with)
  bytemove(handle_addr + sMySeqNum, pkt + TCP_acknum, 4)
  bytemove(handle_addr + sMyAckNum, pkt + TCP_seqnum, 4)

  'LONG[handle_addr + sMyAckNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMyAckNum]) + 1)

  build_ipheaderskeleton(handle_addr)
  build_tcpskeleton(handle_addr, TCP_RST)
  send_tcpfinal(handle_addr, 0)

  ' set socket state, now free
  BYTE[handle_addr + sConState] := SCLOSED

{elseif (BYTE[handle_addr + sConState] == SCLOSING2) AND (BYTE[pkt][TCP_hdrflags] & TCP_ACK) > 0
  ' the other side ACK'd our FIN, so let's just reset instead of negotiating another graceful FIN

  ' get updated sequence and ack numbers (gaurantee we have correct ones to kill connection with)
  bytemove(handle_addr + sMySeqNum, pkt + TCP_acknum, 4)
  bytemove(handle_addr + sMyAckNum, pkt + TCP_seqnum, 4)

  'LONG[handle_addr + sMyAckNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMyAckNum]) + 1)

  build_ipheaderskeleton(handle_addr)
  build_tcpskeleton(handle_addr, TCP_RST)
  send_tcpfinal(handle_addr, 0)

  ' set socket state, now free
  BYTE[handle_addr + sConState] := SCLOSED}
```

```
  elseif (BYTE[handle_addr + sConState] == SSYNSENT) AND (BYTE[pkt][TCP_hdrflags] & TCP_ACK) > 0
    ' if just an ack, and we sent a syn before, then it's established
    ' this just gives us the ability to send on connect
    BYTE[handle_addr + sConState] := SESTABLISHED

  elseif (BYTE[pkt][TCP_hdrflags] & TCP_RST) > 0
    ' Reset, reset states
    BYTE[handle_addr + sConState] := SCLOSED

PRI build_ipheaderskeleton(handle_addr) | hdrlen, hdr_chksum

  bytemove(pkt + ip_destaddr, handle_addr + sSrcIp, 4)                        ' Set destination address

  bytemove(pkt + ip_srcaddr, @ip_addr, 4)                                     ' Set source address

  bytemove(pkt + enetpacketDest0, handle_addr + sSrcMac, 6)                   ' Set destination mac address

  bytemove(pkt + enetpacketSrc0, mac_ptr, 6)                                  ' Set source mac address

  BYTE[pkt][enetpacketType0] := $08
  BYTE[pkt][constant(enetpacketType0 + 1)] := $00

  BYTE[pkt][ip_vers_len] := $45
  BYTE[pkt][ip_tos] := $00

  ++pkt_id

  BYTE[pkt][ip_id] := pkt_id >> 8                                             ' Used for fragmentation
  BYTE[pkt][constant(ip_id + 1)] := pkt_id

  BYTE[pkt][ip_frag_offset] := $40                                            ' Don't fragment
  BYTE[pkt][constant(ip_frag_offset + 1)] := 0

  BYTE[pkt][ip_ttl] := $80                                                    ' TTL = 128

  BYTE[pkt][ip_proto] := $06                                                  ' TCP protocol

PRI build_tcpskeleton(handle_addr, flags) | handle, size

  bytemove(pkt + TCP_srcport, handle_addr + sDstPort, 2)                      ' Source port
  bytemove(pkt + TCP_destport, handle_addr + sSrcPort, 2)                     ' Destination port

  bytemove(pkt + TCP_seqnum, handle_addr + sMySeqNum, 4)                      ' Seq Num
  bytemove(pkt + TCP_acknum, handle_addr + sMyAckNum, 4)                      ' Ack Num

  BYTE[pkt][TCP_hdrlen] := $50                                                ' Header length

  BYTE[pkt][TCP_hdrflags] := flags                                            ' TCP state flags

  ' we have to recalculate the window size often otherwise our stack
  ' might explode from too much data :(
  handle := BYTE[handle_addr + sSockIndex]
  size := (buffer_mask - ((rx_head[handle] - rx_tail[handle]) & buffer_mask))
  WORD[handle_addr + sLastWin] := size

  BYTE[pkt][TCP_window] := (size & $FF00) >> 8
  BYTE[pkt][constant(TCP_window + 1)] := size & $FF

  'BYTE[pkt][TCP_window] := constant((window_size & $FF00) >> 8)             ' Window size (max data that can be
  received before ACK must be sent)
  'BYTE[pkt][constant(TCP_window + 1)] := constant(window_size & $FF)         '  we use our buffer_length to ensure
  our buffer won't get overloaded
                                                                              '  may cause slowness so some people
  may want to use $FFFF on high latency networks

PRI send_tcpfinal(handle_addr, datalen) | i, tcplen, hdrlen, hdr_chksum

  LONG[handle_addr + sMySeqNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMySeqNum]) + datalen)
  ' update running sequence number

  tcplen := 40 + datalen                                                      ' real length = data + headers

  BYTE[pkt][ip_pktlen] := tcplen >> 8
  BYTE[pkt][constant(ip_pktlen + 1)] := tcplen

  ' calc ip header checksum
  BYTE[pkt][ip_hdr_cksum] := $00
  BYTE[pkt][constant(ip_hdr_cksum + 1)] := $00
```

```
  hdrlen := (BYTE[pkt][ip_vers_len] & $0F) * 4
  hdr_chksum := calc_chksum(@BYTE[pkt][ip_vers_len], hdrlen)
  BYTE[pkt][ip_hdr_cksum] := hdr_chksum >> 8
  BYTE[pkt][constant(ip_hdr_cksum + 1)] := hdr_chksum

  ' calc checksum
  BYTE[pkt][TCP_cksum] := $00
  BYTE[pkt][constant(TCP_cksum + 1)] := $00
  hdr_chksum := nic.chksum_add(@BYTE[pkt][ip_srcaddr], 8)
  hdr_chksum += BYTE[pkt][ip_proto]
  i := tcplen - ((BYTE[pkt][ip_vers_len] & $0F) * 4)
  hdr_chksum += i
  hdr_chksum += nic.chksum_add(@BYTE[pkt][TCP_srcport], i)
  hdr_chksum := calc_chksumfinal(hdr_chksum)
  BYTE[pkt][TCP_cksum] := hdr_chksum >> 8
  BYTE[pkt][constant(TCP_cksum + 1)] := hdr_chksum

  tcplen += 14
  if tcplen < 60
    tcplen := 60

  ' protect from buffer overrun
  if tcplen => nic#TX_BUFFER_SIZE
    return

  ' send the packet
  nic.start_frame

  nic.wr_block(pkt, tcplen)

  {repeat i from 0 to tcplen - 1
    nic.wr_frame(BYTE[pkt][i])}

  ' send the packet
  nic.send_frame

PRI find_socket(srcip, dstport, srcport) | handle, free_handle, handle_addr
  ' Search for socket, matches ip address, port states
  ' Returns handle address (start memory location of socket)
  '  If no matches, will abort with -1
  '  If supplied with srcip = 0 then will return free unused handle, aborts with -1 if none avail

  free_handle := -1
  repeat handle from 0 to constant(sNumSockets - 1)
    handle_addr := @sSockets + (sSocketBytes * handle)   ' generate handle address (mapped to memory)
    if BYTE[handle_addr + sConState] <> SCLOSED
      if (LONG[handle_addr + sSrcIp] == 0) OR (LONG[handle_addr + sSrcIp] == conv_endianlong(srcip))
         ' ip match, ip socket srcip = 0, then will try to match dst port (find listening socket)
         if (WORD[handle_addr + sDstPort] == conv_endianword(dstport)) AND (WORD[handle_addr + sSrcPort] == 0 OR
  WORD[handle_addr + sSrcPort] == conv_endianword(srcport))
            ' port match, will match port, if srcport = 0 then will match dstport only (find listening socket)
           return handle_addr
    elseif srcip == 0
      ' we only return a free handle if we are searching for srcip = 0 (just looking for free handle)
      free_handle := handle_addr     ' we found a free handle, may need this later

  if free_handle <> -1
    return free_handle
  else
    abort(-1)

' ******************************
' ** Transmit Buffer Handlers **
' ******************************
PRI tick_tcpsend | len, state, ptr, handle, handle_addr
  ' Check buffers for data to send (called in main loop)

  repeat handle from 0 to constant(sNumSockets - 1)
    handle_addr := @sSockets + (sSocketBytes * handle)
    state := BYTE[handle_addr + sConState]
    if state == SESTABLISHED OR state == SCLOSING
      ' Check to see if we have data to send, if we do, send it
      if tx_tail[handle] <> tx_head[handle]
        ' we have data to send, so send it!
        ptr := @tx_buffer + (handle * buffer_length)
        len := (tx_head[handle] - tx_tail[handle]) & buffer_mask
        if (len + tx_tail[handle]) > buffer_length
          bytemove(@BYTE[pkt][TCP_data], ptr + tx_tail[handle], buffer_length - tx_tail[handle])
```

```
          bytemove(@BYTE[pkt][TCP_data] + (buffer_length - tx_tail[handle]), ptr, len - (buffer_length -
  tx_tail[handle]))
        else
          bytemove(@BYTE[pkt][TCP_data], ptr + tx_tail[handle], len)
        tx_tail[handle] := (tx_tail[handle] + len) & buffer_mask

        build_ipheaderskeleton(handle_addr)
        build_tcpskeleton(handle_addr, constant(TCP_ACK | TCP_PSH))
        send_tcpfinal(handle_addr, len)

        {i := 0
        repeat while WORD[@tx_tail][handle] <> WORD[@tx_head][handle]
          ptr := @tx_buffer + (handle * buffer_length)
          BYTE[pkt][TCP_data + i] := BYTE[ptr][WORD[@tx_tail][handle]]
          WORD[@tx_tail][handle] := (WORD[@tx_tail][handle] + 1) & buffer_mask
          ++i
          if i => constant(nic#TX_BUFFER_SIZE - 100) ' avoid overflowing the buffer (we're using 100 here for
  testing, it will be fixed later)
            quit

        if i > 0
          build_ipheaderskeleton(handle_addr)
          build_tcpskeleton(handle_addr, constant(TCP_ACK | TCP_PSH))
          send_tcpfinal(handle_addr, i)}

      else
        ' we have no data to send, but we may have to update the window size
        if WORD[handle_addr + sLastWin] <> (buffer_mask - ((rx_head[handle] - rx_tail[handle]) & buffer_mask))
          ' update window size!
          build_ipheaderskeleton(handle_addr)
          build_tcpskeleton(handle_addr, TCP_ACK)
          send_tcpfinal(handle_addr, 0)

    if state == SCLOSING
      'LONG[handle_addr + sMyAckNum] := conv_endianlong(conv_endianlong(LONG[handle_addr + sMyAckNum]) + 1)

      build_ipheaderskeleton(handle_addr)
      build_tcpskeleton(handle_addr, constant(TCP_ACK | TCP_FIN))
      send_tcpfinal(handle_addr, 0)

      ' we now wait for the other side to terminate
      LONG[handle_addr + sTime] := cnt
      BYTE[handle_addr + sConState] := SCLOSING2

    elseif state == SCONNECTINGARP1
      ' We need to send an arp request

      arp_request_checkgateway(handle_addr)

    elseif state == SCONNECTING
      ' Yea! We got an arp response previously, so now we can send the SYN

      LONG[handle_addr + sMySeqNum] := conv_endianlong(++pkt_isn)
      LONG[handle_addr + sMyAckNum] := 0

      build_ipheaderskeleton(handle_addr)
      build_tcpskeleton(handle_addr, TCP_SYN)
      send_tcpfinal(handle_addr, 0)

      BYTE[handle_addr + sConState] := SSYNSENTCL

    elseif (state == SFORCECLOSE) OR ((state == SCLOSING2 OR state == SSYNSENT) AND ((cnt - LONG[handle_addr +
  sTime]) / (clkfreq / 1000) > TIMEOUTMS))
      ' Force close (send RST, and say the socket is closed!)

      ' This is triggered when any of the following happens:
      '  1 - we don't get a response to our SSYNSENT state
      '  2 - we get stuck in the SSCLOSING2 state

      build_ipheaderskeleton(handle_addr)
      build_tcpskeleton(handle_addr, TCP_RST)
      send_tcpfinal(handle_addr, 0)

      BYTE[handle_addr + sConState] := SCLOSED

PRI arp_request_checkgateway(handle_addr) | ip_ptr

  ip_ptr := handle_addr + sSrcIp
```

```
  if (BYTE[ip_ptr] & ip_subnet[0]) == (ip_addr[0] & ip_subnet[0]) AND (BYTE[ip_ptr + 1] & ip_subnet[1]) ==
  (ip_addr[1] & ip_subnet[1]) AND (BYTE[ip_ptr + 2] & ip_subnet[2]) == (ip_addr[2] & ip_subnet[2]) AND (BYTE[ip_ptr +
  3] & ip_subnet[3]) == (ip_addr[3] & ip_subnet[3])
    arp_request(BYTE[ip_ptr], BYTE[ip_ptr + 1], BYTE[ip_ptr + 2], BYTE[ip_ptr + 3])
    BYTE[handle_addr + sConState] := SCONNECTINGARP2
  else
    arp_request(ip_gateway[0], ip_gateway[1], ip_gateway[2], ip_gateway[3])
    BYTE[handle_addr + sConState] := SCONNECTINGARP2G

PRI arp_request(ip1, ip2, ip3, ip4) | i
  nic.start_frame

  ' destination mac address (broadcast mac)
  repeat i from 0 to 5
    nic.wr_frame($FF)

  ' source mac address (this device)
  repeat i from 0 to 5
    nic.wr_frame(BYTE[mac_ptr][i])

  nic.wr_frame($08)             ' arp packet
  nic.wr_frame($06)

  nic.wr_frame($00)             ' 10mb ethernet
  nic.wr_frame($01)

  nic.wr_frame($08)             ' ip proto
  nic.wr_frame($00)

  nic.wr_frame($06)             ' mac addr len
  nic.wr_frame($04)             ' proto addr len

  nic.wr_frame($00)             ' arp request
  nic.wr_frame($01)

  ' source mac address (this device)
  repeat i from 0 to 5
    nic.wr_frame(BYTE[mac_ptr][i])

  ' source ip address (this device)
  repeat i from 0 to 3
    nic.wr_frame(ip_addr[i])

  ' unknown mac address area
  repeat i from 0 to 5
    nic.wr_frame($00)

  ' figure out if we need router arp request or host arp request
  ' this means some subnet masking

  ' dest ip address
  nic.wr_frame(ip1)
  nic.wr_frame(ip2)
  nic.wr_frame(ip3)
  nic.wr_frame(ip4)

  ' send the request
  return nic.send_frame

' ********************************
' ** IP Packet Helpers (Calcs) **
' ********************************
PRI calc_chksum(packet, hdrlen) : chksum
  ' Calculates IP checksums
  ' packet = pointer to IP packet
  ' returns: chksum
  ' http://www.geocities.com/SiliconValley/2072/bit33.txt
  'chksum := calc_chksumhalf(packet, hdrlen)
  chksum := nic.chksum_add(packet, hdrlen)
  chksum := calc_chksumfinal(chksum)

