Thumper

Internet Radio Stream Player with MP3 Player, Recorder, and Web Interface

Harrison Pham
harrison@harrisonpham.com

February 26, 2010
Parallax 2009/2010 Propeller Design Contest
Project PC091923

Table of Contents

1.0 Project Number

PC091923

2.0 Project Description

Thumper is a Propeller based Internet Radio Player with MP3 Recording and Playback capabilities. The hardware contains a single Parallax Propeller Chip and some external support chips to implement a complete internet radio player. Thumper contains features not found in any commercially available products and can be built for only a fraction of the cost of commercial products.

Thumper's long feature set beats out many commercial internet radio devices, many of which cost hundreds of dollars. The feature list includes many items not available in commercial internet radio devices. One such feature is the real-time MP3 recording capabilities. The user can easily record any music stream to standard MP3 files. Thumper also supports a full featured web interface which allows users to remotely control the music playing functionality from any device with a standard web browser. Song metadata such as the Title and Artist information is fully supported and displayed. Thumper also supports standard ID3v1 MP3 tags for easy MP3 library sharing. Files saved with Thumper can be played with any device or program that supports MP3 files (such as iPods, Windows Media Player, Winamp, etc).

The real advantage of Thumper is the extremely fault tolerant internet radio streaming functionality. The TCP/IP stack, which provides the network protocol used to stream the music, was custom designed and tweaked for this project. It features lost packet recovery and fast connection recovery in order to reduce music streaming stuttering. Commercial products and programs (such as Winamp) have problems recovering from network problems. Thumper can resume music streams within seconds after an error occurs.

The hardware for Thumper is custom designed to be as small as possible. The PCB size is 3.2" by 1.5", which fits perfectly behind a standard 2 line by 16 character LCD display. The board stacks behind the display, forming a very elegant and small package. All external connections are specially positioned for easy accessibility. The only connections needed are power, ethernet, and audio out. User input can be performed via a standard Sony compatible universal infrared remote or via any standards complaint web browser.

This project was only possible thanks to the Propeller's powerful multiple core architecture. Conventional microcontrollers do not have the required computational power to perform the multiple tasks required to implement Thumper's feature set. Even software solutions for desktop computers do not have many of the features included in Thumper.

2.1 Features

The feature list for Thumper is shown in Table 1.

Table 1 - Feature List

Playback
- MP3 Shoutcast Streams
- AAC Shoutcast Streams
- MP3 Files on SD Card

Recording
- MP3 to SD Card
- Simultaneous Listen / Record

Supported Audio Formats
- MP3 up to 320Kbps
- Streaming AAC/MP4

User Interface
- Sony Compatible IR Remote
- Elegant 2x16 LCD
- Powerful Web Interface

Compatibility Features
- ID3v1 MP3 Metadata Tags (Read / Write)
- Standard MP3 Files
- Standard FAT16/32 File System

Network Connectivity
- 10Base-T Ethernet
- HTTP Client / Webserver
- Time Synchronization w/ NIST Time Std
- Static IP Configuration
- Low Latency TCP/IP Stack

Processor and Memory
- Propeller P8X32A @ 100MHz
- Dedicated MP3 Decoder Codec
- 64KB EEPROM for Firmware / Settings
- 64KB SRAM for Stream Buffering
- 8KB Ethernet Buffer
- SDHC microSD Card Support (up to 4GB)

Web Interface
- Full Remote Control
- Auto Updating Status Information
- Compatible with all Major Browsers
- Flexible HTML / Javascript Based UI

3.0 Schematic

The schematic for the hardware is shown in Figures 1 and 2. The full sized schematic sheets are provided in Appendix B of this report and as an external PDF file. The hardware consists of six main chips:

- 1 x Parallax Propeller
- 1 x ENC28J60 Ethernet MAC/PHY
- 1 x VS1053b MP3/AAC Decoder
- 1 x 24C512 I2C EEPROM
- 2 x 23K256 32KB SRAMs