PRI calc_chksumfinal(chksumin) : chksum
  ' Performs the final part of checksums
  chksum := (chksumin >> 16) + (chksumin & $FFFF)
  chksum := (!chksum) & $FFFF

{PRI calc_chksumhalf(packet, hdrlen) : chksum
```

```
  ' Calculates checksum without doing the final stage of calculations
  chksum := 0
  repeat while hdrlen > 1
    chksum += (BYTE[packet++] << 8) + BYTE[packet++]
    chksum := (chksum >> 16) + (chksum & $FFFF)
    hdrlen -= 2
  if hdrlen > 0
    chksum += BYTE[packet] << 8}

' ***************************
' ** Memory Access Helpers **
' ***************************
PRI conv_endianlong(in)
  return (in << 24) + ((in & $FF00) << 8) + ((in & $FF0000) >> 8) + (in >> 24)  ' we can sometimes get away with
  shifting without masking, since shifts kill extra bits anyways

PRI conv_endianword(in)
  return ((in & $FF) << 8) + ((in & $FF00) >> 8)

' ************************************
' ** Public Accessors (Thread Safe) **
' ************************************
PUB listen(port) | handle_addr, handle
'' Sets up a socket for listening on a port
''    port = port number to listen on
'' Returns handle if available, -1 if none available
'' Nonblocking

  ' just find any avail closed socket
  handle_addr := \find_socket(0, 0, 0)

  if handle_addr < 0
    return -1

  bytefill(handle_addr, 0, sSockIndex)                  ' clean socket state data, up to the socket index since this
  must stay
                                                        ' assumes socket index is last part of the array

  WORD[handle_addr + sSrcPort] := 0                     ' no source port yet
  WORD[handle_addr + sDstPort] := conv_endianword(port) ' we do have a dest port though

  WORD[handle_addr + sLastWin] := buffer_length

  ' it's now listening
  BYTE[handle_addr + sConState] := SLISTEN

  return BYTE[handle_addr + sSockIndex]

PUB connect(ipaddr, remoteport) | handle_addr
'' Connect to remote host
''    ipaddr     = ipv4 address packed into a long (ie: 1.2.3.4 => $01_02_03_04)
''    remoteport = port number to connect to
'' Returns handle to new socket, -1 if no socket available
'' Nonblocking

  ' just find any avail closed socket
  handle_addr := \find_socket(0, 0, 0)

  if handle_addr < 0
    return -1

  bytefill(handle_addr, 0, sSockIndex)                  ' clean socket state data, up to the socket index since this
  must stay
                                                        ' assumes socket index is last part of the array

  if(ip_ephport => EPHPORTEND)                          ' constrain ephport to specified range
    ip_ephport := EPHPORTSTART

  ' copy in ip, port data (with respect to the remote host, since we use same code as server)
  LONG[handle_addr + sSrcIp] := conv_endianlong(ipaddr)
  WORD[handle_addr + sSrcPort] := conv_endianword(remoteport)
  WORD[handle_addr + sDstPort] := conv_endianword(ip_ephport++)

  WORD[handle_addr + sLastWin] := buffer_length

  BYTE[handle_addr + sConState] := SCONNECTINGARP1

  return BYTE[handle_addr + sSockIndex]
```

```
PUB close(handle) | handle_addr, t
'' Closes a connection

  handle_addr := @sSockets + (sSocketBytes * handle)
  if isConnected(handle)
    BYTE[handle_addr + sConState] := SCLOSING
  else
    BYTE[handle_addr + sConState] := SCLOSED

  {' wait a bit for the connection to close
  ' if we get no response from remote host then we just RST
  t := cnt
  repeat until (BYTE[handle_addr + sConState] == SCLOSED) or (cnt - t) / (clkfreq / 1000) > TIMEOUTMS
  if BYTE[handle_addr + sConState] <> SCLOSED
    ' if we haven't closed by now then we will force close via RST
    BYTE[handle_addr + sConState] := SFORCECLOSE}

PUB isConnected(handle) | handle_addr
'' Returns true if the socket is connected, false otherwise

  handle_addr := @sSockets + (sSocketBytes * handle)
  return (BYTE[handle_addr + sConState] == SESTABLISHED)

PUB isValidHandle(handle) | handle_addr
'' Checks to see if the handle is valid, handles will become invalid once they are used
'' In other words, a closed listening socket is now invalid, etc

  if handle < 0 OR handle > constant(sNumSockets - 1)
    ' obviously the handle index is out of range, so it's not valid!
    return false

  handle_addr := @sSockets + (sSocketBytes * handle)
  return (BYTE[handle_addr + sConState] <> SCLOSED)

PUB readDataNonBlocking(handle, ptr, maxlen) | len, rxptr
'' Reads bytes from the socket
'' Returns number of read bytes
'' Not blocking (returns -1 if no data)

  if rx_tail[handle] == rx_head[handle]
    return -1

  len := (rx_head[handle] - rx_tail[handle]) & buffer_mask
  if maxlen < len
    len := maxlen

  rxptr := @rx_buffer + (handle * buffer_length)

  if (len + rx_tail[handle]) > buffer_length
    bytemove(ptr, rxptr + rx_tail[handle], buffer_length - rx_tail[handle])
    bytemove(ptr + (buffer_length - rx_tail[handle]), rxptr, len - (buffer_length - rx_tail[handle]))
  else
    bytemove(ptr, rxptr + rx_tail[handle], len)

  rx_tail[handle] := (rx_tail[handle] + len) & buffer_mask

  return len

PUB readData(handle, ptr, maxlen) : len
'' Reads bytes from the socket
'' Returns the number of read bytes
'' Will block until data is received

  repeat while (len := readDataNonBlocking(handle, ptr, maxlen)) < 0
    ifnot isConnected(handle)
      abort -1

PUB readByteNonBlocking(handle) : rxbyte | ptr
'' Read a byte from the specified socket
'' Will not block (returns -1 if no byte avail)

  rxbyte := -1
  if rx_tail[handle] <> rx_head[handle]
    ptr := @rx_buffer + (handle * buffer_length)
    rxbyte := BYTE[ptr][rx_tail[handle]]
    rx_tail[handle] := (rx_tail[handle] + 1) & buffer_mask
```

```
PUB readByte(handle) : rxbyte | ptr
'' Read a byte from the specified socket
'' Will block until a byte is received

  repeat while (rxbyte := readByteNonBlocking(handle)) < 0
    ifnot isConnected(handle)
      abort -1

PUB writeDataNonBlocking(handle, ptr, len) | txptr
'' Writes bytes to the socket
'' Will not write anything unless your data fits in the buffer
'' Non blocking (returns -1 if can't fit data)

  if (buffer_mask - ((tx_head[handle] - tx_tail[handle]) & buffer_mask)) < len
    return -1

  txptr := @tx_buffer + (handle * buffer_length)

  if (len + tx_head[handle]) > buffer_length
    bytemove(txptr + tx_head[handle], ptr, buffer_length - tx_head[handle])
    bytemove(txptr, ptr + (buffer_length - tx_head[handle]), len - (buffer_length - tx_head[handle]))
  else
    bytemove(txptr + tx_head[handle], ptr, len)

  tx_head[handle] := (tx_head[handle] + len) & buffer_mask

  return len

PUB writeData(handle, ptr, len)
'' Writes data to the specified socket
'' Will block until all data is queued to be sent

  repeat while len > constant(buffer_length - 1)
    repeat while writeDataNonBlocking(handle, ptr, constant(buffer_length - 1)) < 0
      ifnot isConnected(handle)
        abort -1
    len -= constant(buffer_length - 1)
    ptr += constant(buffer_length - 1)

  repeat while writeDataNonBlocking(handle, ptr, len) < 0
    ifnot isConnected(handle)
      abort -1

PUB writeByteNonBlocking(handle, txbyte) | ptr
'' Writes a byte to the specified socket
'' Will not block (returns -1 if no buffer space available)

  ifnot (tx_tail[handle] <> (tx_head[handle] + 1) & buffer_mask)
    return -1

  ptr := @tx_buffer + (handle * buffer_length)
  BYTE[ptr][tx_head[handle]] := txbyte
  tx_head[handle] := (tx_head[handle] + 1) & buffer_mask

  return txbyte

PUB writeByte(handle, txbyte)
'' Write a byte to the specified socket
'' Will block until space is available for byte to be sent

  repeat while writeByteNonBlocking(handle, txbyte) < 0
    ifnot isConnected(handle)
      abort -1

PUB resetBuffers(handle)
'' Resets send/receive buffers for the specified socket

  rx_tail[handle] := rx_head[handle]
  tx_head[handle] := tx_tail[handle]

PUB flush(handle)
'' Flushes the send buffer (waits till the buffer is empty)
'' Will block until all tx data is sent

  repeat while isConnected(handle) AND tx_tail[handle] <> tx_head[handle]

PUB getSocketState(handle) | handle_addr
'' Gets the socket state (internal state numbers)
```

```
'' You can include driver_socket in any object and use the S... state constants for comparison

  handle_addr := @sSockets + (sSocketBytes * handle)
  return BYTE[handle_addr + sConState]

CON
  '*****************************************************************
  '*      TCP Flags
  '*****************************************************************
  TCP_FIN = 1
  TCP_SYN = 2
  TCP_RST = 4
  TCP_PSH = 8
  TCP_ACK = 16
  TCP_URG = 32
  TCP_ECE = 64
  TCP_CWR = 128
  '*****************************************************************
  '*      Ethernet Header Layout
  '*****************************************************************
  enetpacketDest0 = $00  'destination mac address
  enetpacketDest1 = $01
  enetpacketDest2 = $02
  enetpacketDest3 = $03
  enetpacketDest4 = $04
  enetpacketDest5 = $05
  enetpacketSrc0 = $06  'source mac address
  enetpacketSrc1 = $07
  enetpacketSrc2 = $08
  enetpacketSrc3 = $09
  enetpacketSrc4 = $0A
  enetpacketSrc5 = $0B
  enetpacketType0 = $0C  'type/length field
  enetpacketType1 = $0D
  enetpacketData = $0E  'IP data area begins here
  '*****************************************************************
  '*      ARP Layout
  '*****************************************************************
  arp_hwtype = $0E
  arp_prtype = $10
  arp_hwlen = $12
  arp_prlen = $13
  arp_op = $14
  arp_shaddr = $16   'arp source mac address
  arp_sipaddr = $1C   'arp source ip address
  arp_thaddr = $20   'arp target mac address
  arp_tipaddr = $26   'arp target ip address
  '*****************************************************************
  '*      IP Header Layout
  '*****************************************************************
  ip_vers_len = $0E       'IP version and header length 1a19
  ip_tos = $0F    'IP type of service
  ip_pktlen = $10 'packet length
  ip_id = $12     'datagram id
  ip_frag_offset = $14    'fragment offset
  ip_ttl = $16    'time to live
  ip_proto = $17  'protocol (ICMP=1, TCP=6, UDP=11)
  ip_hdr_cksum = $18      'header checksum 1a23
  ip_srcaddr = $1A        'IP address of source
  ip_destaddr = $1E       'IP addess of destination
  ip_data = $22   'IP data area
  '*****************************************************************
  '*      TCP Header Layout
  '*****************************************************************
  TCP_srcport = $22       'TCP source port
  TCP_destport = $24      'TCP destination port
  TCP_seqnum = $26        'sequence number
  TCP_acknum = $2A        'acknowledgement number
  TCP_hdrlen = $2E        '4-bit header len (upper 4 bits)
  TCP_hdrflags = $2F      'TCP flags
  TCP_window = $30        'window size
  TCP_cksum = $32 'TCP checksum
  TCP_urgentptr = $34     'urgent pointer
  TCP_data = $36 'option/data
  '*****************************************************************
  '*      IP Protocol Types
  '*****************************************************************
  PROT_ICMP = $01
```

```
PROT_TCP = $06
PROT_UDP = $11
'*****************************************************************
'*      ICMP Header
'*****************************************************************
ICMP_type = ip_data
ICMP_code = ICMP_type+1
ICMP_cksum = ICMP_code+1
ICMP_id = ICMP_cksum+2
ICMP_seqnum = ICMP_id+2
ICMP_data = ICMP_seqnum+2
'*****************************************************************
'*      UDP Header
'*****************************************************************
UDP_srcport = ip_data
UDP_destport = UDP_srcport+2
UDP_len = UDP_destport+2
UDP_cksum = UDP_len+2
UDP_data = UDP_cksum+2
'*****************************************************************
'*      DHCP Message
'*****************************************************************
DHCP_op = UDP_data
DHCP_htype = DHCP_op+1
DHCP_hlen = DHCP_htype+1
DHCP_hops = DHCP_hlen+1
DHCP_xid = DHCP_hops+1
DHCP_secs = DHCP_xid+4
DHCP_flags = DHCP_secs+2
DHCP_ciaddr = DHCP_flags+2
DHCP_yiaddr = DHCP_ciaddr+4
DHCP_siaddr = DHCP_yiaddr+4
DHCP_giaddr = DHCP_siaddr+4
DHCP_chaddr = DHCP_giaddr+4
DHCP_sname = DHCP_chaddr+16
DHCP_file = DHCP_sname+64
DHCP_options = DHCP_file+128
DHCP_message_end = DHCP_options+312
```

driver_enc28j60.spin

```
{{
  ENC28J60 Ethernet NIC / MAC Driver
  ----------------------------------

  Copyright (C) 2006-2008 Harrison Pham

  This file is part of PropTCP.

  PropTCP is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 3 of the License, or
  (at your option) any later version.

  PropTCP is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.