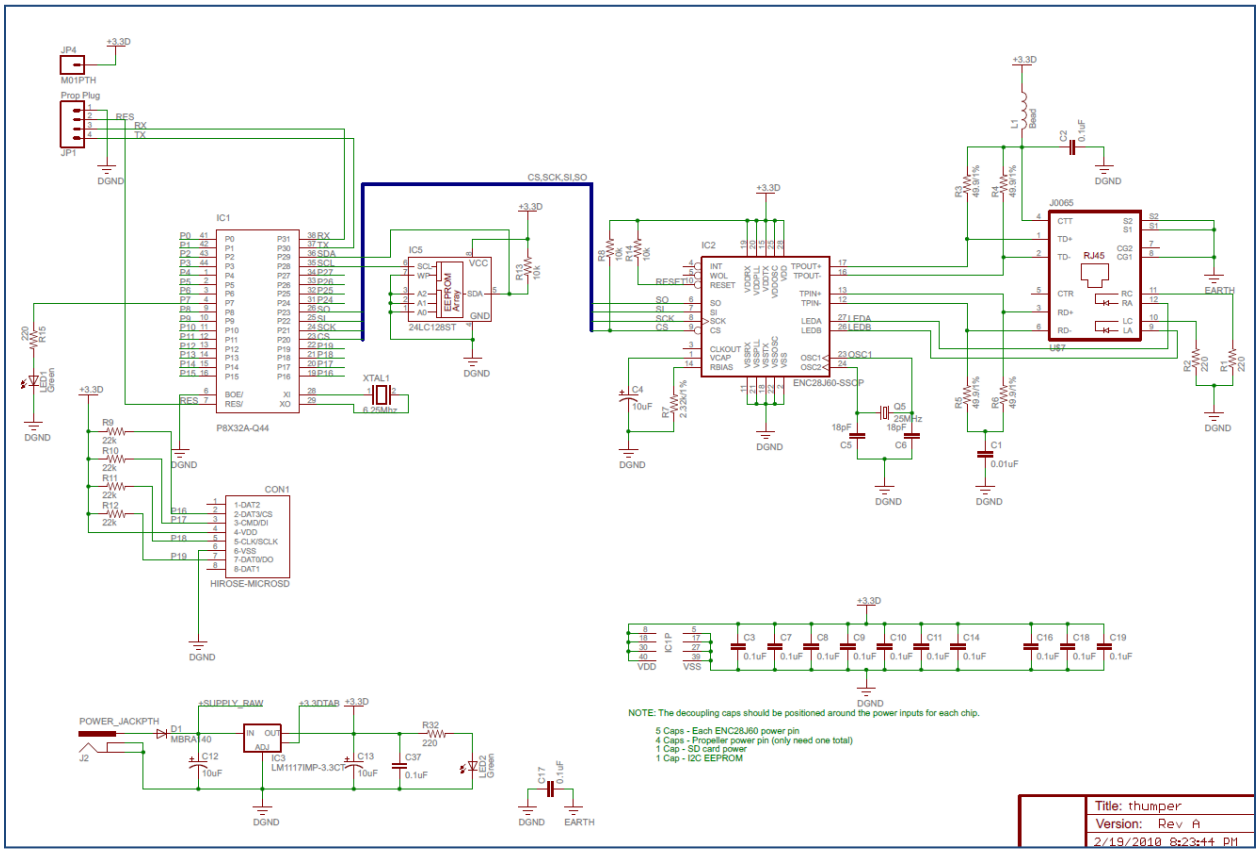

Figure 1 - Propeller and Ethernet Interface

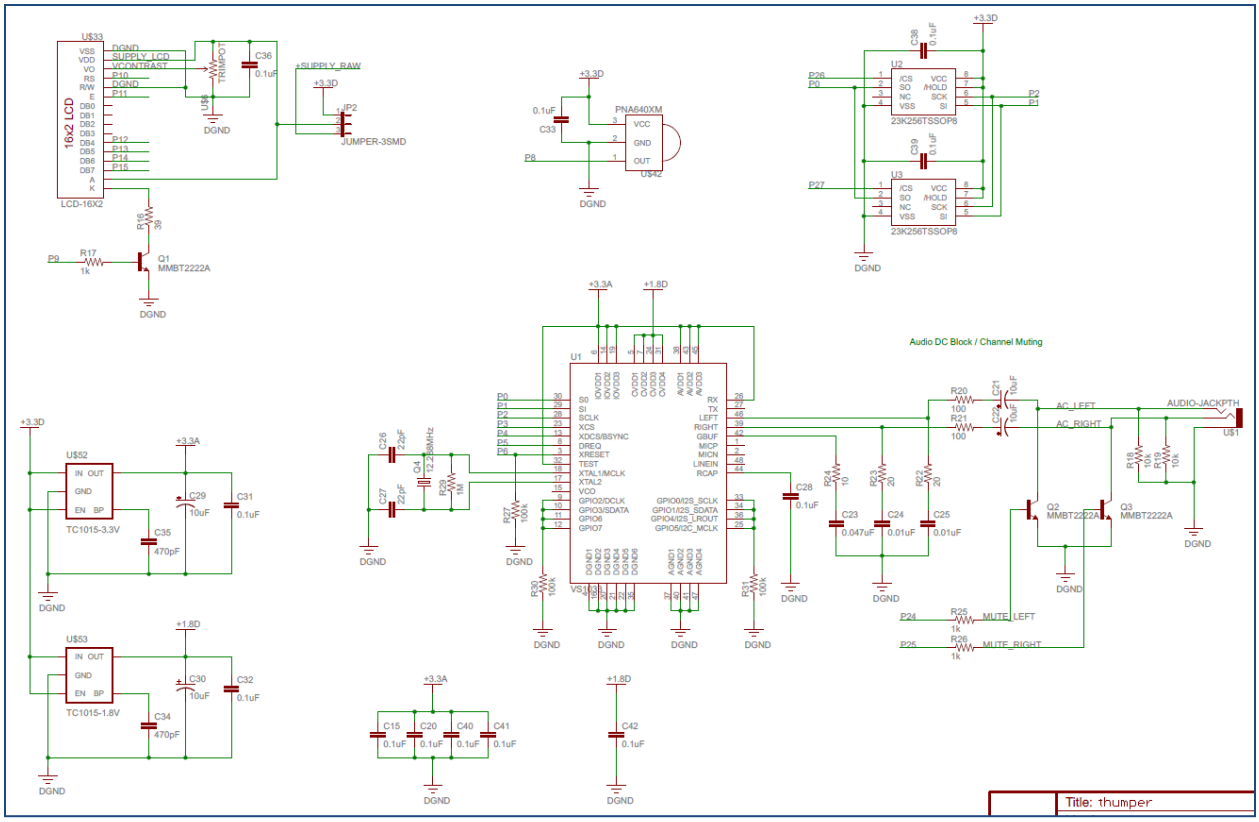

Figure 2 - MP3 Decoder, LCD Display, Static RAM, and IR Receiver

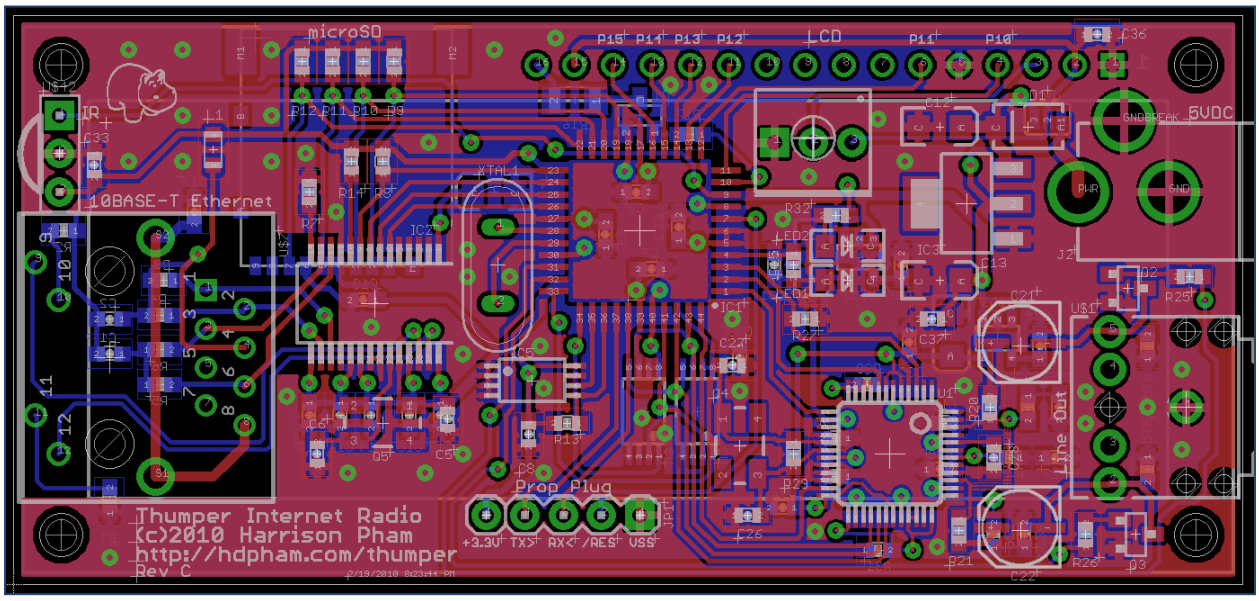

Figure 3 - PCB Layout

4.0 Block Diagram

The block diagram for the Thumper design is shown in Figure 4. The design consists of a few major parts, each of which is explained in the following sections.

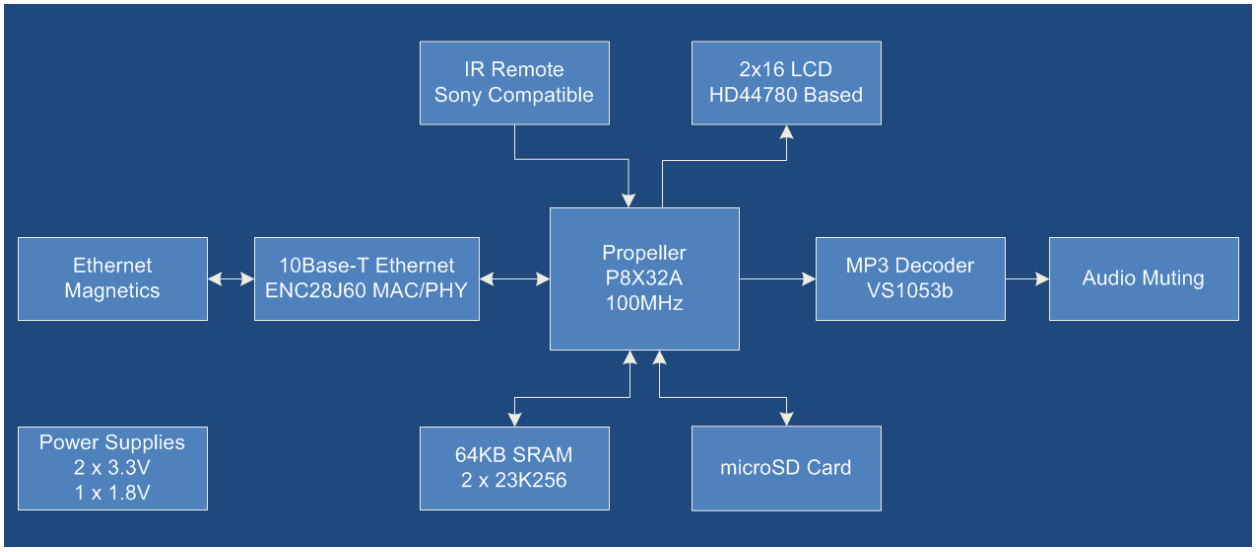

Figure 4 - Hardware Block Diagram

4.1 Propeller

A Propeller is the heart of the internet radio design. The Propeller's eight powerful and fast processors each handle a dedicated task that wouldn't be possible using common microcontrollers like Atmel AVRs and Microchip PICs. The Propeller runs a software TCP/IP stack, FAT16/FAT32 file system, various communication bus drivers, and user interface logic.

4.2 Audio Decoder / Codec

The compressed audio streams are decoded by a dedicated VS1053b MP3 / AAC decoder chip. This chip is utilized in many commercial music players and frees the main controller, a Propeller in this case, from having to dedicate cycles to audio stream decoding. The communication from the Propeller to the VS1053 is performed via a low speed (2.5MHz) SPI bus that is also shared with the SRAM buffer chips.

4.3 10Base-T Ethernet

The network connection is provided by a ENC28J60 10mbps ethernet MAC/PHY chip. The chip communicates with the Propeller over a high speed 25MHz SPI bus. The chip does not have a hardwired TCP/IP stack like other chips, which means the Propeller has to handle all the complex TCP/IP stack operations in software. This is actually a benefit because it allows fine tweaking of various TCP connection parameters, which allows for extremely low latency and fault tolerant communications. The TCP/IP stack used was custom written and tweaked specifically for the high throughput and low latency required for streaming audio applications.

4.4 SD Card / File System

The FAT16/FAT32 file system is stored on an external microSD card. The file system allows users to easily play and save MP3 files. The SD card also stores the system configuration parameters in an easy to modify format (users can edit these parameters with Notepad or any other text editor). SD cards up to 4GB are supported, which provides for many hours of high quality music storage.

4.5 LCD

An elegant white-on-black 2 line by 16 character LCD was chosen for this project. The LCD provides a very professional and high contrast display that blends in well with any home entertainment system. It works especially well in the dark and has a software controllable backlight.

The hardware was also designed to support a standard NTSC video output. This was done by bringing pins P10 - P15 to an external header (which also happens to be the same pins used by the character LCD).

4.6 IR Receiver

Thumper accepts user input from a wireless infrared remote control. The software running on the Propeller supports the standard Sony IR protocol, which allows the user to use cheap and readily available remote controls.

4.7 Static RAM

A total of 64KBytes of SRAM is used to buffer the audio stream. The SRAM chips used are the 23K256 32KByte SRAM chips made by Microchip. These devices are interfaced over SPI and share a SPI bus with the VS1053b audio decoder chip. The Propeller uses the two SRAM chips as an external FIFO queue. The SPI bus is clocked at 25MHz during SRAM communication in order to provide the highest data bandwidth as possible.

5.0 Source Code

The source code for this project is provided in Appendix A of this document. The next sections explain the major portions of the source code.

5.1 Main Object (thumper-main.spin)

All the main application logic for Thumper are located in *thumper-main.spin*. The main job of this object is to handle the high level user interface functions (the LCD screen, the webserver, and the IR remote control). The object also handles the stream / music playing functionality by routing the audio data between the child objects.

5.2 TCP/IP Stack (driver-socket.spin)

The most complex code for this project is housed in the TCP/IP stack object. The TCP/IP stack implements TCP client and server functionality. The stack is designed for low latency and high reliability which is required for playing the high bitrate audio streams.

5.3 MP3 Decoder (vs10xx_mp3.spin)

The MP3 Decoder object communicates to the VS1053b Decoder/Codec chip and the two 32KByte Static RAM chips over a single 20MHz SPI bus. The object provides a clean API to the application code by internally handling all the stream buffering operations. The object also provides various methods for changing the bass and volume.

5.4 SD Card Driver (fsrw26.spin)

The *fsrw26.spin* object provides the FAT16/FAT32 file system and SD card support. This object provides a very clean and streamlined API for opening, reading, and writing files on the onboard microSD card. This code was obtained from the OBEX and was not written by the author of this project.

5.5 Software RTC (softrtc.spin)

The Software RTC object provides a software Real Time Clock which is used to display the current time and date. This information is also used for time stamping the files on the SD card. The object also supports internet time synchronization with the NIST (National Institute of Standards and Technology) atomic clock. The synchronization functionality is completely automatic and requires no user intervention (except to set the time zone of course).

6.0 Bill of Materials

The project was built using almost all SMT (surface mount) devices, which requires the use of a custom PCB. Users wishing to build their own will probably want to have a custom PCB manufactured. The alternative is to build the device using SMT breakout boards and through hole components.

The bill of materials is provided below for those interested in building this project. The project costs around $75 to build assuming that you can have the board manufactured for $25 or less. Almost all components can be purchased from Digikey and Mouser. The audio decoder chip can be purchased from Sparkfun or directly from VLSI Solution.

Qty	Value	Device	Parts
3	0.01uF	CAP0603-CAP	C1, C24, C25
25	0.1uF	CAP0603-CAP	C2, C3, C7, C8, C9, C10, C11, C14, C15, C16, C17, C18, C19, C20, C28, C31, C32, C33, C36, C37, C38, C39, C40, C41, C42
2	10uF	CAP_POLC	C21, C22
1	0.047uF	CAP0603-CAP	C23
2	22pF	CAP0603-CAP	C26, C27
2	470pF	CAP0603-CAP	C34, C35
5	10uF	CAP_POL1206	C4, C12, C13, C29, C30
2	18pF	CAP0603-CAP	C5, C6
1	HIROSE-MICROSD	HIROSE-MICROSD	CON1
1	MBRA140	DIODESMA	D1
1	P8X32A-Q44	P8X32A-Q44	IC1
1	ENC28J60-SSOP	ENC28J60-SSOP	IC2
1	LM1117IMP-3.3CT	V_REG_LM1117SOT223	IC3
1	24LC128ST	24LC128ST	IC5
1	POWER_JACKPTH	POWER_JACKPTH	J2
1	Prop Plug	M04PTH	JP1
1	JUMPER-3SMD	JUMPER-3SMD	JP2
1	M01PTH	M01PTH	JP4
1	Bead	L-USL2012C	L1
2	Green	LED1206	LED1, LED2
3	MMBT2222A	TRANSISTOR_NPNSOT23	Q1, Q2, Q3
1	12.288MHz	CRYSTAL5X3	Q4
1	25MHz	CRYSTAL5X3	Q5
4	220	RESISTOR0603-RES	R1, R2, R15, R32
1	39	RESISTOR1206	R16
3	1k	RESISTOR0603-RES	R17, R25, R26
2	100	RESISTOR0603-RES	R20, R21
2	20	RESISTOR0603-RES	R22, R23
1	10	RESISTOR0603-RES	R24
3	100k	RESISTOR0603-RES	R27, R30, R31
1	1M	RESISTOR0603-RES	R29
4	49.9/1%	RESISTOR0603-RES	R3, R4, R5, R6
1	2.32k/1%	RESISTOR0603-RES	R7

5	10k	RESISTOR0603-RES	R8, R13, R14, R18, R19
4	22k	RESISTOR0603-RES	R9, R10, R11, R12
1	AUDIO-JACKPTH	AUDIO-JACKPTH	U$1
1	LCD-16X2	LCD-16X2	U$33
1	PNA640XM	PNA640XM	U$42
1	TC1015-3.3V	V_REG_LDOSMD	U$52
1	TC1015-1.8V	V_REG_LDOSMD	U$53
1	TRIMPOT	TRIMPOT	U$6
1	J0065	J0065	U$7
1	VS1033	VS1033	U1
2	23K256TSSOP8	23K256TSSOP8	U2, U3
1	6.25Mhz	XTAL/S	XTAL1

7.0 Pictures

This section contains various pictures of the project. Full size high resolution pictures are also included in the project submission zip file.