  ----

  Driver Framework / API derived from EDTP Framethrower Fundamental Driver by Fred Eady
  Constant names / Theoretical Code Logic derived from Microchip Technology, Inc.'s enc28j60.c / enc28j60.h files
  SPI Assembly Driver derived from SPI Engine v1.1 by Beau Schwabe (Parallax)
}}

CON
  version = 3     ' major version
  release = 5     ' minor version

CON
' ****************************************
' **       ENC28J60 SRAM Defines       **
' ****************************************
  ' Silicon Revision
  silicon_rev = 5                             ' required silicon revision (current is B5)

  ' ENC28J60 Frequency
  enc_freq = 5_000_000

  ' ENC28J60 SRAM Usage Constants
  MAXFRAME = 1518                             ' 6 (src addr) + 6 (dst addr) + 2 (type) + 1500 (data) + 4 (FCS CRC)
  = 1518 bytes
  TX_BUFFER_SIZE = 1518

  TXSTART = 8192 - (TX_BUFFER_SIZE + 8)
  TXEND = TXSTART + (TX_BUFFER_SIZE + 8)
  RXSTART = $0000
  RXSTOP = (TXSTART - 2) | $0001              ' must be odd (B5 Errata)
  RXSIZE = (RXSTOP - RXSTART + 1)

DAT
' ****************************************
' **    MAC Address Vars / Defaults    **
' ****************************************
  ' ** This is the default MAC address used by this driver.  The parent object
  '    can override this by passing a pointer to a new MAC address in the public
  '    start() method.  It is recommend that this is done to provide a level of
  '    abstraction and makes tcp stack design easier.
  ' ** This is the ethernet MAC address, it is critical that you change this
  '    if you have more than one device using this code on a local network.
  ' ** If you plan on commercial deployment, you must purchase MAC address
  '    groups from IEEE or some other standards organization.
  eth_mac         byte    $10, $00, $00, $00, $00, $01

' ****************************************
' **         Global Variables          **
' ****************************************
  rxlen           word 0
  tx_end          word 0

  intpin             byte 0
```

```
  packetheader    byte 0[6]

  packet          byte 0[MAXFRAME]

PUB start(_cs, _sck, _si, _so, _int, xtalout, macptr)
'' Starts the driver (uses 1 cog for spi engine)

  intpin := _int
  dira[intpin] := %0

  ' Since some people don't have 25mhz crystals, we use the cog counters
  ' to generate a 25mhz frequency for the ENC28J60 (I love the Propeller)
  ' Note: This requires a main crystal that is a multiple of 25mhz (5mhz works).
  spi_start(_cs, _sck, _so, _si, xtalout)

  ' If a MAC address pointer is provided (addr > -1) then copy it into
  ' the MAC address array (this kind of wastes space, but simplifies usage).
  if macptr > -1
    bytemove(@eth_mac, macptr, 6)

  delay_ms(50)
  init_ENC28J60

  ' check to make sure its a valid supported silicon rev
  banksel(EREVID)
  return rd_cntlreg(EREVID) == silicon_rev      ' return true if silicon rev match

PUB stop
'' Stops the driver, frees 1 cog

  spi_stop

PUB rd_macreg(address) : data
'' Read MAC Control Register

  spi_out_cs(cRCR | address)
  spi_out_cs(0)                 ' transmit dummy byte
  data := spi_in                ' get actual data

PUB rd_cntlreg(address) : data
'' Read ETH Control Register

  spi_out_cs(cRCR | address)
  data := spi_in

PUB wr_reg(address, data)
'' Write MAC and ETH Control Register

  spi_out_cs(cWCR | address)
  spi_out(data)

PUB bfc_reg(address, data)
'' Clear Control Register Bits

  spi_out_cs(cBFC | address)
  spi_out(data)

PUB bfs_reg(address, data)
'' Set Control Register Bits

  spi_out_cs(cBFS | address)
  spi_out(data)

PUB soft_reset
'' Soft Reset ENC28J60

  spi_out(cSC)

PUB banksel(register)
'' Select Control Register Bank

  bfc_reg(ECON1, %0000_0011)
  bfs_reg(ECON1, register >> 8)                         ' high byte

PUB rd_phy(register) | low, high
'' Read ENC28J60 PHY Register
```

```
  banksel(MIREGADR)
  wr_reg(MIREGADR, register)
  wr_reg(MICMD, MICMD_MIIRD)
  banksel(MISTAT)
  repeat while ((rd_macreg(MISTAT) & MISTAT_BUSY) > 0)
  banksel(MIREGADR)
  wr_reg(MICMD, $00)
  low := rd_macreg(MIRDL)
  high := rd_macreg(MIRDH)
  return (high << 8) + low

PUB wr_phy(register, data)
'' Write ENC28J60 PHY Register

  banksel(MIREGADR)
  wr_reg(MIREGADR, register)
  wr_reg(MIWRL, data)
  wr_reg(MIWRH, data >> 8)
  banksel(MISTAT)
  repeat while ((rd_macreg(MISTAT) & MISTAT_BUSY) > 0)

PUB rd_sram : data
'' Read ENC28J60 8k Buffer Memory

  spi_out_cs(cRBM)
  data := spi_in

PUB wr_sram(data)
'' Write ENC28J60 8k Buffer Memory

  spi_out_cs(cWBM)
  spi_out(data)

PUB init_ENC28J60 | i
'' Init ENC28J60 Chip

  repeat
    i := rd_cntlreg(ESTAT)
  while (i & $08) OR (!i & ESTAT_CLKRDY)

  soft_reset
  delay_ms(5)                                           ' reset delay

  bfc_reg(ECON1, ECON1_RXEN)                            ' stop send / recv
  bfc_reg(ECON1, ECON1_TXRTS)

  bfs_reg(ECON2, ECON2_AUTOINC)                         ' enable auto increment of sram pointers (already default)

  packetheader[nextpacket_low] := RXSTART
  packetheader[nextpacket_high] := constant(RXSTART >> 8)

  banksel(ERDPTL)
  wr_reg(ERDPTL, RXSTART)
  wr_reg(ERDPTH, constant(RXSTART >> 8))

  banksel(ERXSTL)
  wr_reg(ERXSTL, RXSTART)
  wr_reg(ERXSTH, constant(RXSTART >> 8))
  wr_reg(ERXRDPTL, RXSTOP)
  wr_reg(ERXRDPTH, constant(RXSTOP >> 8))
  wr_reg(ERXNDL, RXSTOP)
  wr_reg(ERXNDH, constant(RXSTOP >> 8))
  wr_reg(ETXSTL, TXSTART)
  wr_reg(ETXSTH, constant(TXSTART >> 8))

  banksel(MACON1)
  wr_reg(MACON1, constant(MACON1_TXPAUS | MACON1_RXPAUS | MACON1_MARXEN))
  wr_reg(MACON3, constant(MACON3_TXCRCEN | MACON3_PADCFG0 | MACON3_FRMLNEN))

  ' don't timeout transmissions on saturated media
  wr_reg(MACON4, MACON4_DEFER)
  ' collisions occur at 63rd byte
  wr_reg(MACLCON2, 63)

  wr_reg(MAIPGL, $12)
  wr_reg(MAIPGH, $0C)
  wr_reg(MAMXFLL, MAXFRAME)
  wr_reg(MAMXFLH, constant(MAXFRAME >> 8))
```

```
  ' back-to-back inter-packet gap time
  ' full duplex = 0x15 (9.6us)
  ' half duplex = 0x12 (9.6us)
  wr_reg(MABBIPG, $12)
  wr_reg(MAIPGL, $12)
  wr_reg(MAIPGH, $0C)

  ' write mac address to the chip
  banksel(MAADR1)
  wr_reg(MAADR1, eth_mac[0])
  wr_reg(MAADR2, eth_mac[1])
  wr_reg(MAADR3, eth_mac[2])
  wr_reg(MAADR4, eth_mac[3])
  wr_reg(MAADR5, eth_mac[4])
  wr_reg(MAADR6, eth_mac[5])

  ' half duplex
  wr_phy(PHCON2, PHCON2_HDLDIS)
  wr_phy(PHCON1, $0000)

  ' set LED options (led A = link, led B = tx/rx)
  wr_phy(PHLCON, $0472)         '$0472

  ' enable packet reception
  bfs_reg(ECON1, ECON1_RXEN)

PUB get_frame | packet_addr, new_rdptr
'' Get Ethernet Frame from Buffer

  banksel(ERDPTL)
  wr_reg(ERDPTL, packetheader[nextpacket_low])
  wr_reg(ERDPTH, packetheader[nextpacket_high])

  repeat packet_addr from 0 to 5
    packetheader[packet_addr] := rd_sram

  rxlen := (packetheader[rec_bytecnt_high] << 8) + packetheader[rec_bytecnt_low]

  'bytefill(@packet, 0, MAXFRAME)                       ' Uncomment this if you want to clean out the buffer first
                                                        '   otherwise, leave commented since it's faster to just leave
  stuff
                                                        '   in the buffer

  ' protect from oversized packet
  if rxlen =< MAXFRAME
    rd_block(@packet, rxlen)
    {repeat packet_addr from 0 to rxlen - 1
      BYTE[@packet][packet_addr] := rd_sram}

  new_rdptr := (packetheader[nextpacket_high] << 8) + packetheader[nextpacket_low]

  ' handle errata read pointer start (must be odd)
  --new_rdptr

  if (new_rdptr < RXSTART) OR (new_rdptr > RXSTOP)
    new_rdptr := RXSTOP

  bfs_reg(ECON2, ECON2_PKTDEC)

  banksel(ERXRDPTL)
  wr_reg(ERXRDPTL, new_rdptr)
  wr_reg(ERXRDPTH, new_rdptr >> 8)

PUB start_frame
'' Start frame - Inits the NIC and sets stuff

  banksel(EWRPTL)
  wr_reg(EWRPTL, TXSTART)
  wr_reg(EWRPTH, constant(TXSTART >> 8))

  tx_end := constant(TXSTART - 1)         ' start location is really address 0, so we are sending a count of - 1

  wr_frame(cTXCONTROL)

PUB wr_frame(data)
'' Write frame data
```

```
  wr_sram(data)
  ++tx_end

PUB wr_block(startaddr, count)
  blockwrite(startaddr, count)
  tx_end += count

PUB rd_block(startaddr, count)
  blockread(startaddr, count)

PUB send_frame
'' Sends frame
'' Will retry on send failure up to 15 times with a 1ms delay in between repeats

  repeat 15
    if p_send_frame             ' send packet, if successful then quit retry loop
      quit
    delay_ms(1)

PRI p_send_frame | i, eirval
' Sends the frame
  banksel(ETXSTL)
  wr_reg(ETXSTL, TXSTART)
  wr_reg(ETXSTH, constant(TXSTART >> 8))

  banksel(ETXNDL)
  wr_reg(ETXNDL, tx_end)
  wr_reg(ETXNDH, tx_end >> 8)

  ' B5 Errata #10 - Reset transmit logic before send
  bfs_reg(ECON1, ECON1_TXRST)
  bfc_reg(ECON1, ECON1_TXRST)

  ' B5 Errata #10 & #13: Reset interrupt error flags
  bfc_reg(EIR, constant(EIR_TXERIF | EIR_TXIF))

  ' trigger send
  bfs_reg(ECON1, ECON1_TXRTS)

  ' fix for transmit stalls (derived from errata B5 #13), watches TXIF and TXERIF bits
  ' also implements a ~3.75ms (15 * 250us) timeout if send fails (occurs on random packet collisions)
  ' btw: this took over 10 hours to fix due to the elusive undocumented bug
  i := 0
  repeat
    eirval := rd_cntlreg(EIR)
    if ((eirval & constant(EIR_TXERIF | EIR_TXIF)) > 0)
      quit
    if (++i => 15)
      eirval := EIR_TXERIF
      quit
    delay_us(250)

  ' B5 Errata #13 - Reset TXRTS if failed send then reset logic
  bfc_reg(ECON1, ECON1_TXRTS)

  if ((eirval & EIR_TXERIF) == 0)
    return true   ' successful send (no error interrupt)
  else
    return false  ' failed send (error interrupt)

PUB get_packetpointer
'' Gets packet pointer (for external object access)
  return @packet

PUB get_mac_pointer
'' Gets mac address pointer
  return @eth_mac

PUB get_rxlen
'' Gets received packet length
  return rxlen - 4             ' knock off the 4 byte Frame Check Sequence CRC, not used anywhere outside of this
  driver (pg 31 datasheet)

PRI delay_us(Duration)
  waitcnt(((clkfreq / 1_000_000 * Duration - 3928)) + cnt)

PRI delay_ms(Duration)
  waitcnt(((clkfreq / 1_000 * Duration - 3932)) + cnt)
```

```
' ****************************************
' **          ASM SPI Engine            **
' ****************************************
DAT
  cog         long 0
  command     long 0

CON
  SPIOUT        = %0000_0001
  SPIIN         = %0000_0010
  SRAMWRITE     = %0000_0100
  SRAMREAD      = %0000_1000
  CSON          = %0001_0000
  CSOFF         = %0010_0000
  CKSUM         = %0100_0000

  SPIBITS       = 8

PRI spi_out(value)
  setcommand(constant(SPIOUT | CSON | CSOFF), @value)

PRI spi_out_cs(value)
  setcommand(constant(SPIOUT | CSON), @value)

PRI spi_in : value
  setcommand(constant(SPIIN | CSON | CSOFF), @value)

PRI spi_in_cs : value
  setcommand(constant(SPIIN | CSON), @value)

PRI blockwrite(startaddr, count)
  setcommand(SRAMWRITE, @startaddr)

PRI blockread(startaddr, count)
  setcommand(SRAMREAD, @startaddr)

PUB chksum_add(startaddr, count)
  setcommand(CKSUM, @startaddr)
  return startaddr

PRI spi_start(_cs, _sck, _di, _do, _freqpin)
  spi_stop

  cspin := |< _cs
  dipin := |< _di
  dopin := |< _do
  clkpin := |< _sck

  spi_setupfreqsynth(_freqpin)

  cog := cognew(@init, @command) + 1

PRI spi_stop
  if cog
    cogstop(cog~ - 1)
  command~

PRI setcommand(cmd, argptr)
  command := cmd << 16 + argptr                        'write command and pointer
  repeat while command                                 'wait for command to be cleared, signifying receipt

PRI spi_setupfreqsynth(pin)

  if pin < 0
    ' pin num was negative -> disable freq synth
    freqpin := 0
    return

  freqpin := |< pin

  asmctr := constant(%00010 << 26)                                                    '..set PLL
  mode
  asmctr |= constant((>|((enc_freq - 1) / 1_000_000)) << 23)                          'set PLLDIV

  asmfrq := spi_fraction(enc_freq, CLKFREQ, constant(4 - (>|((enc_freq - 1) / 1_000_000)))   'Compute
  FRQA/FRQB value
```

```
  asmctr |= pin                                                                                    'set PINA to
  complete CTRA/CTRB value

PRI spi_fraction(a, b, shift) : f

  if shift > 0                          'if shift, pre-shift a or b left
    a <<= shift                         'to maintain significant bits while
  if shift < 0                          'insuring proper result
    b <<= -shift

  repeat 32                             'perform long division of a/b
    f <<= 1
    if a => b
      a -= b
      f++
    a <<= 1

DAT
              org
init
              or        dira, cspin             'pin directions
              andn      dira, dipin
              or        dira, dopin
              or        dira, clkpin

              or        outa, cspin             'turn off cs (bring it high)

              tjz       freqpin, #loop          'jump to main loop if synth freq disabled (freqpin = 0)

ctr_setup     or        dira, freqpin           'freq synth pin
              mov       frqa, asmfrq            'frq
              mov       ctra, asmctr            'ctr

loop          wrlong    zero,par                'zero command (tell spin we are done processing)
:subloop      rdlong    t1,par wz               'wait for command
        if_z  jmp       #:subloop

              mov       addr, t1                'used for holding return addr to spin vars

              rdlong    arg0, t1                'arg0
              add       t1, #4
              rdlong    arg1, t1                'arg1

              mov       lkup, addr              'get the command var from spin
              shr       lkup, #16               'extract the cmd from the command var

              test      lkup, #CSON wz          'turn on cs
        if_nz andn      outa, cspin

              test      lkup, #SPIOUT wz        'spi out
        if_nz call      #spi_out_
              test      lkup, #SPIIN wz         'spi in
        if_nz call      #xspi_in_
              test      lkup, #SRAMWRITE wz     'sram block write
        if_nz jmp       #sram_write_
              test      lkup, #SRAMREAD wz      'sram block read
        if_nz jmp       #sram_read_

              test      lkup, #CSOFF wz         'cs off
        if_nz or        outa, cspin

              test      lkup, #CKSUM wz         'perform checksum
        if_nz call      #csum16

              jmp #loop                         ' no cmd found



spi_out_                                        'SHIFTOUT Entry
              mov       t4, #SPIBITS            '     Load number of data bits
              andn      outa, dopin             '          PreSet DataPin LOW
              andn      outa, clkpin            '          PreSet ClockPin LOW

              mov       t3, arg0                '          Load t3 with DataValue
              mov       t5, #%1                 '          Create MSB mask     ;     load t5 with "1"
              shl       t5, #SPIBITS            '          Shift "1" N number of bits to the left.
              shr       t5, #1                  '          Shifting the number of bits left actually puts
                                                '          us one more place to the left than we want. To
```

```
                                                '           compensate we'll shift one position right.
:sout_loop    test      t3, t5      wc          '           Test MSB of DataValue
              muxc      outa, dopin             '           Set DataBit HIGH or LOW
              shr       t5, #1                  '           Prepare for next DataBit
              call      #clock                  '           Send clock pulse
              djnz      t4, #:sout_loop         '           Decrement t4 ; jump if not Zero
              andn      outa, dopin

spi_out__ret  ret                               '      Go wait for next command



spi_in_                                         'SHIFTIN Entry
              mov       t4, #SPIBITS            '      Load number of data bits
              andn      outa, clkpin            '           PreSet ClockPin LOW

:sin_loop     test      dipin, ina wc           '           Read Data Bit into 'C' flag
              rcl       t3, #1                  '           rotate "C" flag into return value
              call      #clock                  '           Send clock pulse
              djnz      t4, #:sin_loop          '           Decrement t4 ; jump if not Zero

              mov       arg0, t3
spi_in__ret   ret                               '      Go wait for next command

clock
              mov       clkpin, #0 wz,nr        '      Clock Pin
              muxz      outa, clkpin            '           Set ClockPin HIGH
              muxnz     outa, clkpin            '           Set ClockPin LOW
clock_ret     ret                               '           return

xspi_in_
              call      #spi_in_
              wrbyte    arg0, addr                 ' write byte back to spin result var
xspi_in__ret  ret

' SRAM Block Read/Write
sram_write_   ' block write (arg0=hub addr, arg1=count)
              mov       t1, arg0
              mov       t2, arg1

              andn      outa, cspin
              mov       arg0, #cWBM
              call      #spi_out_
:loop         rdbyte    arg0, t1
              call      #spi_out_
              add       t1, #1
              djnz      t2, #:loop
              or        outa, cspin

              jmp       #loop

sram_read_    ' block read (arg0=hub addr, arg1=count)
              mov       t1, arg0
              mov       t2, arg1

              andn      outa, cspin
              mov       arg0, #cRBM
              call      #spi_out_
:loop         call      #spi_in_
              wrbyte    arg0, t1
              add       t1, #1
              djnz      t2, #:loop
              or        outa, cspin

              jmp       #loop

csum16        ' performs checksum 16bit additions on the data
              ' arg0=hub addr, arg1=length, writes sum to first arg
              mov       t1, #0                  ' clear sum
:loop         rdbyte    t2, arg0                ' read two bytes (16 bits)
              add       arg0, #1
              rdbyte    t3, arg0
              add       arg0, #1
              shl       t2, #8                  ' build the word
              add       t2, t3
              add       t1, t2                  ' add numbers
              mov       t2, t1                  ' add lower and upper words together
              shr       t2, #16
```

```
             and       t1, hffff
             add       t1, t2
             sub       arg1, #2
             cmp       arg1, #1 wz, wc
 if_nc_and_nz jmp      #:loop
        if_z  rdbyte   t2, arg0                ' add last byte (odd)
        if_z  shl      t2, #8
        if_z  add      t1, t2
             wrlong    t1, addr                ' return result back too SPIN
csum16_ret   ret

zero                   long    0                       'constants

                                                       'values filled by spin code before launching
cspin                  long    0                       ' chip select pin
dipin                  long    0                       ' data in pin (enc28j60 -> prop)
dopin                  long    0                       ' data out pin (prop -> enc28j60)
clkpin                 long    0                       ' clock pin (prop -> enc28j60)
freqpin                long    0                       ' frequency synth pin (prop -> enc28j60)

asmctr                 long    0                       ' precalculated synth freq values
asmfrq                 long    0                       '

hffff                  long    $FFFF

                                                       'temp variables
t1                     res     1                       '     loop and cog shutdown
t2                     res     1                       '     loop and cog shutdown
t3                     res     1                       '     Used to hold DataValue SHIFTIN/SHIFTOUT
t4                     res     1                       '     Used to hold # of Bits
t5                     res     1                       '     Used for temporary data mask

addr                   res     1                       '     Used to hold return address of first Argument passed
lkup                   res     1                       '     Used to hold command lookup

                                                       'arguments passed to/from high-level Spin
arg0                   res     1                       ' bits / start address
arg1                   res     1                       ' value / count

CON
' ****************************************
' **    ENC28J60 Control Constants     **
' ****************************************
  ' ENC28J60 opcodes (OR with 5bit address)
  cWCR = %010 << 5              ' write control register command
  cBFS = %100 << 5              ' bit field set command
  cBFC = %101 << 5              ' bit field clear command
  cRCR = %000 << 5              ' read control register command
  cRBM = (%001 << 5) | $1A      ' read buffer memory command
  cWBM = (%011 << 5) | $1A      ' write buffer memory command
  cSC = (%111 << 5) | $1F       ' system command

  ' This is used to trigger TX in the ENC28J60, it shouldn't change, but you never know...
  cTXCONTROL = $0E

  ' Packet header format (tail of the receive packet in the ENC28J60 SRAM)
  #0,nextpacket_low,nextpacket_high,rec_bytecnt_low,rec_bytecnt_high,rec_status_low,rec_status_high

' ****************************************
' **     ENC28J60 Register Defines     **
' ****************************************
  ' Bank 0 registers --------
  ERDPTL = $00
  ERDPTH = $01
  EWRPTL = $02
  EWRPTH = $03
  ETXSTL = $04
  ETXSTH = $05
  ETXNDL = $06
  ETXNDH = $07
  ERXSTL = $08
  ERXSTH = $09
  ERXNDL = $0A
  ERXNDH = $0B
  ERXRDPTL = $0C
  ERXRDPTH = $0D
  ERXWRPTL = $0E
  ERXWRPTH = $0F
```

```
EDMASTL = $10
EDMASTH = $11
EDMANDL = $12
EDMANDH = $13
EDMADSTL = $14
EDMADSTH = $15
EDMACSL = $16
EDMACSH = $17
' = $18
' = $19
' r = $1A
EIE = $1B
EIR = $1C
ESTAT = $1D
ECON2 = $1E
ECON1 = $1F

' Bank 1 registers -----
EHT0 = $100
EHT1 = $101
EHT2 = $102
EHT3 = $103
EHT4 = $104
EHT5 = $105
EHT6 = $106
EHT7 = $107
EPMM0 = $108
EPMM1 = $109
EPMM2 = $10A
EPMM3 = $10B
EPMM4 = $10C
EPMM5 = $10D
EPMM6 = $10E
EPMM7 = $10F
EPMCSL = $110
EPMCSH = $111
' = $112
' = $113
EPMOL = $114
EPMOH = $115
EWOLIE = $116
EWOLIR = $117
ERXFCON = $118
EPKTCNT = $119
' r = $11A
' EIE = $11B
' EIR = $11C
' ESTAT = $11D
' ECON2 = $11E
' ECON1 = $11F

' Bank 2 registers -----
MACON1 = $200
MACON2 = $201
MACON3 = $202
MACON4 = $203
MABBIPG = $204
' = $205
MAIPGL = $206
MAIPGH = $207
MACLCON1 = $208
MACLCON2 = $209
MAMXFLL = $20A
MAMXFLH = $20B
' r = $20C
MAPHSUP = $20D
' r = $20E
' = $20F
' r = $210
MICON = $211
MICMD = $212
' = $213
MIREGADR = $214
' r = $215
MIWRL = $216
MIWRH = $217
MIRDL = $218
MIRDH = $219
```

```
' r = $21A
' EIE = $21B
' EIR = $21C
' ESTAT = $21D
' ECON2 = $21E
' ECON1 = $21F

' Bank 3 registers -----

MAADR5 = $300
MAADR6 = $301
MAADR3 = $302
MAADR4 = $303
MAADR1 = $304
MAADR2 = $305

{MAADR1 = $300
MAADR0 = $301
MAADR3 = $302
MAADR2 = $303
MAADR5 = $304
MAADR4 = $305}

EBSTSD = $306
EBSTCON = $307
EBSTCSL = $308
EBSTCSH = $309
MISTAT = $30A
' = $30B
' = $30C
' = $30D
' = $30E
' = $30F
' = $310
' = $311
EREVID = $312
' = $313
' = $314
ECOCON = $315
' EPHTST      $316
EFLOCON = $317
EPAUSL = $318
EPAUSH = $319
' r = $31A
' EIE = $31B
' EIR = $31C
' ESTAT = $31D
' ECON2 = $31E
' ECON1 = $31F

{*******************************************************************************
* PH Register Locations
*******************************************************************************}
PHCON1 = $00
PHSTAT1 = $01
PHID1 = $02
PHID2 = $03
PHCON2 = $10
PHSTAT2 = $11
PHIE = $12
PHIR = $13
PHLCON = $14

{*******************************************************************************
* Individual Register Bits
*******************************************************************************}
' ETH/MAC/MII bits

' EIE bits ----------
EIE_INTIE = (1<<7)
EIE_PKTIE = (1<<6)
EIE_DMAIE = (1<<5)
EIE_LINKIE = (1<<4)
EIE_TXIE = (1<<3)
EIE_WOLIE = (1<<2)
EIE_TXERIE = (1<<1)
EIE_RXERIE = (1)
```

```
' EIR bits ----------
EIR_PKTIF = (1<<6)
EIR_DMAIF = (1<<5)
EIR_LINKIF = (1<<4)
EIR_TXIF = (1<<3)
EIR_WOLIF = (1<<2)
EIR_TXERIF = (1<<1)
EIR_RXERIF = (1)

' ESTAT bits ---------
ESTAT_INT = (1<<7)
ESTAT_LATECOL = (1<<4)
ESTAT_RXBUSY = (1<<2)
ESTAT_TXABRT = (1<<1)
ESTAT_CLKRDY = (1)

' ECON2 bits --------
ECON2_AUTOINC = (1<<7)
ECON2_PKTDEC = (1<<6)
ECON2_PWRSV = (1<<5)
ECON2_VRTP = (1<<4)
ECON2_VRPS = (1<<3)

' ECON1 bits --------
ECON1_TXRST = (1<<7)
ECON1_RXRST = (1<<6)
ECON1_DMAST = (1<<5)
ECON1_CSUMEN = (1<<4)
ECON1_TXRTS = (1<<3)
ECON1_RXEN = (1<<2)
ECON1_BSEL1 = (1<<1)
ECON1_BSEL0 = (1)

' EWOLIE bits -------
EWOLIE_UCWOLIE = (1<<7)
EWOLIE_AWOLIE = (1<<6)
EWOLIE_PMWOLIE = (1<<4)
EWOLIE_MPWOLIE = (1<<3)
EWOLIE_HTWOLIE = (1<<2)
EWOLIE_MCWOLIE = (1<<1)
EWOLIE_BCWOLIE = (1)

' EWOLIR bits -------
EWOLIR_UCWOLIF = (1<<7)
EWOLIR_AWOLIF = (1<<6)
EWOLIR_PMWOLIF = (1<<4)
EWOLIR_MPWOLIF = (1<<3)
EWOLIR_HTWOLIF = (1<<2)
EWOLIR_MCWOLIF = (1<<1)
EWOLIR_BCWOLIF = (1)

' ERXFCON bits ------
ERXFCON_UCEN = (1<<7)
ERXFCON_ANDOR = (1<<6)
ERXFCON_CRCEN = (1<<5)
ERXFCON_PMEN = (1<<4)
ERXFCON_MPEN = (1<<3)
ERXFCON_HTEN = (1<<2)
ERXFCON_MCEN = (1<<1)
ERXFCON_BCEN = (1)

' MACON1 bits --------
MACON1_LOOPBK = (1<<4)
MACON1_TXPAUS = (1<<3)
MACON1_RXPAUS = (1<<2)
MACON1_PASSALL = (1<<1)
MACON1_MARXEN = (1)

' MACON2 bits --------
MACON2_MARST = (1<<7)
MACON2_RNDRST = (1<<6)
MACON2_MARXRST = (1<<3)
MACON2_RFUNRST = (1<<2)
MACON2_MATXRST = (1<<1)
MACON2_TFUNRST = (1)

' MACON3 bits --------
MACON3_PADCFG2 = (1<<7)
```

```
MACON3_PADCFG1 = (1<<6)
MACON3_PADCFG0 = (1<<5)
MACON3_TXCRCEN = (1<<4)
MACON3_PHDRLEN = (1<<3)
MACON3_HFRMEN = (1<<2)
MACON3_FRMLNEN = (1<<1)
MACON3_FULDPX = (1)

' MACON4 bits --------
MACON4_DEFER = (1<<6)
MACON4_BPEN = (1<<5)
MACON4_NOBKOFF = (1<<4)
MACON4_LONGPRE = (1<<1)
MACON4_PUREPRE = (1)

' MAPHSUP bits ----
MAPHSUP_RSTRMII = (1<<3)

' MICON bits --------
MICON_RSTMII = (1<<7)

' MICMD bits ---------
MICMD_MIISCAN = (1<<1)
MICMD_MIIRD = (1)

' EBSTCON bits -----
EBSTCON_PSV2 = (1<<7)
EBSTCON_PSV1 = (1<<6)
EBSTCON_PSV0 = (1<<5)
EBSTCON_PSEL = (1<<4)
EBSTCON_TMSEL1 = (1<<3)
EBSTCON_TMSEL0 = (1<<2)
EBSTCON_TME = (1<<1)
EBSTCON_BISTST = (1)

' MISTAT bits --------
MISTAT_NVALID = (1<<2)
MISTAT_SCAN = (1<<1)
MISTAT_BUSY = (1)

' ECOCON bits -------
ECOCON_COCON2 = (1<<2)
ECOCON_COCON1 = (1<<1)
ECOCON_COCON0 = (1)

' EFLOCON bits -----
EFLOCON_FULDPXS = (1<<2)
EFLOCON_FCEN1 = (1<<1)
EFLOCON_FCEN0 = (1)



' PHY bits

' PHCON1 bits ----------
PHCON1_PRST = (1<<15)
PHCON1_PLOOPBK = (1<<14)
PHCON1_PPWRSV = (1<<11)
PHCON1_PDPXMD = (1<<8)

' PHSTAT1 bits --------
PHSTAT1_PFDPX = (1<<12)
PHSTAT1_PHDPX = (1<<11)
PHSTAT1_LLSTAT = (1<<2)
PHSTAT1_JBSTAT = (1<<1)

' PHID2 bits --------
PHID2_PID24 = (1<<15)
PHID2_PID23 = (1<<14)
PHID2_PID22 = (1<<13)
PHID2_PID21 = (1<<12)
PHID2_PID20 = (1<<11)
PHID2_PID19 = (1<<10)
PHID2_PPN5 = (1<<9)
PHID2_PPN4 = (1<<8)
PHID2_PPN3 = (1<<7)
PHID2_PPN2 = (1<<6)
PHID2_PPN1 = (1<<5)
```

```
PHID2_PPN0 = (1<<4)
PHID2_PREV3 = (1<<3)
PHID2_PREV2 = (1<<2)
PHID2_PREV1 = (1<<1)
PHID2_PREV0 = (1)