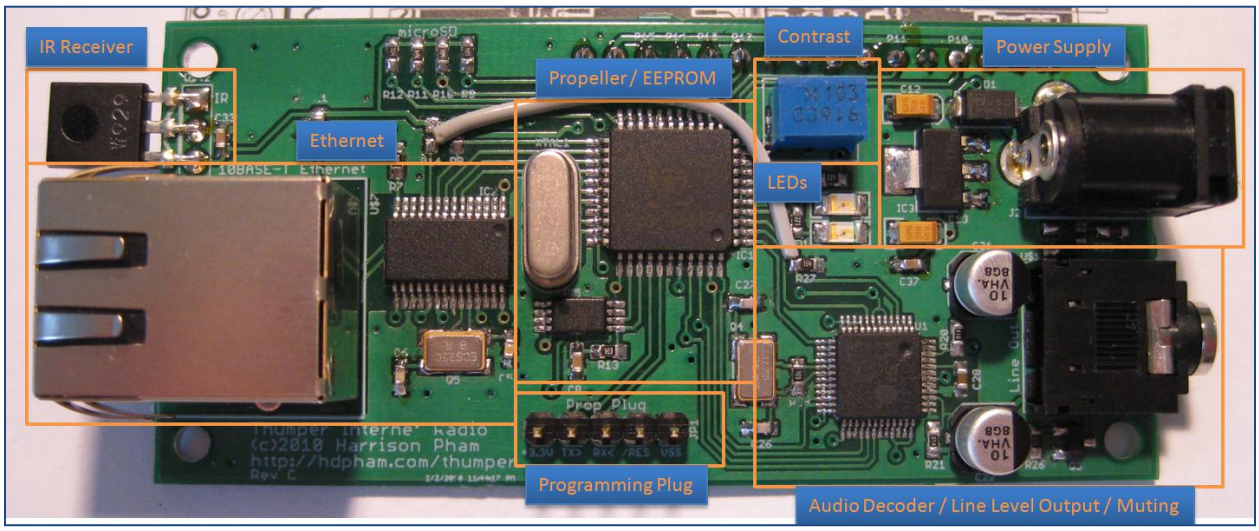

Figure 5 - PCB Top Side

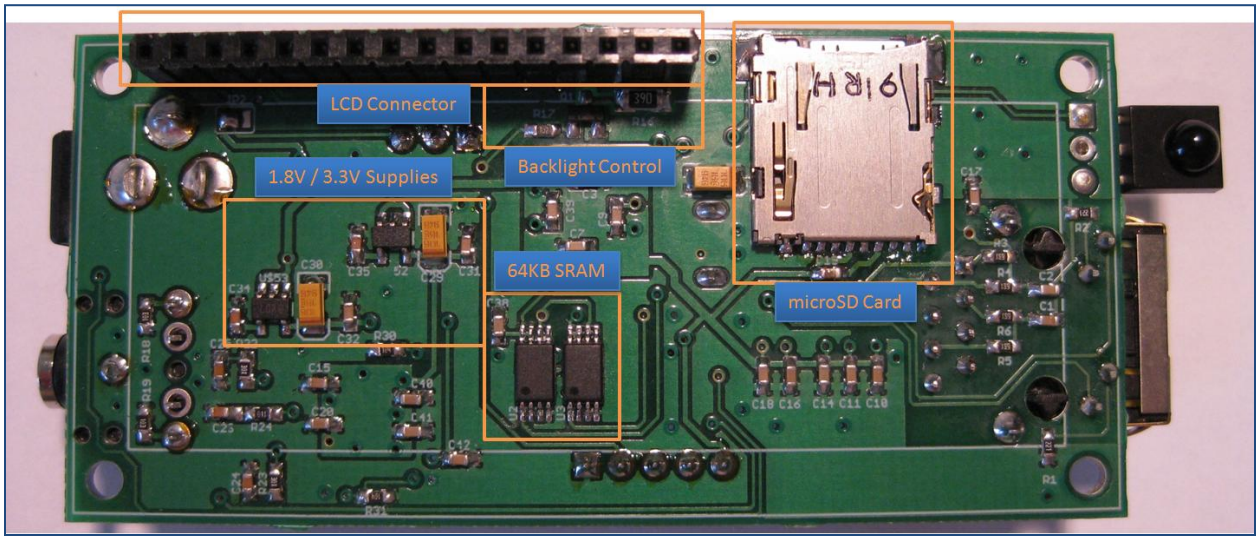

Figure 6 - PCB Bottom Side

Figure 7 - Complete Package w/ SD Card Reader and Remote

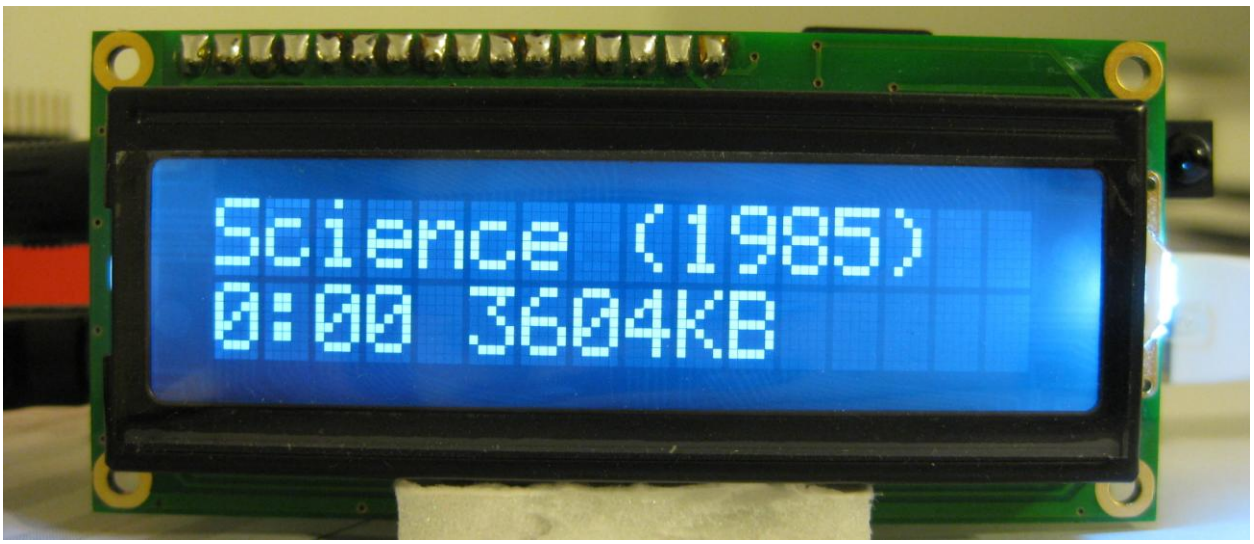

Figure 8 - SD Card MP3 Playback Display

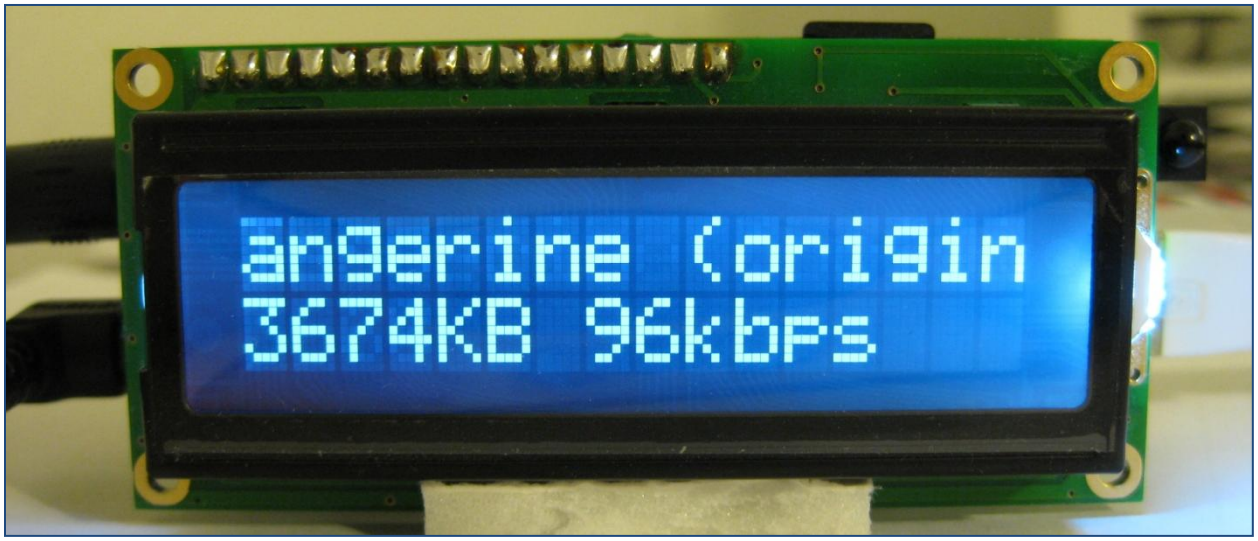

Figure 9 - Internet Streaming Display

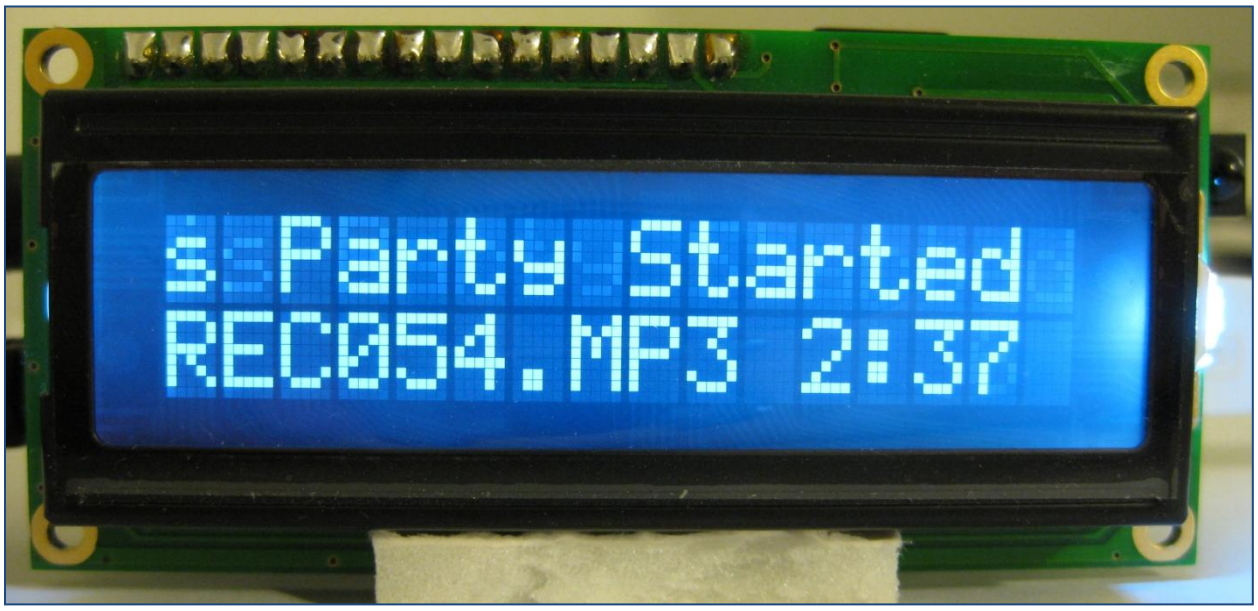

Figure 10 - Stream Record Mode

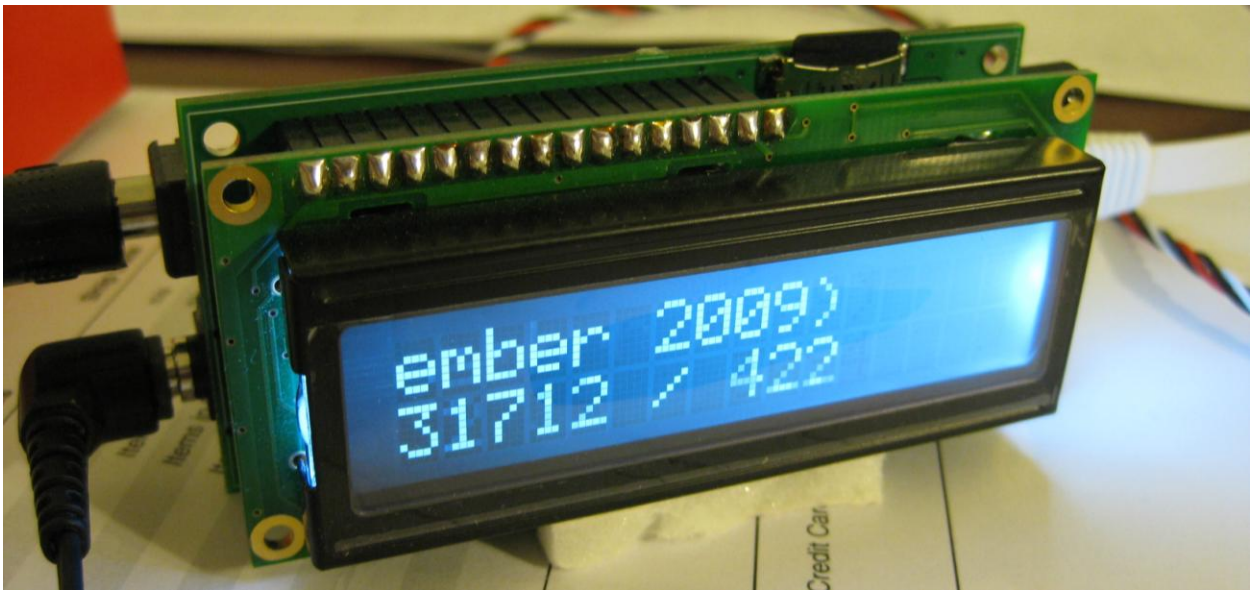

Figure 11 - Angled View

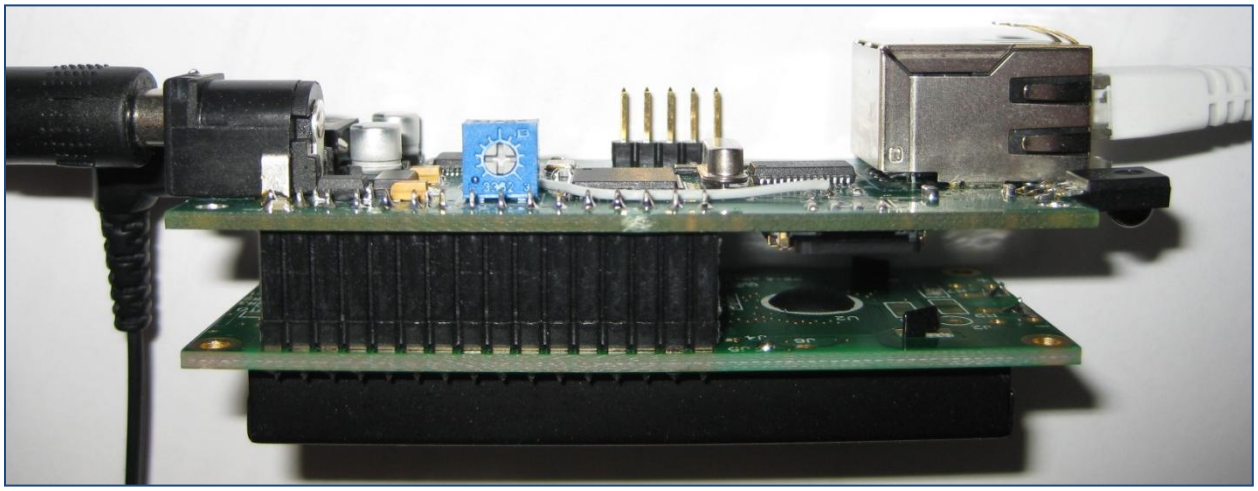

Figure 12 - PCB Stack View