' PHCON2 bits ----------
PHCON2_FRCLNK = (1<<14)
PHCON2_TXDIS = (1<<13)
PHCON2_JABBER = (1<<10)
PHCON2_HDLDIS = (1<<8)

' PHSTAT2 bits --------
PHSTAT2_TXSTAT = (1<<13)
PHSTAT2_RXSTAT = (1<<12)
PHSTAT2_COLSTAT = (1<<11)
PHSTAT2_LSTAT = (1<<10)
PHSTAT2_DPXSTAT = (1<<9)
PHSTAT2_PLRITY = (1<<5)

' PHIE bits -----------
PHIE_PLNKIE = (1<<4)
PHIE_PGEIE = (1<<1)

' PHIR bits -----------
PHIR_PLNKIF = (1<<4)
PHIR_PGIF = (1<<2)

' PHLCON bits -------
PHLCON_LACFG3 = (1<<11)
PHLCON_LACFG2 = (1<<10)
PHLCON_LACFG1 = (1<<9)
PHLCON_LACFG0 = (1<<8)
PHLCON_LBCFG3 = (1<<7)
PHLCON_LBCFG2 = (1<<6)
PHLCON_LBCFG1 = (1<<5)
PHLCON_LBCFG0 = (1<<4)
PHLCON_LFRQ1 = (1<<3)
PHLCON_LFRQ0 = (1<<2)
PHLCON_STRCH = (1<<1)
```

util_strings.spin

```
'' String Utilities
'' ----------------
'' Copyright (C) 2006-2008 Harrison Pham

'' JAVA-like strings methods

CON

VAR

PUB indexOf(haystack, needle) | i, j
  '' Searches for a 'needle' inside a 'haystack'
  '' Returns starting index of 'needle' inside 'haystack'

  repeat i from 0 to strsize(haystack) - strsize(needle)
    repeat j from 0 to strsize(needle) - 1
      if byte[haystack][i + j] <> byte[needle][j]
        quit
    if j == strsize(needle)
      return i

  return -1

{PUB indexOfChar(haystack, char) | i
  repeat i from 0 to strsize(haystack) - 1
    if byte[haystack][i] == char
      return i

  return -1}

PUB subString(src, start, end, dst) | len
  '' Extracts a portion of a string
  '' The dst string must be large enough to fit the resultant string

  if end == -1
    len := strsize(src) - start
  else
    len := end - start

  bytemove(dst, src + start, len)
  byte[dst][len] := 0

PUB toLower(str) | i, len
  '' Converts string to lower case
  '' This WILL mutate your string

  if (len := strsize(str)) == 0
    return

  repeat i from 0 to len - 1
    'if byte[str][i] => "A" and byte[str][i] =< "Z"
      byte[str][i] := byte[str][i] | constant(1 << 5)
```

softrtc.spin

```
{{
  Software RTC w/ NIST Daytime Sync Support

  (c) 2008 Harrison Pham.
}}


OBJ
  clsock : "api_telnet_serial"
  dt : "date_time_epoch"

VAR
  long timeset        ' stores the value to add to the counter time to get a unix timestamp

PUB start(pin)
  '' Starts the RTC
  '' Does not set the time or perform any updates

  timeset := 0
  initCounters(pin)

PUB update
  '' Updates the RTC using the NIST daytime services
  '' Returns negative numbers on error

  ' try to update the time, retry 5 times on failure
  repeat 5
    if \_update => 0
      return 1
    delay_ms(500)

  return -1

PRI _update | y, mo, d, h, m, s

  clsock.connect(constant((132 << 24) + (163 << 16) + (4 << 8) + 102), 13)
  clsock.resetBuffers
  clsock.waitConnectTimeout(2000)
  if clsock.isConnected
    ' connected
    'JJJJJ YR-MO-DA HH:MM:SS TT L H msADV UTC(NIST) OTM

    'timeset := 10000

    repeat 7
      clsock.rx

    y := 2000 + ((clsock.rx - "0") * 10) + (clsock.rx - "0")
    clsock.rx
    mo := ((clsock.rx - "0") * 10) + (clsock.rx - "0")
    clsock.rx
    d := ((clsock.rx - "0") * 10) + (clsock.rx - "0")
    clsock.rx
    h := ((clsock.rx - "0") * 10) + (clsock.rx - "0")
    clsock.rx
    m := ((clsock.rx - "0") * 10) + (clsock.rx - "0")
    clsock.rx
    s := ((clsock.rx - "0") * 10) + (clsock.rx - "0")

    timeset := dt.toETV(y,mo,d,h,m,s) - getCounter

    clsock.close
    return 0
  else
    clsock.close
    return -1

PUB setTimestamp(newstamp)
  '' Sets a new unix timestamp

  timeset := newstamp - getCounter

PUB getTimestamp
```

```
  return getCounter + timeset

PRI getCounter
  '' Gets the current unix timestamp

  if clkfreq == 80_000_000
    return phsb ** $35AFE535                            ' timer value is stored in phsb (seconds)
  else
    return phsb

PRI initCounters(commPin)

  phsa := phsb := 0
  if clkfreq == 80_000_000
    frqa := 256
  else
    frqa := POSX / CLKFREQ * 2
  frqb := 1
  dira[commPin] := 1
  ctra := constant(%00100<<26) + commPin                ' NCO mode
  ctrb := constant(%01010<<26) + commPin                ' POSEDGE detect

PRI delay_ms(Duration)
  waitcnt(((clkfreq / 1_000 * Duration - 3932)) + cnt)
```

propirc-vgatext.spin

```
{{
  VGA Driver w/ Multiple Regions for IRC Client
  ---------------------------------------------

  Copyright (C) 2006-2008 Harrison Pham

  This file is part of PropTCP-IRC.

  PropTCP-IRC is free software; you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation; either version 3 of the License, or
  (at your option) any later version.

  PropTCP-IRC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.
}}

CON

  cols = vgatext#cols
  rows = vgatext#rows
  chrs = cols * rows

  screensize = cols * rows / 4

  topfirstrow = 1

  toplastrow = rows - 12

  middlelastrow = rows - 11

  lastrow = rows - 1

OBJ

  vgatext : "propirc-vga_hires_text"

VAR

  'sync long - written to -1 by VGA driver after each screen refresh
  long  sync
  'screen buffer - could be bytes, but longs allow more efficient scrolling
  long  screen[screensize]
  'row colors
  word  colors[rows]
  'cursor control bytes
  byte  cx0, cy0, cm0, cx1, cy1, cm1

  long  col, row, color, flag

  long  dbgCol, dbgRow

PUB start(basepin) | i
  vgatext.start(basepin, @screen, @colors, @cx0, @sync)

  cm0 := %110

  '            back fore
  ' colors = %%rgb0_rgb0 where r,g,b from 0 to 3

  colors[0] := %%0030_3300      '%%0000_0300

  repeat i from 1 to toplastrow
    colors[i] := %%0000_1130    '%%0020_3300    '%%0020_3330

  repeat i from toplastrow to middlelastrow
    colors[i] := %%1110_2220    '%%0100_1310    '%%3310_0000

  repeat i from middlelastrow to lastrow
```

```
    colors[i] := %%0000_0100

  out($00)                        ' clear screen

  bytemove(@screen, @titlebar, strsize(@titlebar))

PUB getScreenPtr
'' Gets a pointer to the screen text array
  return @screen

PUB str(stringptr)

'' Print a zero-terminated string

  repeat strsize(stringptr)
    out(byte[stringptr++])

PUB invstr(stringptr)

  repeat strsize(stringptr)
    if byte[stringptr] =< 13
      out(byte[stringptr++])
    else
      out(byte[stringptr++] + 128)

PUB dec(value) | i

'' Print a decimal number

  if value < 0
    -value
    out("-")

  i := 1_000_000_000

  repeat 10
    if value => i
      out(value / i + "0")
      value //= i
      result~~
    elseif result or i == 1
      out("0")
    i /= 10


PUB hex(value, digits)

'' Print a hexadecimal number

  value <<= (8 - digits) << 2
  repeat digits
    out(lookupz((value <-= 4) & $F : "0"..."9", "A"..."F"))


PUB bin(value, digits)

'' Print a binary number

  value <<= 32 - digits
  repeat digits
    out((value <-= 1) & 1 + "0")

PUB out(c) | i, k

'' Output a character
''
''     $00 = clear screen
''     $01 = home
''     $08 = backspace
''     $09 = tab (8 spaces per)
''     $0A = set X position (X follows)
''     $0B = set Y position (Y follows)
''     $0C = set color (color follows)
''     $0D = return
''   others = printable characters

  case flag
    $00: case c
```

```
          $00: longfill(@screen, $20_20_20_20, screensize)
               col := 0
               row := topfirstrow
               dbgCol := 0
               dbgRow := middlelastrow
          $01: col := row := 0
          $08: if col
                 col--
          $09: repeat
                 print(" ")
               while col & 7
          $0A..$0C: flag := c
                    return
          $0D: newline
          other: print(c)
    $0A: col := c // cols
    $0B: row := c // rows
    $0C: color := c & 7
  flag := 0

PRI print(c)

  screen.byte[row * cols + col] := c
  if ++col == cols
    newline

PRI newline | i

  col := 0
  if ++row == toplastrow
    row--
    bytemove(@screen + constant(topfirstrow * cols), @screen + constant((topfirstrow + 1) * cols),
  constant((toplastrow - topfirstrow - 1) * cols))
    bytefill(@screen + constant((toplastrow - 1) * cols), $20, cols)


PUB printChatStr(prompt, ptr) | c, ps

  sync := 0
  repeat until sync

  bytefill(@screen + constant(toplastrow * cols), $20, cols * (middlelastrow - toplastrow))

  ps := strsize(prompt)

  screen.byte[constant(toplastrow * cols)] := "["
  bytemove(@screen + constant(toplastrow * cols + 1), prompt, ps)
  bytemove(@screen + constant(toplastrow * cols + 1) + ps, string("] "), 2)

  c := ps + 3

  ' calculate scroll offset
  ps := c + strsize(ptr)
  'ptr += ps * ((c + ps) / cols)

  if ps => cols
    ptr += (ps // constant(cols - 1)) + (constant(cols - 1) * ((ps - constant(cols - 1)) / constant(cols - 1)))

  repeat
    if byte[ptr] == 0
      quit
    screen.byte[toplastrow * cols + c++] := byte[ptr++]

  cx0 := (c <# constant(cols - 1)) 'c // cols
  cy0 := toplastrow '+ (c / cols)

PUB printDbg(c)

  if c == 13
    dbgNewLine
    return

  'if c =< 13
  '  return
  if c == 10
    return

  screen.byte[dbgRow * cols + dbgCol] := c
```

```
  if ++dbgCol == cols
    dbgNewLine

PRI dbgNewLine

  dbgCol := 0
  if ++dbgRow == rows
    dbgRow--
    bytemove(@screen + constant(middlelastrow * cols), @screen + constant((middlelastrow + 1) * cols),
  constant((lastrow - middlelastrow) * cols))
    bytefill(@screen + constant(lastrow * cols), $20, cols)

DAT

'                          |
  |
titlebar      byte      "PropIRC - The Propeller IRC Client",0
```

propirc-vga_hires_text.spin

```
''***********************************
''*  VGA High-Res Text Driver v1.0  *
''*  (C) 2006 Parallax, Inc.        *
''***********************************
''
'' This object generates a 1024x768 VGA signal which contains 128 columns x 64
'' rows of 8x12 characters. Each row can have a unique forground/background
'' color combination and each character can be inversed. There are also two
'' cursors which can be independently controlled (ie. mouse and keyboard). A
'' sync indicator signals each time the screen is refreshed (you may ignore).
''
'' You must provide buffers for the screen, colors, cursors, and sync. Once
'' started, all interfacing is done via memory. To this object, all buffers are
'' read-only, with the exception of the sync indicator which gets written with
'' -1. You may freely write all buffers to affect screen appearance. Have fun!
''

CON

'{
' 1024 x 768 @ 57Hz settings: 128 x 64 characters

  hp = 1024     'horizontal pixels
  vp = 768      'vertical pixels
  hf = 16       'horizontal front porch pixels
  hs = 96       'horizontal sync pixels
  hb = 176      'horizontal back porch pixels
  vf = 1        'vertical front porch lines
  vs = 3        'vertical sync lines
  vb = 28       'vertical back porch lines
  hn = 1        'horizontal normal sync state (0|1)
  vn = 1        'vertical normal sync state (0|1)
  pr = 60       'pixel rate in MHz at 80MHz system clock (5MHz granularity)
'}
{
' 800 x 600 @ 75Hz settings: 100 x 50 characters

  hp = 800      'horizontal pixels
  vp = 600      'vertical pixels
  hf = 40       'horizontal front porch pixels
  hs = 128      'horizontal sync pixels
  hb = 88       'horizontal back porch pixels
  vf = 1        'vertical front porch lines
  vs = 4        'vertical sync lines
  vb = 23       'vertical back porch lines
  hn = 0        'horizontal normal sync state (0|1)
  vn = 0        'vertical normal sync state (0|1)
  pr = 50       'pixel rate in MHz at 80MHz system clock (5MHz granularity)
}
{
' 640 x 480 @ 69Hz settings: 80 x 40 characters

  hp = 640      'horizontal pixels
  vp = 480      'vertical pixels
  hf = 24       'horizontal front porch pixels
  hs = 40       'horizontal sync pixels
  hb = 128      'horizontal back porch pixels
  vf = 9        'vertical front porch lines
  vs = 3        'vertical sync lines
  vb = 28       'vertical back porch lines
  hn = 1        'horizontal normal sync state (0|1)
  vn = 1        'vertical normal sync state (0|1)
  pr = 30       'pixel rate in MHz at 80MHz system clock (5MHz granularity)
}

' columns and rows

  cols = hp / 8
  rows = vp / 12


VAR long cog[2]

PUB start(BasePin, ScreenPtr, ColorPtr, CursorPtr, SyncPtr) : okay | i, j
```

```
'' Start VGA driver - starts two COGs
'' returns false if two COGs not available
''
''     BasePin = VGA starting pin (0, 8, 16, 24, etc.)
''
''   ScreenPtr = Pointer to 8,192 bytes containing ASCII codes for each of the
''               128x64 screen characters. Each byte's top bit controls color
''               inversion while the lower seven bits provide the ASCII code.
''               Screen memory is arranged left-to-right, top-to-bottom.
''
''               screen byte example: %1_1000001 = inverse "A"
''
''    ColorPtr = Pointer to 64 words which define the foreground and background
''               colors for each row. The lower byte of each word contains the
''               foreground RGB data for that row, while the upper byte
''               contains the background RGB data. The RGB data in each byte is
''               arranged as %RRGGBB00 (4 levels each).
''
''               color word example: %%0020_3300 = gold on blue
''
''   CursorPtr = Pointer to 6 bytes which control the cursors:
''
''               bytes 0,1,2: X, Y, and MODE of cursor 0
''               bytes 3,4,5: X, Y, and MODE of cursor 1
''
''               X and Y are in terms of screen characters
''               (left-to-right, top-to-bottom)
''
''               MODE uses three bottom bits:
''
''                      %x00 = cursor off
''                      %x01 = cursor on
''                      %x10 = cursor on, blink slow
''                      %x11 = cursor on, blink fast
''                      %0xx = cursor is solid block
''                      %1xx = cursor is underscore
''
''               cursor example: 127, 63, %010 = blinking block in lower-right
''
''     SyncPtr = Pointer to long which gets written with -1 upon each screen
''               refresh. May be used to time writes/scrolls, so that chopiness
''               can be avoided. You must clear it each time if you want to see
''               it re-trigger.