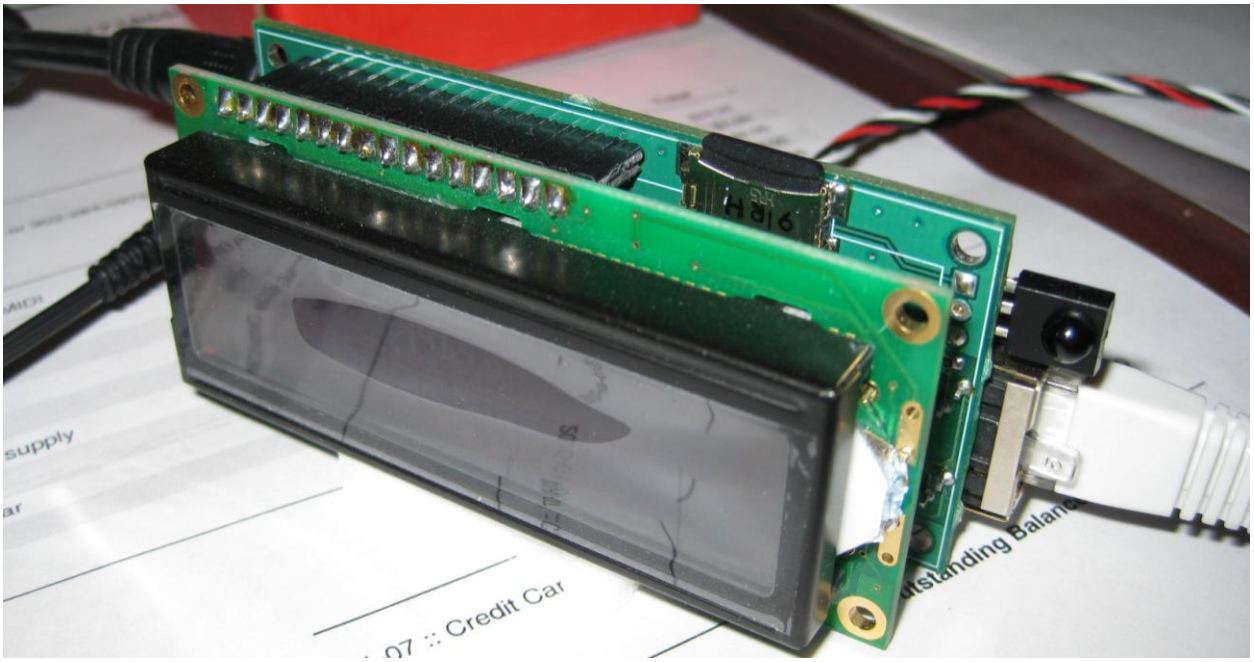

Figure 13 - Angled View

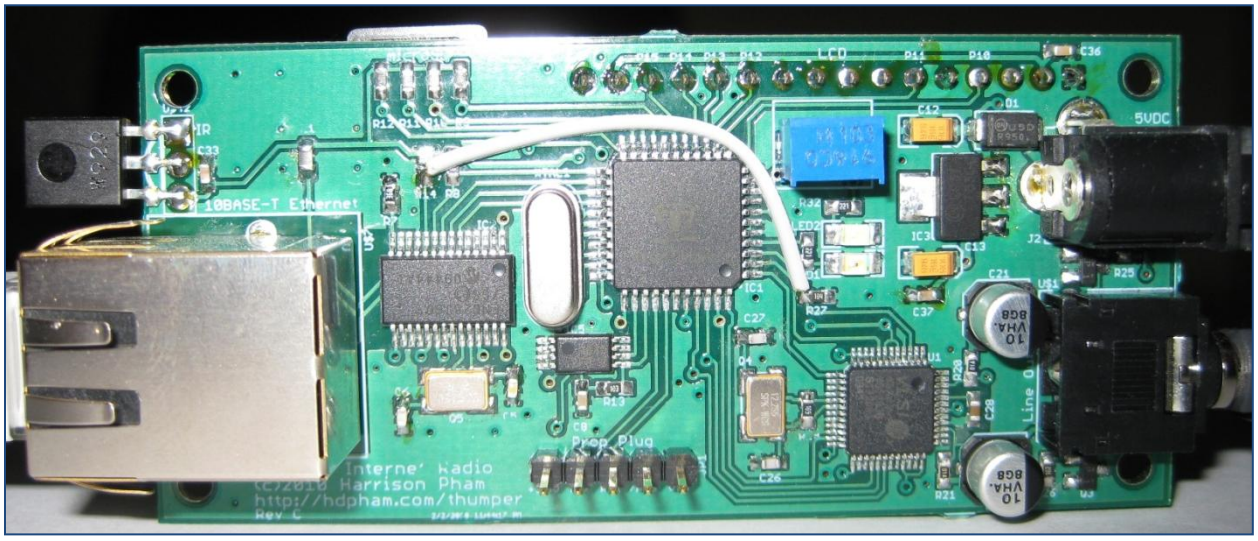

Figure 14 - Assembled Back View

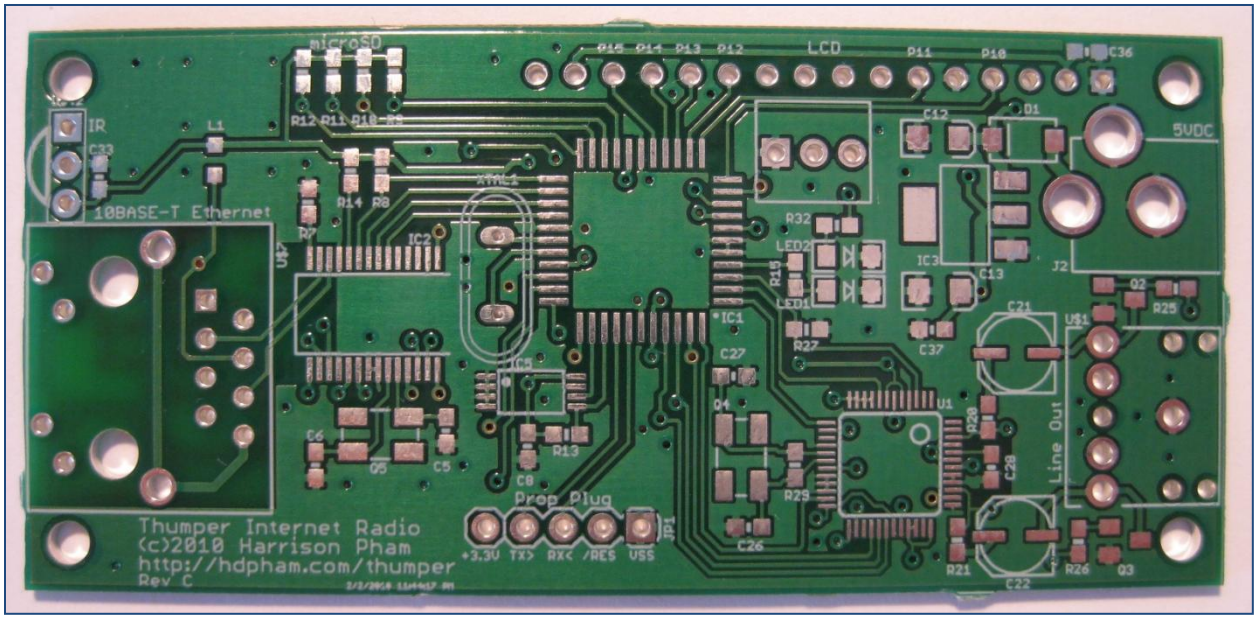

Figure 15 - Unpopulated PCB Top View

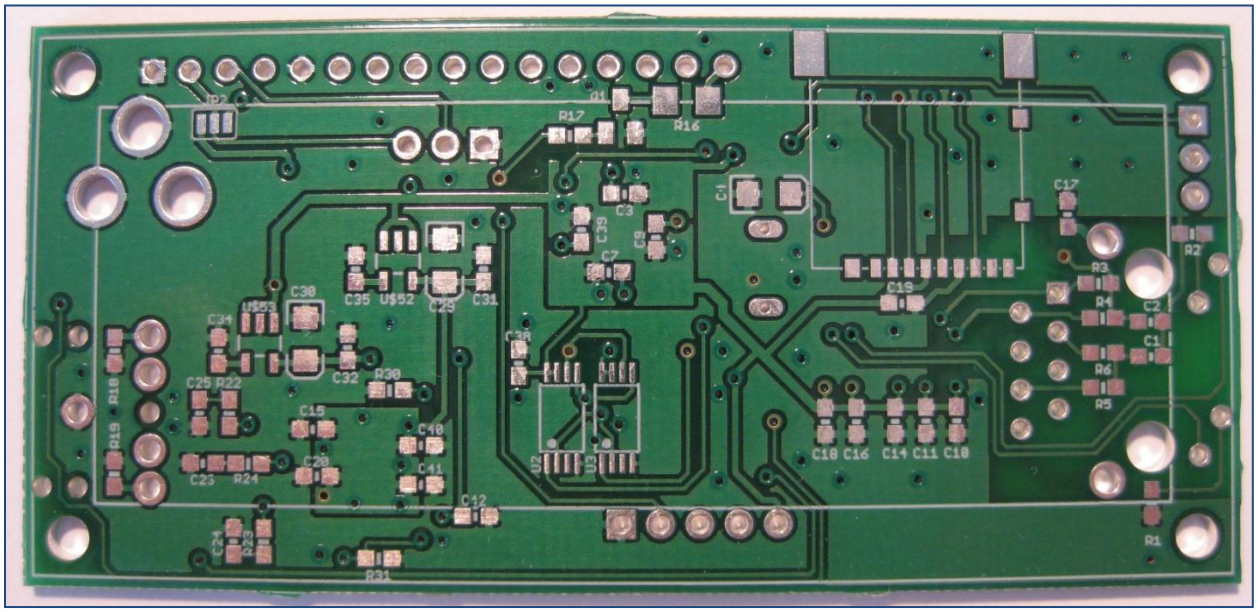

Figure 16 - Unpopulated PCB Bottom View

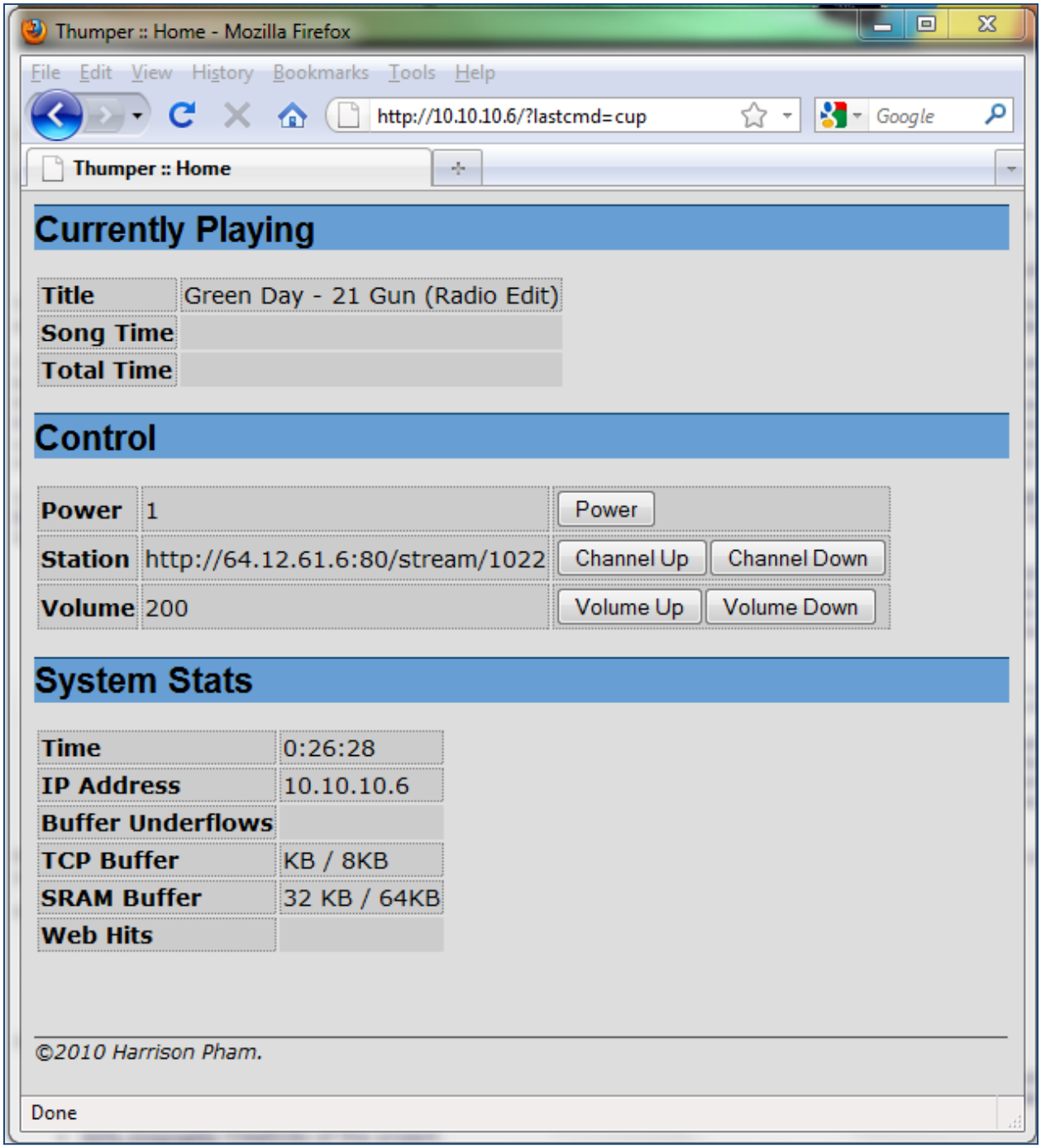

Figure 17 - Web Interface

8.0 Licensing

9.0 Source Code

The following sections contain the source code used in the project. Default objects shipped with the official Parallax Propeller Tool and those found on the OBEX are not included for clarity.