  'if driver is already running, stop it
  stop

  'implant pin settings
  reg_vcfg := $200000FF + (BasePin & %111000) << 6
  i := $FF << (BasePin & %011000)
  j := BasePin & %100000 == 0
  reg_dira := i & j
  reg_dirb := i & !j

  'implant CNT value to sync COGs to
  sync_cnt := cnt + $10000

  'implant pointers
  longmove(@screen_base, @ScreenPtr, 3)
  font_base := @font

  'implant unique settings and launch first COG
  vf_lines.byte := vf
  vb_lines.byte := vb
  font_third := 1
  cog[1] := cognew(@d0, SyncPtr) + 1

  'allow time for first COG to launch
  waitcnt($2000 + cnt)

  'differentiate settings and launch second COG
  vf_lines.byte := vf+4
  vb_lines.byte := vb-4
  font_third := 0
  cog[0] := cognew(@d0, SyncPtr) + 1

  'if both COGs launched, return true
```

```
  if cog[0] and cog[1]
    return true

  'else, stop any launched COG and return false
  else
    stop


PUB stop | i

'' Stop VGA driver - frees two COGs

  repeat i from 0 to 1
    if cog[i]
      cogstop(cog[i]~ - 1)


CON

  #1, scanbuff[128], scancode[128*2-1+3], maincode      'enumerate COG RAM usage

  main_size = $1F0 - maincode                           'size of main program

  hv_inactive = (hn << 1 + vn) * $0101                  'H,V inactive states


DAT

'*****************************************************
'* Assembly language VGA high-resolution text driver *
'*****************************************************

' This program runs concurrently in two different COGs.
'
' Each COG's program has different values implanted for front-porch lines and
' back-porch lines which surround the vertical sync pulse lines. This allows
' timed interleaving of their active display signals during the visible portion
' of the field scan. Also, they are differentiated so that one COG displays
' even four-line groups while the other COG displays odd four-line groups.
'
' These COGs are launched in the PUB 'start' and are programmed to synchronize
' their PLL-driven video circuits so that they can alternately prepare sets of
' four scan lines and then display them. The COG-to-COG switchover is seemless
' due to two things: exact synchronization of the two video circuits and the
' fact that all COGs' driven output states get OR'd together, allowing one COG
' to output lows during its preparatory state while the other COG effectively
' drives the pins to create the visible and sync portions of its scan lines.
' During non-visible scan lines, both COGs output together in unison.
'
' COG RAM usage:  $000      = d0 - used to inc destination fields for indirection
'                 $001-$080 = scanbuff - longs which hold 4 scan lines
'                 $081-$182 = scancode - stacked WAITVID/SHR for fast display
'                 $183-$1EF = maincode - main program loop which drives display

                        org                             'set origin to $000 for start of program

d0                      long    1 << 9                  'd0 always resides here at $000, executes as NOP


' Initialization code and data - after execution, space gets reused as scanbuff

                        'Move main program into maincode area

:move                   mov     $1EF,main_begin+main_size-1
                        sub     :move,d0s0              '(do reverse move to avoid overwrite)
                        djnz    main_ctr,#:move

                        'Build scanbuff display routine into scancode

:waitvid                mov     scancode+0,i0           'org     scancode
:shr                    mov     scancode+1,i1           'waitvid color,scanbuff+0
                        add     :waitvid,d1             'shr     scanbuff+0,#8
                        add     :shr,d1                 'waitvid color,scanbuff+1
                        add     i0,#1                   'shr     scanbuff+1,#8
                        add     i1,d0                   '...
                        djnz    scan_ctr,#:waitvid      'waitvid color,scanbuff+cols-1

                        mov     scancode+cols*2-1,i2    'mov     vscl,#hf
```

```
                        mov     scancode+cols*2+0,i3    'waitvid hvsync,#0
                        mov     scancode+cols*2+1,i4    'jmp     #scanret

                        'Init I/O registers and sync COGs' video circuits

                        mov     dira,reg_dira           'set pin directions
                        mov     dirb,reg_dirb
                        movi    frqa,#(pr / 5) << 2     'set pixel rate
                        mov     vcfg,reg_vcfg           'set video configuration
                        mov     vscl,#1                 'set video to reload on every pixel
                        waitcnt sync_cnt,colormask      'wait for start value in cnt, add ~1ms
                        movi    ctra,#%00001_110        'COGs in sync! enable PLLs now - NCOs locked!
                        waitcnt sync_cnt,#0             'wait ~1ms for PLLs to stabilize - PLLs locked!
                        mov     vscl,#100               'insure initial WAITVIDs lock cleanly

                        'Jump to main loop

                        jmp     #vsync                  'jump to vsync - WAITVIDs will now be locked!

                        'Data

d0s0                    long    1 << 9 + 1
d1                      long    1 << 10
main_ctr                long    main_size
scan_ctr                long    cols

i0                      waitvid x,scanbuff+0
i1                      shr     scanbuff+0,#8
i2                      mov     vscl,#hf
i3                      waitvid hvsync,#0
i4                      jmp     #scanret

reg_dira                long    0                       'set at runtime
reg_dirb                long    0                       'set at runtime
reg_vcfg                long    0                       'set at runtime
sync_cnt                long    0                       'set at runtime

                        'Directives

                        fit     scancode                'make sure initialization code and data fit
main_begin              org     maincode                'main code follows (gets moved into maincode)


' Main loop, display field - each COG alternately builds and displays four scan lines

vsync                   mov     x,#vs                   'do vertical sync lines
                        call    #blank_vsync

vb_lines                mov     x,#vb                   'do vertical back porch lines (# set at runtime)
                        call    #blank_vsync

                        mov     screen_ptr,screen_base  'reset screen pointer to upper-left character
                        mov     color_ptr,color_base    'reset color pointer to first row
                        mov     row,#0                  'reset row counter for cursor insertion
                        mov     fours,#rows * 3 / 2     'set number of 4-line builds for whole screen

                        'Build four scan lines into scanbuff

fourline                mov     font_ptr,font_third     'get address of appropriate font section
                        shl     font_ptr,#7+2
                        add     font_ptr,font_base

                        movd    :pixa,#scanbuff-1       'reset scanbuff address (pre-decremented)
                        movd    :pixb,#scanbuff-1

                        mov     y,#2                    'must build scanbuff in two sections because
                        mov     vscl,vscl_line2x        '..pixel counter is limited to twelve bits

:halfrow                waitvid underscore,#0           'output lows to let other COG drive VGA pins
                        mov     x,#cols/2               '..for 2 scan lines, ready for half a row

:column                 rdbyte  z,screen_ptr            'get character from screen memory
                        ror     z,#7                    'get inverse flag into bit 0, keep chr high
                        shr     z,#32-7-2       wc      'get inverse flag into c, chr into bits 8..2
                        add     z,font_ptr              'add font section address to point to 8*4 pixels
                        add     :pixa,d0                'increment scanbuff destination addresses
                        add     :pixb,d0
                        add     screen_ptr,#1           'increment screen memory address
```

```
:pixa                   rdlong  scanbuff,z              'read pixel long (8*4) into scanbuff
:pixb   if_nc           xor     scanbuff,longmask       'invert pixels according to inverse flag
                        djnz    x,#:column              'another character in this half-row?

                        djnz    y,#:halfrow             'loop to do 2nd half-row, time for 2nd WAITVID

                        sub     screen_ptr,#cols        'back up to start of same row in screen memory

                        'Insert cursors into scanbuff

                        mov     z,#2                    'ready for two cursors

:cursor                 rdbyte  x,cursor_base           'x in range?
                        add     cursor_base,#1
                        cmp     x,#cols         wc

                        rdbyte  y,cursor_base           'y match?
                        add     cursor_base,#1
                        cmp     y,row           wz

                        rdbyte  y,cursor_base           'get cursor mode
                        add     cursor_base,#1

        if_nc_or_nz     jmp     #:nocursor              'if cursor not in scanbuff, no cursor

                        add     x,#scanbuff             'cursor in scanbuff, set scanbuff address
                        movd    :xor,x

                        test    y,#%010         wc      'get mode bits into flags
                        test    y,#%001         wz
        if_nc_and_z     jmp     #:nocursor              'if cursor disabled, no cursor

        if_c_and_z      test    slowbit,cnt     wc      'if blink mode, get blink state
        if_c_and_nz     test    fastbit,cnt     wc

                        test    y,#%100         wz      'get box or underscore cursor piece
        if_z            mov     x,longmask
        if_nz           mov     x,underscore
        if_nz           cmp     font_third,#2   wz      'if underscore, must be last font section

:xor    if_nc_and_z     xor     scanbuff,x              'conditionally xor cursor into scanbuff

:nocursor               djnz    z,#:cursor              'second cursor?

                        sub     cursor_base,#3*2        'restore cursor base

                        'Display four scan lines from scanbuff

                        rdword  x,color_ptr             'get color pattern for current row
                        and     x,colormask             'mask away hsync and vsync signal states
                        or      x,hv                    'insert inactive hsync and vsync states

                        mov     y,#4                    'ready for four scan lines

scanline                mov     vscl,vscl_chr           'set pixel rate for characters
                        jmp     #scancode               'jump to scanbuff display routine in scancode
scanret                 mov     vscl,#hs                'do horizontal sync pixels
                        waitvid hvsync,#1               '#1 makes hsync active
                        mov     vscl,#hb                'do horizontal back porch pixels
                        waitvid hvsync,#0               '#0 makes hsync inactive
                        shr     scanbuff+cols-1,#8      'shift last column's pixels right by 8
                        djnz    y,#scanline             'another scan line?

                        'Next group of four scan lines

                        add     font_third,#2           'if font_third + 2 => 3, subtract 3 (new row)
                        cmpsub  font_third,#3   wc      'c=0 for same row, c=1 for new row
        if_c            add     screen_ptr,#cols        'if new row, advance screen pointer
        if_c            add     color_ptr,#2            'if new row, advance color pointer
        if_c            add     row,#1                  'if new row, increment row counter
                        djnz    fours,#fourline         'another 4-line build/display?

                        'Visible section done, do vertical sync front porch lines

                        wrlong  longmask,par            'write -1 to refresh indicator

vf_lines                mov     x,#vf                   'do vertical front porch lines (# set at runtime)
                        call    #blank
```

```
                        jmp     #vsync                  'new field, loop to vsync

                        'Subroutine - do blank lines

blank_vsync             xor     hvsync,#$101            'flip vertical sync bits

blank                   mov     vscl,hx                 'do blank pixels
                        waitvid hvsync,#0
                        mov     vscl,#hf                'do horizontal front porch pixels
                        waitvid hvsync,#0
                        mov     vscl,#hs                'do horizontal sync pixels
                        waitvid hvsync,#1
                        mov     vscl,#hb                'do horizontal back porch pixels
                        waitvid hvsync,#0
                        djnz    x,#blank                'another line?
blank_ret
blank_vsync_ret         ret

                        'Data

screen_base             long    0                       'set at runtime (3 contiguous longs)
color_base              long    0                       'set at runtime
cursor_base             long    0                       'set at runtime

font_base               long    0                       'set at runtime
font_third              long    0                       'set at runtime

hx                      long    hp                      'visible pixels per scan line
vscl_line2x             long    (hp + hf + hs + hb) * 2 'total number of pixels per 2 scan lines
vscl_chr                long    1 << 12 + 8             '1 clock per pixel and 8 pixels per set
colormask               long    $FCFC                   'mask to isolate R,G,B bits from H,V
longmask                long    $FFFFFFFF               'all bits set
slowbit                 long    1 << 25                 'cnt mask for slow cursor blink
fastbit                 long    1 << 24                 'cnt mask for fast cursor blink
underscore              long    $FFFF0000               'underscore cursor pattern
hv                      long    hv_inactive             '-H,-V states
hvsync                  long    hv_inactive ^ $200      '+/-H,-V states

                        'Uninitialized data

screen_ptr              res     1
color_ptr               res     1
font_ptr                res     1

x                       res     1
y                       res     1
z                       res     1

row                     res     1
fours                   res     1


' 8 x 12 font - characters 0..127
'
' Each long holds four scan lines of a single character. The longs are arranged into
' groups of 128 which represent all characters (0..127). There are three groups which
' each contain a vertical third of all characters. They are ordered top, middle, and
' bottom.

font  long

long  $0C080000,$30100000,$7E3C1800,$18181800,$81423C00,$99423C00,$8181FF00,$E7C3FF00  'top
long  $1E0E0602,$1C000000,$00000000,$00000000,$18181818,$18181818,$00000000,$18181818
long  $00000000,$18181818,$18181818,$18181818,$18181818,$00FFFF00,$CC993366,$66666666
long  $AA55AA55,$0F0F0F0F,$0F0F0F0F,$0F0F0F0F,$0F0F0F0F,$00000000,$00000000,$00000000
long  $00000000,$3C3C1800,$77666600,$7F363600,$667C1818,$46000000,$1B1B0E00,$1C181800
long  $0C183000,$180C0600,$66000000,$18000000,$00000000,$00000000,$00000000,$60400000
long  $73633E00,$1E181000,$66663C00,$60663C00,$3C383000,$06067E00,$060C3800,$63637F00
long  $66663C00,$66663C00,$1C000000,$00000000,$18306000,$00000000,$180C0600,$60663C00
long  $63673E00,$66663C00,$66663F00,$63663C00,$66361F00,$06467F00,$06467F00,$63663C00
long  $63636300,$18183C00,$30307800,$36666700,$06060F00,$7F776300,$67636300,$63361C00
long  $66663F00,$63361C00,$66663F00,$66663C00,$185A7E00,$66666600,$66666600,$63636300
long  $66666600,$66666600,$31637F00,$0C0C3C00,$03010000,$30303C00,$361C0800,$00000000
long  $0C000000,$00000000,$06060700,$00000000,$30303800,$00000000,$0C6C3800,$00000000
long  $06060700,$00181800,$00606000,$06060700,$18181E00,$00000000,$00000000,$00000000
long  $00000000,$00000000,$00000000,$00000000,$0C080000,$00000000,$00000000,$00000000
long  $00000000,$00000000,$00000000,$18187000,$18181800,$18180E00,$73DBCE00,$18180000
```

```
long  $080C7E7E,$10307E7E,$18181818,$7E181818,$81818181,$99BDBDBD,$81818181,$E7BD99BD  'middle
long  $1E3E7E3E,$1C3E3E3E,$30F0C000,$0C0F0300,$00C0F030,$00030F0C,$00FFFF00,$18181818
long  $18FFFF00,$00FFFF18,$18F8F818,$181F1F18,$18FFFF18,$00FFFF00,$CC993366,$66666666
long  $AA55AA55,$FFFF0F0F,$F0F00F0F,$0F0F0F0F,$00000F0F,$FFFF0000,$F0F00000,$0F0F0000
long  $00000000,$0018183C,$00000033,$7F363636,$66603C06,$0C183066,$337B5B0E,$0000000C
long  $0C060606,$18303030,$663CFF3C,$18187E18,$00000000,$00007E00,$00000000,$060C1830
long  $676F6B7B,$18181818,$0C183060,$60603860,$307F3336,$60603E06,$66663E06,$0C183060
long  $66763C6E,$60607C66,$1C00001C,$00001C1C,$180C060C,$007E007E,$18306030,$00181830
long  $033B7B7B,$66667E66,$66663E66,$63030303,$66666666,$06263E26,$06263E26,$63730303
long  $63637F63,$18181818,$33333030,$36361E36,$66460606,$63636B7F,$737B7F6F,$63636363
long  $06063E66,$7B636363,$66363E66,$66301C06,$18181818,$66666666,$66666666,$366B6B63
long  $663C183C,$18183C66,$43060C18,$0C0C0C0C,$30180C06,$30303030,$00000063,$00000000
long  $0030381C,$333E301E,$6666663E,$0606663C,$3333333E,$067E663C,$0C0C3E0C,$3333336E
long  $66666E36,$1818181C,$60606070,$361E3666,$18181818,$6B6B6B3F,$6666663E,$6666663C
long  $6666663B,$3333336E,$066E7637,$300C663C,$0C0C0C7E,$33333333,$66666666,$6B6B6363
long  $1C1C3663,$66666666,$0C30627E,$180C060C,$18181818,$18306030,$00000000,$0018187E

long  $00000000,$00000000,$00001818,$0000183C,$00003C42,$00003C42,$0000FF81,$0000FFC3  'bottom
long  $0002060E,$00000000,$18181818,$18181818,$00000000,$00000000,$00000000,$18181818
long  $18181818,$00000000,$18181818,$18181818,$18181818,$00FFFF00,$CC993366,$66666666
long  $AA55AA55,$FFFFFFFF,$F0F0F0F0,$0F0F0F0F,$00000000,$FFFFFFFF,$F0F0F0F0,$0F0F0F0F
long  $00000000,$00001818,$00000000,$00003636,$0018183E,$00006266,$00006E3B,$00000000
long  $00003018,$0000060C,$00000000,$00000000,$0C181C1C,$00000000,$00001C1C,$00000103
long  $00003E63,$00007E18,$00007E66,$00003C66,$00007830,$00003C66,$00003C66,$00000C0C
long  $00003C66,$00001C30,$0000001C,$0C181C1C,$00006030,$00000000,$0000060C,$00001818
long  $00003E07,$00006666,$00003F66,$00003C66,$00001F36,$00007F46,$00000F06,$00007C66
long  $00006363,$00003C18,$00001E33,$00006766,$00007F66,$00006363,$00006363,$00001C36
long  $00000F06,$00603C36,$00006766,$00003C66,$00003C18,$00003C66,$0000183C,$00003636
long  $00006666,$00003C18,$00007F63,$00003C0C,$00004060,$00003C30,$00000000,$FFFF0000
long  $00000000,$00006E33,$00003B66,$00003C66,$00006E33,$00003C66,$00001E0C,$1E33303E
long  $00006766,$00007E18,$3C666660,$00006766,$00007E18,$00006B6B,$00006666,$00003C66
long  $0F063E66,$78303E33,$00000F06,$00003C66,$0000386C,$00006E33,$0000183C,$00003636
long  $00006336,$1C30607C,$00007E46,$00007018,$00001818,$00000E18,$00000000,$0000007E
```

keyboard.spin

```
''*********************************
''*  PS/2 Keyboard Driver v1.0.1  *
''*  (C) 2004 Parallax, Inc.      *
''*********************************

{-----------------REVISION HISTORY-----------------
 v1.0.1 - Updated 6/15/2006 to work with Propeller Tool 0.96}

VAR

  long  cog

  long  par_tail        'key buffer tail        read/write      (19 contiguous longs)
  long  par_head        'key buffer head        read-only
  long  par_present     'keyboard present       read-only
  long  par_states[8]   'key states (256 bits)  read-only
  long  par_keys[8]     'key buffer (16 words)  read-only       (also used to pass initial parameters)


PUB start(dpin, cpin) : okay

'' Start keyboard driver - starts a cog
'' returns false if no cog available
''
''   dpin  = data signal on PS/2 jack
''   cpin  = clock signal on PS/2 jack
''
''     use 100-ohm resistors between pins and jack
''     use 10K-ohm resistors to pull jack-side signals to VDD
''     connect jack-power to 5V, jack-gnd to VSS
''
'' all lock-keys will be enabled, NumLock will be initially 'on',
'' and auto-repeat will be set to 15cps with a delay of .5s

  okay := startx(dpin, cpin, %0_000_100, %01_01000)


PUB startx(dpin, cpin, locks, auto) : okay

'' Like start, but allows you to specify lock settings and auto-repeat
''
''   locks = lock setup
''           bit 6 disallows shift-alphas (case set soley by CapsLock)
''           bits 5..3 disallow toggle of NumLock/CapsLock/ScrollLock state
''           bits 2..0 specify initial state of NumLock/CapsLock/ScrollLock
''           (eg. %0_001_100 = disallow ScrollLock, NumLock initially 'on')
''
''   auto  = auto-repeat setup
''           bits 6..5 specify delay (0=.25s, 1=.5s, 2=.75s, 3=1s)
''           bits 4..0 specify repeat rate (0=30cps..31=2cps)
''           (eg %01_00000 = .5s delay, 30cps repeat)

  stop
  longmove(@par_keys, @dpin, 4)
  okay := cog := cognew(@entry, @par_tail) + 1


PUB stop

'' Stop keyboard driver - frees a cog

  if cog
    cogstop(cog~ - 1)
  longfill(@par_tail, 0, 19)


PUB present : truefalse

'' Check if keyboard present - valid ~2s after start
'' returns t|f

  truefalse := -par_present
```

```
PUB key : keycode

'' Get key (never waits)
'' returns key (0 if buffer empty)

  if par_tail <> par_head
    keycode := par_keys.word[par_tail]
    par_tail := ++par_tail & $F


PUB getkey : keycode

'' Get next key (may wait for keypress)
'' returns key

  repeat until (keycode := key)


PUB newkey : keycode

'' Clear buffer and get new key (always waits for keypress)
'' returns key

  par_tail := par_head
  keycode := getkey


PUB gotkey : truefalse

'' Check if any key in buffer
'' returns t|f

  truefalse := par_tail <> par_head


PUB clearkeys

'' Clear key buffer

  par_tail := par_head


PUB keystate(k) : state

'' Get the state of a particular key
'' returns t|f

  state := -(par_states[k >> 5] >> k & 1)


DAT

'*****************************************
'* Assembly language PS/2 keyboard driver *
'*****************************************

                        org
'
'
' Entry
'
entry                   movd    :par,#_dpin             'load input parameters _dpin/_cpin/_locks/_auto
                        mov     x,par
                        add     x,#11*4
                        mov     y,#4
:par                    rdlong  0,x
                        add     :par,dlsb
                        add     x,#4
                        djnz    y,#:par

                        mov     dmask,#1                'set pin masks
                        shl     dmask,_dpin
                        mov     cmask,#1
                        shl     cmask,_cpin

                        test    _dpin,#$20      wc      'modify port registers within code
                        muxc    _d1,dlsb
                        muxc    _d2,dlsb
```

```
                        muxc    _d3,#1
                        muxc    _d4,#1
                        test    _cpin,#$20      wc
                        muxc    _c1,dlsb
                        muxc    _c2,dlsb
                        muxc    _c3,#1

                        mov     _head,#0                'reset output parameter _head
'
'
' Reset keyboard
'
reset                   mov     dira,#0                 'reset directions
                        mov     dirb,#0

                        movd    :par,#_present          'reset output parameters _present/_states[8]
                        mov     x,#1+8
:par                    mov     0,#0
                        add     :par,dlsb
                        djnz    x,#:par

                        mov     stat,#8                 'set reset flag
'
'
' Update parameters
'
update                  movd    :par,#_head             'update output parameters _head/_present/_states[8]
                        mov     x,par
                        add     x,#1*4
                        mov     y,#1+1+8
:par                    wrlong  0,x
                        add     :par,dlsb
                        add     x,#4
                        djnz    y,#:par

                        test    stat,#8         wc      'if reset flag, transmit reset command
        if_c            mov     data,#$FF
        if_c            call    #transmit
'
'
' Get scancode
'
newcode                 mov     stat,#0                 'reset state

:same                   call    #receive                'receive byte from keyboard

                        cmp     data,#$83+1     wc      'scancode?

        if_nc           cmp     data,#$AA       wz      'powerup/reset?
        if_nc_and_z     jmp     #configure

        if_nc           cmp     data,#$E0       wz      'extended?
        if_nc_and_z     or      stat,#1
        if_nc_and_z     jmp     #:same

        if_nc           cmp     data,#$F0       wz      'released?
        if_nc_and_z     or      stat,#2
        if_nc_and_z     jmp     #:same

        if_nc           jmp     #newcode                'unknown, ignore
'
'
' Translate scancode and enter into buffer
'
                        test    stat,#1         wc      'lookup code with extended flag
                        rcl     data,#1
                        call    #look

                        cmp     data,#0         wz      'if unknown, ignore
        if_z            jmp     #newcode

                        mov     t,_states+6             'remember lock keys in _states

                        mov     x,data                  'set/clear key bit in _states
                        shr     x,#5
                        add     x,#_states
                        movd    :reg,x
                        mov     y,#1
```

```
                        shl     y,data
                        test    stat,#2         wc
:reg                    muxnc   0,y

        if_nc           cmpsub  data,#$F0       wc      'if released or shift/ctrl/alt/win, done
        if_c            jmp     #update

                        mov     y,_states+7             'get shift/ctrl/alt/win bit pairs
                        shr     y,#16

                        cmpsub  data,#$E0       wc      'translate keypad, considering numlock
        if_c            test    _locks,#%100    wz
        if_c_and_z      add     data,#@keypad1-@table
        if_c_and_nz     add     data,#@keypad2-@table
        if_c            call    #look
        if_c            jmp     #:flags

                        cmpsub  data,#$DD       wc      'handle scrlock/capslock/numlock
        if_c            mov     x,#%001_000
        if_c            shl     x,data
        if_c            andn    x,_locks
        if_c            shr     x,#3
        if_c            shr     t,#29                   'ignore auto-repeat
        if_c            andn    x,t             wz
        if_c            xor     _locks,x
        if_c            add     data,#$DD
        if_c_and_nz     or      stat,#4                 'if change, set configure flag to update leds

                        test    y,#%11          wz      'get shift into nz

        if_nz           cmp     data,#$60+1     wc      'check shift1
        if_nz_and_c     cmpsub  data,#$5B       wc
        if_nz_and_c     add     data,#@shift1-@table
        if_nz_and_c     call    #look
        if_nz_and_c     andn    y,#%11

        if_nz           cmp     data,#$3D+1     wc      'check shift2
        if_nz_and_c     cmpsub  data,#$27       wc
        if_nz_and_c     add     data,#@shift2-@table
        if_nz_and_c     call    #look
        if_nz_and_c     andn    y,#%11

                        test    _locks,#%010    wc      'check shift-alpha, considering capslock
                        muxnc   :shift,#$20
                        test    _locks,#$40     wc
        if_nz_and_nc    xor     :shift,#$20
                        cmp     data,#"z"+1     wc
        if_c            cmpsub  data,#"a"       wc
:shift  if_c            add     data,#"A"
        if_c            andn    y,#%11

:flags                  ror     data,#8                 'add shift/ctrl/alt/win flags
                        mov     x,#4                    '+$100 if shift
:loop                   test    y,#%11          wz      '+$200 if ctrl
                        shr     y,#2                    '+$400 if alt
        if_nz           or      data,#1                 '+$800 if win
                        ror     data,#1
                        djnz    x,#:loop
                        rol     data,#12

                        rdlong  x,par                   'if room in buffer and key valid, enter
                        sub     x,#1
                        and     x,#$F
                        cmp     x,_head         wz
        if_nz           test    data,#$FF       wz
        if_nz           mov     x,par
        if_nz           add     x,#11*4
        if_nz           add     x,_head
        if_nz           add     x,_head
        if_nz           wrword  data,x
        if_nz           add     _head,#1
        if_nz           and     _head,#$F

                        test    stat,#4         wc      'if not configure flag, done
        if_nc           jmp     #update                 'else configure to update leds
'
'
' Configure keyboard
```

```
'
configure               mov     data,#$F3               'set keyboard auto-repeat
                        call    #transmit
                        mov     data,_auto
                        and     data,#%11_11111
                        call    #transmit

                        mov     data,#$ED               'set keyboard lock-leds
                        call    #transmit
                        mov     data,_locks
                        rev     data,#-3 & $1F
                        test    data,#%100      wc
                        rcl     data,#1
                        and     data,#%111
                        call    #transmit

                        mov     x,_locks                'insert locks into _states
                        and     x,#%111
                        shl     _states+7,#3
                        or      _states+7,x
                        ror     _states+7,#3

                        mov     _present,#1             'set _present

                        jmp     #update                 'done
'
'
' Lookup byte in table
'
look                    ror     data,#2                 'perform lookup
                        movs    :reg,data
                        add     :reg,#table
                        shr     data,#27
                        mov     x,data
:reg                    mov     data,0
                        shr     data,x

                        jmp     #rand                   'isolate byte
'
'
' Transmit byte to keyboard
'
transmit
_c1                     or      dira,cmask              'pull clock low
                        movs    napshr,#13              'hold clock for ~128us (must be >100us)
                        call    #nap
_d1                     or      dira,dmask              'pull data low
                        movs    napshr,#18              'hold data for ~4us
                        call    #nap
_c2                     xor     dira,cmask              'release clock

                        test    data,#$0FF      wc      'append parity and stop bits to byte
                        muxnc   data,#$100
                        or      data,dlsb

                        mov     x,#10                   'ready 10 bits
transmit_bit            call    #wait_c0                'wait until clock low
                        shr     data,#1         wc      'output data bit
_d2                     muxnc   dira,dmask
                        mov     wcond,c1                'wait until clock high
                        call    #wait
                        djnz    x,#transmit_bit         'another bit?

                        mov     wcond,c0d0              'wait until clock and data low
                        call    #wait
                        mov     wcond,c1d1              'wait until clock and data high
                        call    #wait

                        call    #receive_ack            'receive ack byte with timed wait
                        cmp     data,#$FA       wz      'if ack error, reset keyboard
        if_nz           jmp     #reset

transmit_ret            ret
'
'
' Receive byte from keyboard
'
receive                 test    _cpin,#$20      wc      'wait indefinitely for initial clock low
```

```
                        waitpne cmask,cmask
receive_ack
                        mov     x,#11                   'ready 11 bits
receive_bit             call    #wait_c0                'wait until clock low
                        movs    napshr,#16              'pause ~16us
                        call    #nap
_d3                     test    dmask,ina       wc      'input data bit
                        rcr     data,#1
                        mov     wcond,c1                'wait until clock high
                        call    #wait
                        djnz    x,#receive_bit          'another bit?