9.1 thumper-main.spin

```
{{
  Internet Radio
  --------------

  Copyright (C) 2009 Harrison Pham <harrison@harrisonpham.com>

  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/>.
}}

' Shoutcast Metadata Description: http://www.smackfu.com/stuff/programming/shoutcast.html

CON
  _clkmode = xtal1+pll16x
  _xinfreq = 6_250_000

OBJ
  sock  : "api_telnet_serial"
  mp3   : "vs10xx_mp3"
  'vga   : "vga_text"
  lcd   : "driver_hd44780"
  str   : "util_strings"
  num   : "Numbers"
  rtc   : "softrtc"
  ir    : "IR_Remote"
  sd    : "fsrw26"
  web   : "api_telnet_serial"
  dt    : "date_time_epoch"

CON

  SPI_RESET     = 6

  NIC_CS        = 20
  NIC_SCK       = 21
  NIC_SI        = 22
  NIC_SO        = 23

  MP3_MISO      = 0
  MP3_MOSI      = 1
  MP3_CLK       = 2
  MP3_CS        = 3
  MP3_DCS       = 4
  MP3_DREQ      = 5

  AUDIO_MUTE_L  = 24
  AUDIO_MUTE_R  = 25

  SRAM1_CS      = 26
  SRAM2_CS      = 27

  SD_DO         = 19
  SD_CLK        = 18
```

```
  SD_DI         = 17
  SD_CS         = 16

  LED_STATUS    = 7

  LCD_BASE      = 10

  IR_DET        = 8

  BACKLIGHT     = 9

  CHUNKLEN      = 256

  MAXSTATIONS   = 16

DAT
  mac_addr      byte    $02, $00, $00, $00, $00, $03

                long
  ip_addr       byte    10, 10, 0, 9                      ' device's ip address
  ip_subnet     byte    255, 255, 255, 0                  ' network subnet
  ip_gateway    byte    10, 10, 0, 254                    ' network gateway (router)
  ip_dns        byte    10, 10, 0, 254                    ' network dns

DAT

  {            long
  station1    byte      205, 188, 234, 5
              word      80
              byte      "/stream/1003",0
              long
  station2    byte      64, 237, 49, 76
              word      8015
              byte      "/",0
              long
  station3    byte      207, 200, 96, 135
              word      80
              byte      "/stream/1024",0
              long
  station4    byte      207, 200, 96, 137
              word      80
              byte      "/stream/1013",0
              long
  station5    byte      64, 12, 61, 6
              word      80
              byte      "/stream/1022",0
              long
  station6    byte      209, 51, 161, 54
              word      8072
              byte      "/",0

  stations    long      @station1, @station2, @station3, @station4, @station5, @station6
  endstations

  numstations byte      (@endstations - @stations) / 4}

  ' station structure
  ' 0 = IP    4 bytes
  ' 4 = port  2 bytes
  ' 6 = uri   58 bytes
  ' 64 total bytes per entry
  stations    byte      0[MAXSTATIONS * 64]
  numstations byte      0

CON

  #0,PLAY_INTERNET,PLAY_SD

VAR
  byte mp3buff[CHUNKLEN]
  byte strTemp[128]
  byte strMeta[128]

  byte strTitle[31]
  byte strArtist[31]

  word connretries
```

```
  long stationidx

  byte isOn
  byte playMode

  long volume
  long balance
  long bassboost

  byte webControl

  long timeOffset

VAR
  byte tcp_mp3rxbuff[4096]
  byte tcp_mp3txbuff[64]
  byte tcp_webrxbuff[128]
  byte tcp_webtxbuff[128]

  byte webbuff[128]
  long webstack[64]

  long stationip, stationport, stationuri

PUB main | ptr

  ' turn on LCD backlight
  dira[BACKLIGHT]~~
  outa[BACKLIGHT]~~
  lcd.start(LCD_BASE, 16, 2)

  ' mute left and right channels to discharge output caps
  outa[AUDIO_MUTE_L..AUDIO_MUTE_R]~~
  dira[AUDIO_MUTE_L..AUDIO_MUTE_R]~~

  ' reset chips
  dira[SPI_RESET]~~             ' output
  outa[SPI_RESET]~              ' low
  delay_ms(10)
  outa[SPI_RESET]~~             ' high
  delay_ms(10)

  if \sd.mount_explicit(SD_DO, SD_CLK, SD_DI, SD_CS) < 0
    lcd.str(string("SD Error"))
    repeat
      waitcnt(0)

  if \loadConfig < 0
    lcd.str(string("Config Error"))
    repeat
      waitcnt(0)

  sock.start(NIC_CS, NIC_SCK, NIC_SI, NIC_SO, -1, @mac_addr, @ip_addr)

  ir.start(IR_DET)

  dira[MP3_DREQ]~
  dira[LED_STATUS]~~

  volume := 200
  balance := 0
  bassboost := 0

  initMP3Decoder

  delay_ms(250)

  ' unmute left and right channels
  dira[AUDIO_MUTE_L..AUDIO_MUTE_R]~

  rtc.start(-1, rtc#MODE_COG)
  rtc.update

  connretries := 0

  stationidx := 0

  isOn := false
```

```
  playMode := PLAY_INTERNET

  {mp3.SRAMWriteByte(0, $FE)
  mp3.SRAMWriteByte(3468, $AD)
  mp3.SRAMWriteBYte(324, $C0)

  if mp3.SRAMReadByte(0) <> $FE or mp3.SRAMReadByte(3468) <> $AD or mp3.SRAMReadByte(324) <> $C0
    repeat
      waitcnt(0)}

  cognew(webserver, @webstack)

  repeat
    ' 205.188.234.1/stream/1003 - di.fm Techno Trance
    ' 64.12.61.4/stream/1040 - .977 80s
    ' 64.237.49.76:8040/ - .977 Alternative
    'if \connect(constant((205<<24) + (188<<16) + (234<<8) + 5), 80, string("/stream/1003")) < 0
    'if \connect(constant((64<<24) + (237<<16) + (49<<8) + 76), 8040, string("/")) < 0
    'if \connect(constant((10<<24) + (10<<16) + (10<<8) + 115), 8080, string("/")) < 0

    lcd.start(LCD_BASE, 16, 2)

    mp3.SetVolume(1, 0)
    mp3.DMASet(0)
    mp3.SRAMEmpty
    mp3.SendZeros

    repeat
      lcd.pos(1, 1)
      lcd.str(string("  Thumper v2.0"))
      lcd.pos(2, 1)
      if playMode == PLAY_INTERNET
        lcd.str(string(" Internet Radio"))
      elseif playMode == PLAY_SD
        lcd.str(string("   MP3 Player"))
      lcd.clearRestOfLine
      irHandle
    while not isOn

    lcd.cls

    ptr := @stations + (stationidx * 64)
    stationip := conv_endianlong(long[ptr])
    stationport := word[ptr + 4]
    stationuri := ptr + 6

    if playMode == PLAY_INTERNET
      if \connect(stationip, stationport, stationuri) < 0
        ' socket exception ocurred, so close it
        \sock.close
        connretries++
    elseif playMode == PLAY_SD
      \mp3Player

    \sd.pclose

    delay_ms(250)

PRI webserver

  webControl := ir#NoNewCode

  \web.listen(80, @tcp_webrxbuff, 128, @tcp_webtxbuff, 128)
  repeat
    \web.relisten
    '\web.resetBuffers
    if \web.isConnected
      if \_webThread == 0
        \web.txflush
      \web.close

PRI _webthread | i, j, k, args, redirect, argcopy[2]

  if _webReadLine == 0
    return 0

  redirect := false
  ' obtain get arguments
```

```
if (i := str.indexOf(@webbuff, string("r.cgi?"))) <> -1
  redirect := true
  args := @webbuff[i + 6]
  if (j := str.indexOf(args, string("="))) <> -1
    byte[args][j] := 0

if strcomp(args, string("pwr"))
  webControl := ir#power
elseif strcomp(args, string("cup"))
  webControl := ir#chUp
elseif strcomp(args, string("cdn"))
  webControl := ir#chDn
elseif strcomp(args, string("vup"))
  webControl := ir#volUp
elseif strcomp(args, string("vdn"))
  webControl := ir#volDn

argcopy[0] := argcopy[1] := 0
bytemove(@argcopy, args, 7)

' read the rest of the headers
repeat until _webReadLine == 0

if redirect
  web.str(string("HTTP/1.0 302 Found",13,10,"Location: /?lastcmd="))
  web.str(@argcopy)
  web.str(string(13,10,13,10))
  return 0

'web.str(string("HTTP/1.0 200 OK",13,10,13,10))

i := @index_htm
repeat while i < @index_htm_end
  if byte[i] == "~"
    i++
    j := num.FromStr(i, num#DEC)
    case j
      1  : web.str(@strMeta)                          ' title
      2  : ' unused
      3  : ' song time
      4  : ' total time
      5  : web.dec(isOn & 1)                          ' power
      6  : web.dec(stationip.byte[3])                 ' station
           web.tx(".")
           web.dec(stationip.byte[2])
           web.tx(".")
           web.dec(stationip.byte[1])
           web.tx(".")
           web.dec(stationip.byte[0])
           web.tx(":")
           web.dec(stationport)
           web.str(stationuri)
      7  : web.dec(volume)                            ' volume
      8  : k := dt.timeETV(rtc.getTimestamp + timeOffset)   ' time
           web.dec(k.byte[2])
           web.tx(":")
           web.dec(k.byte[1] / 10)
           web.dec(k.byte[1] // 10)
           web.tx(":")
           web.dec(k.byte[0] / 10)
           web.dec(k.byte[0] // 10)
      9  : web.dec(ip_addr.byte[0])                   ' ip
           web.tx(".")
           web.dec(ip_addr.byte[1])
           web.tx(".")
           web.dec(ip_addr.byte[2])
           web.tx(".")
           web.dec(ip_addr.byte[3])
      10 : ' underflow
      11 : ' tcp buf
      12 : web.dec(mp3.SRAMBytes / 1000)              ' sram buf
      13 : ' web hits
    i++
  else
    web.tx(byte[i])
  i++

return 0
```

```
PRI _webReadLine | i, ch
  repeat i from 0 to 126
    ch := web.rxtime(500)
    if ch == 13
      ch := web.rxtime(500)
    if ch == -1 or ch == 10
      quit
    webbuff[i] := ch

  webbuff[i] := 0

  return i

DAT
index_htm     file      "thumper-index.htm"
index_htm_end byte      0

PRI loadConfig | len, ip, port, uri, ptr

  sd.popen(string("stations.txt"), "r")

  numstations := 0
  repeat
    len := readSDLine
    if len =< 0
      quit

    parseStationLine(@strTemp, @ip, @port, @uri)
    ptr := @stations + (numstations * 64)

    if strsize(uri) < 57
      long[ptr] := conv_endianlong(ip)
      word[ptr + 4] := port
      bytemove(ptr + 6, uri, strsize(uri) + 1)
      numstations++

    if numstations => MAXSTATIONS
      quit

  sd.pclose

  sd.popen(string("config.txt"), "r")

  readSDLine
  parseIPLine(@strTemp, @ip, @port)
  long[@ip_addr] := conv_endianlong(ip)
  readSDLine
  parseIPLine(@strTemp, @ip, @port)
  long[@ip_subnet] := conv_endianlong(ip)
  readSDLine
  parseIPLine(@strTemp, @ip, @port)
  long[@ip_gateway] := conv_endianlong(ip)
  readSDLine
  parseIPLine(@strTemp, @ip, @port)
  long[@ip_dns] := conv_endianlong(ip)

  readSDLine
  timeOffset := num.FromStr(@strTemp, num#DEC)

  sd.pclose

PRI parseStationLine(line, ip, port, uri) | pos

  pos := parseIPLine(line, ip, port)

  long[uri] := line + pos

PRI parseIPLine(line, ip, port) : pos | octet
  ' extracts the IP and PORT from a string

  long[ip] := 0
  word[port] := 0
  octet := 3
  repeat while octet => 0
    case byte[line][pos]
      "0".."9":
        byte[ip][octet] := (byte[ip][octet] * 10) + (byte[line][pos] - "0")
```

```
      ".":
        octet--
      ":":
        quit
      other:
        return -1
    pos++
  if octet <> 0
    return false
  if byte[line][pos++] == ":"
    repeat while byte[line][pos] <> 0 and byte[line][pos] <> "/"
      if byte[line][pos] => "0" and byte[line][pos] =< "9"
        word[port] := (word[port] * 10) + (byte[line][pos] - "0")
      else
        return -1
      pos++

  return pos

PRI readSDLine | ch, i

  i := 0
  repeat
    ch := \sd.pgetc
    if ch == 13
      next
    if ch == 10 or ch < 0 or i => 127
      quit
    strTemp[i++] := ch

  strTemp[i] := 0

  return i

PRI initMP3Decoder
  mp3.Start(MP3_MOSI, MP3_MISO, MP3_CLK, MP3_CS, MP3_DCS, MP3_DREQ, SRAM1_CS, SRAM2_CS)
  'mp3.SetMode(mp3#STREAM, 1)                              ' enable streaming mode
  mp3.SetVolume(volume, balance)                          ' default volume
  mp3.SetBassBoost(bassboost, 0)

  {bytefill(@mp3buff, 0, 32)
  repeat constant(2048 / 32)                              ' write 2048 zero bytes after reset
    mp3.WriteDataBuffer(@mp3buff)}

PRI mp3Player | i, ircode, numfiles, fileidx, isPlaying, writecount, starttime, songtime, dispcnt, volcnt, fsize,
hasmeta, scrollpos, updatefile, playnext

  isPlaying := false
  playnext := false

  numfiles := _getMP3FileCount
  fileidx := 0
  updatefile := true
  repeat

    if isPlaying
      ' play song mp3 decoder update
      if mp3.SRAMFree => CHUNKLEN

        if (i := sd.pread(@mp3buff, CHUNKLEN)) > 0
          ' copy file bytes to sram buffer
          mp3.SRAMWriteData(@mp3buff, i)

        if mp3.SRAMBytes =< 32
          ' finished playing song (buffer completely empty)
          sd.pclose
          isPlaying := false
          mp3.DMASet(0)
          mp3.SetVolume(1, 0)
          mp3.SendZeros
          lcd.cls
          updatefile := true

      else
        'writeCount := 0
        if (volcnt - cnt) < 0
          mp3.SetVolume(volume, balance)
```

```
      if (dispcnt - cnt) < 0
        lcd.pos(1,1)
        if hasmeta
          ' scrolly metadata
          if strsize(@strMeta) > 16
            i := scrollpos++    ' scroll the display
          else
            i := 0              ' don't scroll (song title fits on the line)
          repeat 16
            if i < strsize(@strMeta)
              lcd.out(strMeta[i++])
            else
              lcd.out(" ")
          if scrollpos => strsize(@strMeta)
            scrollpos := 0
        else
          lcd.str(@strTemp)
        lcd.clearRestOfLine
        lcd.pos(2, 1)
        i := rtc.getTimestamp - starttime
        lcd.str(num.ToStr(i / 60, num#DEC) + 1)
        lcd.out(":")
        lcd.str(num.ToStr(i // 60, num#DEC3) + 1)
        lcd.out(" ")
        lcd.str(num.ToStr(fsize / 1000, num#DEC) + 1)
        lcd.str(string("KB"))
        lcd.clearRestOfLine

        dispcnt := (clkfreq / 2) + cnt

  else
    ' not playing, so just display selected song
    if updatefile
      if _getMP3File(fileidx, @strTemp)
        lcd.cls
        hasmeta := _readID3Tags(@strTemp)
        scrollpos := 0
      else
        abort -1
      updatefile := false
      dispcnt := cnt

    if (dispcnt - cnt) < 0
      lcd.pos(2,1)
      lcd.str(@strTemp)
      lcd.clearRestOfLine
      lcd.pos(1,1)
      if hasmeta
        if strsize(@strMeta) > 16
          i := scrollpos++    ' scroll the display
        else
          i := 0              ' don't scroll (song title fits on the line)
        repeat 16
          if i < strsize(@strMeta)
            lcd.out(strMeta[i++])
          else
            lcd.out(" ")
        if scrollpos => strsize(@strMeta)
          scrollpos := 0
        dispcnt := (clkfreq / 2) + cnt
      else
        lcd.clearRestOfLine

  if (ircode := irHandle) < 0
    quit

  if ircode == ir#skipback
    fileidx--
    updatefile := true
  elseif ircode == ir#skipfwd
    fileidx++
    updatefile := true
  elseif ircode == ir#play or playnext
    ifnot isPlaying
      hasmeta := _readID3Tags(@strTemp)
      sd.popen(@strTemp, "r")
      fsize := sd.get_filesize
      isPlaying := true
```

```
        starttime := rtc.getTimestamp
        scrollpos := 0
        dispcnt := cnt
        volcnt := (clkfreq / 4) + cnt

        playnext := false

        mp3.SRAMEmpty
        mp3.DMASet(1)
    elseif ircode == ir#bstop
      sd.pclose
      isPlaying := false
      mp3.DMASet(0)
      mp3.SetVolume(1, 0)
      mp3.SendZeros
      lcd.cls
      updatefile := true

    if fileidx => numfiles
      fileidx := 0
    if fileidx < 0
      fileidx := numfiles - 1

    if ircode == ir#skipback or ircode == ir#skipfwd
      if isPlaying
        ' move to next song
        sd.pclose
        isPlaying := false
        mp3.DMASet(0)
        mp3.SetVolume(1, 0)
        mp3.SendZeros
        lcd.cls
        updatefile := true

        playnext := true

PRI _getMP3FileCount : numfiles

  sd.opendir
  repeat
    if sd.nextfile(@strTemp) < 0
      quit
    if str.indexOf(@strTemp, string(".MP3")) <> -1
      numfiles++
  sd.pclose

PRI _getMP3File(idx, buff) | i

  sd.opendir

  i := 0
  repeat
    if sd.nextfile(buff) < 0
      quit
    if str.indexOf(buff, string(".MP3")) <> -1
      if i == idx
        sd.pclose
        return true
      i++

  sd.pclose
  return false

PRI _generateNextFilename(ptr) | idx

  idx := _getMP3FileCount

  repeat idx from idx to 999
    bytemove(ptr, string("REC"), 3)
    bytemove(ptr + 3, num.ToStr(idx, num#DEC4) + 1, 3)
    bytemove(ptr + 6, string(".MP3"), 5)

    if \sd.popen(ptr, "r") < 0
      ' could not open file, so it must be non-existant!
      return idx
    else
      \sd.pclose
```

```
  return -1

PRI _extractMetaTitleArtist | tidx
  ' parse the meta string for title and artist information
  ' we assume it's in the format "Artist - Title"

  bytefill(@strArtist, 0, 31)
  bytefill(@strTitle, 0, 31)

  if (tidx := str.indexOf(@strMeta, string(" - "))) > -1
    ' it's a valid artist + title meta string
    bytemove(@strArtist, @strMeta, tidx <# 30)          ' get artist string, max of 30 characters
    tidx += @strMeta + 3                                ' generate pointer to title string
    bytemove(@strTitle, tidx, strsize(tidx) <# 30)      ' get title string, max of 30 characters

PRI _readID3Tags(filename) | i

  sd.popen(filename, "r")
  if \sd.seek(sd.get_filesize - 128) < 0                ' seek to the start of a possible ID3v1 TAG
    return false

  if sd.pgetc == "T" and sd.pgetc == "A" and sd.pgetc == "G"
    ' extract tags
    sd.pread(@strTitle, 30)                             ' read title
    sd.pread(@strArtist, 30)                            ' read artist

    strTitle[30] := 0
    strArtist[30] := 0

    ' generate Artist - Title metadata string
    i := strsize(@strArtist)
    bytemove(@strMeta, @strArtist, i)
    bytemove(@strMeta[i], string(" - "), 3)
    i += 3
    bytemove(@strMeta[i], @strTitle, strsize(@strTitle) + 1)                    ' built in zero termination

    result := true
  else
    result := false                                     ' no ID3v1 tags

  sd.pclose

PRI _writeID3Tags

  ' http://www.vbaccelerator.com/home/vb/code/vbmedia/audio/Reading_and_Writing_MP3_ID3v1_and_v2_Tags/article.asp
  { Tag As String * 3           '-- 03 = "TAG"
    Title As String * 30        '-- 33
    Artist As String * 30       '-- 63
    Album As String * 30        '-- 93
    Year As String * 4          '-- 97
    Comment As String * 30      '-- 127
    Genre As Byte               '-- 128 }

  sd.pwrite(string("TAG"), 3)
  sd.pwrite(@strTitle, 30)
  sd.pwrite(@strArtist, 30)
  repeat constant(30 + 4)
    sd.pputc(0)
  sd.pwrite(string("Recorded with Thumper"), 21)
  repeat constant(30 - 21)
    sd.pputc(0)
  sd.pputc(12)                  ' Genre = Other

PRI connect(addr, port, uri) | len, bytesrecv, blockbytes, metaint, metasize, metarecv, i, j, writecount, bitrate,