                        shr     data,#22                'align byte
                        test    data,#$1FF      wc      'if parity error, reset keyboard
        if_nc           jmp     #reset
rand                    and     data,#$FF               'isolate byte

look_ret
receive_ack_ret
receive_ret             ret
'
'
' Wait for clock/data to be in required state(s)
'
wait_c0                 mov     wcond,c0                '(wait until clock low)

wait                    mov     y,tenms                 'set timeout to 10ms

wloop                   movs    napshr,#18              'nap ~4us
                        call    #nap
_c3                     test    cmask,ina       wc      'check required state(s)
_d4                     test    dmask,ina       wz      'loop until got state(s) or timeout
wcond   if_never        djnz    y,#wloop                '(replaced with c0/c1/c0d0/c1d1)

                        tjz     y,#reset                'if timeout, reset keyboard
wait_ret
wait_c0_ret             ret


c0      if_c            djnz    y,#wloop                '(if_never replacements)
c1      if_nc           djnz    y,#wloop
c0d0    if_c_or_nz      djnz    y,#wloop
c1d1    if_nc_or_z      djnz    y,#wloop
'
'
' Nap
'
nap                     rdlong  t,#0                    'get clkfreq
napshr                  shr     t,#18/16/13             'shr scales time
                        min     t,#3                    'ensure waitcnt won't snag
                        add     t,cnt                   'add cnt to time
                        waitcnt t,#0                    'wait until time elapses (nap)

nap_ret                 ret
'
'
' Initialized data
'
'
dlsb                    long    1 << 9
tenms                   long    10_000 / 4
'
'
' Lookup table
'                               ascii   scan    extkey  regkey  ()=keypad
'
table                   word    $0000   '00
                        word    $00D8   '01             F9
                        word    $0000   '02
                        word    $00D4   '03             F5
                        word    $00D2   '04             F3
                        word    $00D0   '05             F1
                        word    $00D1   '06             F2
                        word    $00DB   '07             F12
                        word    $0000   '08
                        word    $00D9   '09             F10
                        word    $00D7   '0A             F8
                        word    $00D5   '0B             F6
```

```
                word    $00D3   '0C             F4
                word    $0009   '0D             Tab
                word    $0060   '0E             `
                word    $0000   '0F
                word    $0000   '10
                word    $F5F4   '11     Alt-R   Alt-L
                word    $00F0   '12             Shift-L
                word    $0000   '13
                word    $F3F2   '14     Ctrl-R  Ctrl-L
                word    $0071   '15             q
                word    $0031   '16             1
                word    $0000   '17
                word    $0000   '18
                word    $0000   '19
                word    $007A   '1A             z
                word    $0073   '1B             s
                word    $0061   '1C             a
                word    $0077   '1D             w
                word    $0032   '1E             2
                word    $F600   '1F     Win-L
                word    $0000   '20
                word    $0063   '21             c
                word    $0078   '22             x
                word    $0064   '23             d
                word    $0065   '24             e
                word    $0034   '25             4
                word    $0033   '26             3
                word    $F700   '27     Win-R
                word    $0000   '28
                word    $0020   '29             Space
                word    $0076   '2A             v
                word    $0066   '2B             f
                word    $0074   '2C             t
                word    $0072   '2D             r
                word    $0035   '2E             5
                word    $CC00   '2F     Apps
                word    $0000   '30
                word    $006E   '31             n
                word    $0062   '32             b
                word    $0068   '33             h
                word    $0067   '34             g
                word    $0079   '35             y
                word    $0036   '36             6
                word    $CD00   '37     Power
                word    $0000   '38
                word    $0000   '39
                word    $006D   '3A             m
                word    $006A   '3B             j
                word    $0075   '3C             u
                word    $0037   '3D             7
                word    $0038   '3E             8
                word    $CE00   '3F     Sleep
                word    $0000   '40
                word    $002C   '41             ,
                word    $006B   '42             k
                word    $0069   '43             i
                word    $006F   '44             o
                word    $0030   '45             0
                word    $0039   '46             9
                word    $0000   '47
                word    $0000   '48
                word    $002E   '49             .
                word    $EF2F   '4A     (/)     /
                word    $006C   '4B             l
                word    $003B   '4C             ;
                word    $0070   '4D             p
                word    $002D   '4E             -
                word    $0000   '4F
                word    $0000   '50
                word    $0000   '51
                word    $0027   '52             '
                word    $0000   '53
                word    $005B   '54             [
                word    $003D   '55             =
                word    $0000   '56
                word    $0000   '57
                word    $00DE   '58             CapsLock
                word    $00F1   '59             Shift-R
```

```
                        word    $EB0D   '5A     (Enter) Enter
                        word    $005D   '5B             ]
                        word    $0000   '5C
                        word    $005C   '5D             \
                        word    $CF00   '5E     WakeUp
                        word    $0000   '5F
                        word    $0000   '60
                        word    $0000   '61
                        word    $0000   '62
                        word    $0000   '63
                        word    $0000   '64
                        word    $0000   '65
                        word    $00C8   '66             BackSpace
                        word    $0000   '67
                        word    $0000   '68
                        word    $C5E1   '69     End     (1)
                        word    $0000   '6A
                        word    $C0E4   '6B     Left    (4)
                        word    $C4E7   '6C     Home    (7)
                        word    $0000   '6D
                        word    $0000   '6E
                        word    $0000   '6F
                        word    $CAE0   '70     Insert  (0)
                        word    $C9EA   '71     Delete  (.)
                        word    $C3E2   '72     Down    (2)
                        word    $00E5   '73             (5)
                        word    $C1E6   '74     Right   (6)
                        word    $C2E8   '75     Up      (8)
                        word    $00CB   '76             Esc
                        word    $00DF   '77             NumLock
                        word    $00DA   '78             F11
                        word    $00EC   '79             (+)
                        word    $C7E3   '7A     PageDn  (3)
                        word    $00ED   '7B             (-)
                        word    $DCEE   '7C     PrScr   (*)
                        word    $C6E9   '7D     PageUp  (9)
                        word    $00DD   '7E             ScrLock
                        word    $0000   '7F
                        word    $0000   '80
                        word    $0000   '81
                        word    $0000   '82
                        word    $00D6   '83             F7

keypad1                 byte    $CA, $C5, $C3, $C7, $C0, 0, $C1, $C4, $C2, $C6, $C9, $0D, "+-*/"

keypad2                 byte    "0123456789.", $0D, "+-*/"

shift1                  byte    "{|}", 0, 0, "~"

shift2                  byte    $22, 0, 0, 0, 0, "<_>?)!@#$%^&*(", 0, ":", 0, "+"
'
'
' Uninitialized data
'
dmask                   res     1
cmask                   res     1
stat                    res     1
data                    res     1
x                       res     1
y                       res     1
t                       res     1

_head                   res     1       'write-only
_present                res     1       'write-only
_states                 res     8       'write-only
_dpin                   res     1       'read-only at start
_cpin                   res     1       'read-only at start
_locks                  res     1       'read-only at start
_auto                   res     1       'read-only at start
''
''
''
''      _________
''      Key Codes
''
''      00..DF  = keypress and keystate
''      E0..FF  = keystate only
''
```

```
''
''      09      Tab
''      0D      Enter
''      20      Space
''      21      !
''      22      "
''      23      #
''      24      $
''      25      %
''      26      &
''      27      '
''      28      (
''      29      )
''      2A      *
''      2B      +
''      2C      ,
''      2D      -
''      2E      .
''      2F      /
''      30      0..9
''      3A      :
''      3B      ;
''      3C      <
''      3D      =
''      3E      >
''      3F      ?
''      40      @
''      41..5A  A..Z
''      5B      [
''      5C      \
''      5D      ]
''      5E      ^
''      5F      _
''      60      `
''      61..7A  a..z
''      7B      {
''      7C      |
''      7D      }
''      7E      ~
''
''      80-BF   (future international character support)
''
''      C0      Left Arrow
''      C1      Right Arrow
''      C2      Up Arrow
''      C3      Down Arrow
''      C4      Home
''      C5      End
''      C6      Page Up
''      C7      Page Down
''      C8      Backspace
''      C9      Delete
''      CA      Insert
''      CB      Esc
''      CC      Apps
''      CD      Power
''      CE      Sleep
''      CF      Wakeup
''
''      D0..DB  F1..F12
''      DC      Print Screen
''      DD      Scroll Lock
''      DE      Caps Lock
''      DF      Num Lock
''
''      E0..E9  Keypad 0..9
''      EA      Keypad .
''      EB      Keypad Enter
''      EC      Keypad +
''      ED      Keypad -
''      EE      Keypad *
''      EF      Keypad /
''
''      F0      Left Shift
''      F1      Right Shift
''      F2      Left Ctrl
''      F3      Right Ctrl
''      F4      Left Alt
```

```
''      F5      Right Alt
''      F6      Left Win
''      F7      Right Win
''
''      FD      Scroll Lock State
''      FE      Caps Lock State
''      FF      Num Lock State
''
''      +100    if Shift
''      +200    if Ctrl
''      +400    if Alt
''      +800    if Win
''
''      eg. Ctrl-Alt-Delete = $6C9
''
''
'' Note: Driver will buffer up to 15 keystrokes, then ignore overflow.
```

propirc-eepromvar.spin

```
{{
File: Propeller Eeprom.spin
Version: 0.6

Developed for forthcoming Propeller Education Kit Lab: EEPROM Datalogging and I2C

See Propeller Eeprom Docs.spin for explanation and instructions.  To view this
object, press, and the file should appear in the Object Info window's Object Explorer
pane.  Double-click it to open.

}}

CON

  ' 24LC256 EEPROM constants

  SDA = 29                                        ' P29 = data line
  SCL = 28                                        ' P28 = clock line
  ACK = 0                                         ' Acknowledge bit = 0
  NACK = 1                                        '  (No) acknowledge bit = 1

PUB VarBackup(startAddr, endAddr) | addr, page, eeAddr

  ''Copy contents of address range defined by startAddr..endAddr from main RAM to EEPROM.

  FromRam(startAddr, endAddr, startAddr)            ' Pass addresses to the Write method

PUB VarRestore(startAddr, endAddr) | addr

  ''Copy contents of address range defined by startAddr..endAddr from EEPROM to RAM.

  ToRam(startAddr, endAddr, startAddr)             ' Pass addresses to the read method

PUB FromRam(startAddr, endAddr, eeStart) | addr, page, eeAddr

  ''Copy startAddr..endAddr from main RAM to EEPROM beginning at eeStart address.

  addr := startAddr                               ' Initialize main RAM index
  eeAddr := eeStart                               ' Initialize EEPROM index
  repeat
    page := addr+64-eeAddr//64<#endaddr+1         ' Find next EEPROM page boundary
    SetAddr(eeAddr)                               ' Give EEPROM starting address
    repeat                                        ' Bytes -> EEPROM until page boundary
      SendByte(byte[addr++])
    until addr == page
    i2cstop                                       ' From 24LC256's page buffer -> EEPROM
    eeaddr := addr - startAddr + eeStart          ' Next EEPROM starting address
  until addr > endAddr                            ' Quit when RAM index > end address

PUB ToRam(startAddr, endAddr, eeStart) | addr

  ''Copy from EEPROM beginning at eeStart address to startAddr..endAddr in main RAM.

  SetAddr(eeStart)                                ' Set EEPROM's address pointer
  i2cstart
  SendByte(%10100001)                             ' EEPROM I2C address + read operation
  if startAddr == endAddr
    addr := startAddr
  else
    repeat addr from startAddr to endAddr - 1       ' Main RAM index startAddr to endAddr
      byte[addr] := GetByte                         ' GetByte byte from EEPROM & copy to RAM
      SendAck(ACK)                                  ' Acknowledge byte received
  byte[addr] := GetByte                             ' GetByte byte from EEPROM & copy to RAM
  SendAck(NACK)
  i2cstop                                         ' Stop sequential read

PRI SetAddr(addr)

  'Sets EEPROM internal address pointer.

  poll                                            ' Poll until EEPROM acknowledges
  SendByte(addr >> 8)                             ' Send address high byte
  SendByte(addr)                                  ' Send address low byte
```

```
PRI Poll | ackbit

  ' Poll until acknowledge.  This is especially important if the 24LC256 is copying from
  ' buffer to EEPROM.

  ackbit~~                                      ' Make acknowledge 1
  repeat                                        ' Send/check acknowledge loop
    i2cstart                                    ' Send I2C start condition
    ackbit := SendByte(%10100000)               ' Write command with EEPROM's address
  while ackbit                                  ' Repeat while acknowledge is not 0

PRI I2cStart

  ' I2C start condition.  SDA transitions from high to low while the clock is high.
  ' SCL does not have the pullup resistor called for in the I2C protocol, so it has to be
  ' set high. (It can't just be set to inSendByte because the resistor won't pull it up.)

  outa[SCL]~~                                   ' SCL pin outSendByte-high
  dira[SCL]~~
  dira[SDA]~                                    ' Let pulled up SDA pin go high
  outa[SDA]~                                    ' Transition SDA pin low
  dira[SDA]~~                                   ' SDA -> outSendByte for SendByte method

PRI SendByte(b) : ackbit | i

  ' Shift a byte to EEPROM msb first.  Return if EEPROM acknowledged.  Returns
  ' acknowledge bit.  0 = ACK, 1 = NACK.

  b ><= 8                                       ' Reverse bits for shifting msb right
  outa[SCL]~                                    ' SCL low, SDA can change
  repeat 8                                      ' 8 reps sends 8 bits
    outa[SDA] := b                              ' Lowest bit sets state of SDA
    outa[SCL]~~                                 ' Pulse the SCL line
    outa[SCL]~
    b >>= 1                                     ' Shift b right for next bit
  ackbit := GetAck                              ' Call GetByteAck and return EEPROM's Ack

PRI GetAck : ackbit

  ' GetByte and return acknowledge bit transmitted by EEPROM after it receives a byte.
  ' 0 = ACK, 1 = NACK.

  dira[SDA]~                                    ' SDA -> SendByte so 24LC256 controls
  outa[SCL]~~                                   ' Start a pulse on SCL
  ackbit := ina[SDA]                            ' GetByte the SDA state from 24LC256
  outa[SCL]~                                    ' Finish SCL pulse
  outa[SDA]~                                    ' SDA will hold low
  dira[SDA]~~                                   ' SDA -> outSendByte, master controls

PRI I2cStop

  ' Send I2C stop condition.  SCL must be high as SDA transitions from low to high.
  ' See note in i2cStart about SCL line.

  outa[SDA]~                                    ' SDA -> outSendByte low
  dira[SDA]~~
  outa[SCL]~~                                   ' SCL -> high
  dira[SDA]~                                    ' SDA -> inSendByte GetBytes pulled up

PRI GetByte : value

  ' Shift in a byte msb first.

  value~                                        ' Clear value
  dira[SDA]~                                    ' SDA input so 24LC256 can control
  repeat 8                                      ' Repeat shift in eight times
    outa[SCL]~~                                 ' Start an SCL pulse
    value <<= 1                                 ' Shift the value left
    value += ina[SDA]                           ' Add the next most significant bit
    outa[SCL]~                                  ' Finish the SCL pulse

PRI SendAck(ackbit)

  ' Transmit an acknowledgement bit (ackbit).

  outa[SDA]:=ackbit                             ' Set SDA output state to ackbit
  dira[SDA]~~                                   ' Make sure SDA is an output
  outa[SCL]~~                                   ' Send a pulse on SCL
```

```
  outa[SCL]~
  dira[SDA]~                                    ' Let go of SDA
```