scrollpos, dispcnt, dispscreen, starttime, songtime, ircode, recording, recordsinglesong, rectime, volcnt, fname[4]

  metaint := 0
  bytesrecv := 0
  metarecv := 0
  blockbytes := 0
  scrollpos := 0

  writecount := 0

  bitrate := 0
  dispscreen := 0

  recording := false
```

```
strMeta[0] := 0

sock.connect(addr, port, @tcp_mp3rxbuff, 4096, @tcp_mp3txbuff, 64)
'sock.resetBuffers
if sock.waitConnectTimeout(1500)
  ' connected to remote host
  sock.str(string("GET "))
  sock.str(uri)
  sock.str(string(" HTTP/1.0",13,10))
  'sock.str(string("Host: propserve.fwdweb.com",13,10))
  sock.str(string("User-Agent: PropTCP-iRadio",13,10))
  sock.str(string("Connection: close",13,10))
  sock.str(string("Icy-MetaData:1",13,10,13,10))

  ' parse icy headers
  repeat
    readLine
    if strsize(@strTemp) == 0
      ' done with header parsing, start dumping to mp3 decoder
      quit
    if str.indexOf(@strTemp, string("icy-metaint:")) == 0
      ' got icy metadata interval header
      metaint := num.FromStr(@strTemp[12], num#DEC)
    elseif str.indexOf(@strTemp, string("icy-br:")) == 0
      bitrate := num.FromStr(@strTemp[7], num#DEC)

  starttime := rtc.getTimestamp

  dispcnt := cnt

  'fillBuffer

  mp3.DMASet(1)

  repeat while sock.isConnected

    'fillBuffer

    if (ircode := irHandle) < 0
      ' channel change
      quit

    if ircode == ir#dvddsply or ircode == ir#discmenu
      ifnot recording
        if _generateNextFilename(@fname) => 0
          _extractMetaTitleArtist
          sd.popen(@fname, "w")
          recording := true
          rectime := rtc.getTimestamp
          recordsinglesong := (ircode == ir#discmenu)
      else
        _writeID3Tags
        sd.pclose
        recording := false

    if metaint > 0 and blockbytes == metaint
      ' we need to handle our meta data now
      metasize := readByte * 16
      metarecv++

      if metasize > 0
        ' got some new metadata
        bytefill(@strMeta, 0, 128)
        readData(@strMeta, metasize <# 127)
        metarecv += metasize <# 127

        if metasize > 127
          ' dump excess bytes that we don't want
          repeat (metasize - 127)
            readByte
            metarecv++

        if (i := str.indexOf(@strMeta, string("StreamTitle='"))) => 0
          ' we got a new title, parse it
          i += 13                                  ' seek past StreamTitle=
          j := str.indexOf(@strMeta[i], string("';")) ' find end point
          bytemove(@strMeta, @strMeta[i], j)       ' shift the string so it's only the title
```

```
          strMeta[j] := 0                              ' string end

          {i := num.ToStr(bitrate, num#DEC) + 1        ' form bitrate descriptor string

          strMeta[j++] := " "                          ' head
          strMeta[j++] := "["
          bytemove(@strMeta[j], i, strsize(i))         ' copy in the bitrate
          j += strsize(i)
          bytemove(@strMeta[j], string(" kbps]"), 6)   ' tail
          strMeta[j + 6] := 0}

          scrollpos := 0
          songtime := rtc.getTimestamp

          if recording and recordsinglesong
            ' stop recording
            _writeID3Tags
            sd.pclose
            recording := false

        blockbytes := 0

      if mp3.SRAMFree => CHUNKLEN 'and sock.rxcount => CHUNKLEN
        ' SRAM can take more data, so fill it up

        len := readData(@mp3buff, CHUNKLEN)

        {if len < CHUNKLEN
          ' our buffer didn't fully fill up, try to fill the rest of the bytes
          repeat until len == CHUNKLEN
            mp3buff[len++] := readByte}

        if bytesrecv == 0
          volcnt := (clkfreq / 4) + cnt

        bytesrecv += len
        blockbytes += len

        mp3.SRAMWriteData(@mp3buff, len)

        if recording
          sd.pwrite(@mp3buff, len)

      if ina[MP3_DREQ] == 1 {and mp3.SRAMBytes => 32}
      ' decoder needs a data chunk

        'mp3.SRAMToDecoder
        'mp3.WriteDataBuffer(@mp3buff)

        outa[LED_STATUS]~~

        {if writecount++ > constant(32768 / 32)
          ' the MP3 decoder's buffer isn't filling up for some reason (likely because it locked up)
          ' the built in buffer is 16384 bytes, so we assume a failure occurs after two full buffers are
unacknowledged
          initMP3Decoder
          writecount := 0}

      else

        if (volcnt - cnt) < 0
          mp3.SetVolume(volume, balance)

        'writecount := 0         ' the decoder accepted the last bytestream so we reset our 'unacked' counter

        {vga.out($01)
        vga.str(@strMeta)
        vga.str(string($0A, $01, $0B, $05, "Bytes Recv: "))
        vga.dec(bytesrecv)
        vga.str(string("     ",13))
        vga.str(string(" Meta Recv: "))
        vga.dec(metarecv)
        vga.str(string("     ",13))
        vga.str(string(" Meta Interval: "))
        vga.dec(metaint)
        vga.str(string("     ",13))
        vga.str(string(" Retries: "))
        vga.dec(connretries)
```

```
        vga.str(string("     ",13))}

        if (dispcnt - cnt) < 0
          lcd.pos(1,1)
          if strsize(@strMeta) > 16
            i := scrollpos++    ' scroll the display
          else
            i := 0              ' don't scroll (song title fits on the line)
          repeat 16
            if i < strsize(@strMeta)
              lcd.out(strMeta[i++])
            else
              lcd.out(" ")
          if scrollpos => strsize(@strMeta)
            scrollpos := 0

          lcd.pos(2,1)

          ifnot recording
            case dispscreen
              0..10, 31:
                lcd.str(num.ToStr(bytesrecv / 1000, num#DEC) + 1)
                lcd.str(string("KB "))
                lcd.str(num.ToStr(bitrate, num#DEC) + 1)
                lcd.str(string("kbps"))
                if dispscreen == 31
                  dispscreen := 0
              11..20:
                i := rtc.getTimestamp - songtime
                if i => 0
                  lcd.str(num.ToStr(i / 60, num#DEC) + 1)
                  lcd.out(":")
                  lcd.str(num.ToStr(i // 60, num#DEC3) + 1)
                lcd.str(string(" / "))
                i := rtc.getTimestamp - starttime
                lcd.str(num.ToStr(i / 60, num#DEC) + 1)
                lcd.out(":")
                lcd.str(num.ToStr(i // 60, num#DEC3) + 1)
              21..30:
                lcd.str(num.ToStr(mp3.SRAMBytes, num#DEC) + 1)
                lcd.str(string(" / "))
                lcd.str(num.ToStr(sock.rxcount, num#DEC) + 1)

            dispscreen++
          else
            ' recording
            lcd.str(@fname)
            lcd.out(" ")
            i := rtc.getTimestamp - rectime
            lcd.str(num.ToStr(i / 60, num#DEC) + 1)
            lcd.out(":")
            lcd.str(num.ToStr(i // 60, num#DEC3) + 1)


          lcd.clearRestOfLine

          dispcnt := (clkfreq / 2) + cnt

        outa[LED_STATUS]~

  sock.close
  return 0

PRI irHandle | ircode

  ircode := ir.getIrCode

  if ircode == ir#NoNewCode
    ircode := webControl

  webControl := ir#NoNewCode

  if ircode <> ir#NoNewCode
    ' got a new remote control code

    'lcd.pos(2,14)
    'lcd.str(num.ToStr(ircode, num#DEC) + 1)
```

```
    if ircode == ir#chUp
      stationidx++
    elseif ircode == ir#chDn
      stationidx--
    elseif ircode == ir#volUp
      volume := (volume + 5) <# 255
      mp3.SetVolume(volume, balance)
    elseif ircode == ir#volDn
      volume := (volume - 5) #> 0
      mp3.SetVolume(volume, balance)
    elseif ircode == ir#left or ircode == ir#left2
      balance := (balance - 1) #> -20
      mp3.SetVolume(volume, balance)
    elseif ircode == ir#right or ircode == ir#right2
      balance := (balance + 1) <# 20
      mp3.SetVolume(volume, balance)
    elseif ircode == ir#select or ircode == ir#select2
      balance := 0
      bassboost := 0
      mp3.SetVolume(volume, balance)
      mp3.SetBassBoost(bassboost, 0)
    elseif ircode == ir#up or ircode == ir#up2
      bassboost := (bassboost + 10) <# 127
      mp3.SetBassBoost(bassboost, 0)
    elseif ircode == ir#down or ircode == ir#down2
      bassboost := (bassboost - 10) #> 0
      mp3.SetBassBoost(bassboost, 0)
    elseif ircode == ir#power
      isOn := not isOn
      return -2
    elseif ircode == ir#tvvideo or ircode == ir#tvvideo2
      playMode := (!playMode) & 1
      return -3
    else
      ' return all other codes to caller
      return ircode

    if ircode == ir#chUp or ircode == ir#chDn
      if stationidx => numstations
        stationidx := 0
      if stationidx < 0
        stationidx := numstations - 1
      return -1

  return 0

PRI readLine | i, rx

  i := 0
  repeat
    rx := sock.rxtime(500)
    if rx == -1
      strTemp[i] := 0
      abort -1
    if rx == 13
      next
    if rx == 10 or i => 127
      quit
    strTemp[i++] := rx

  strTemp[i] := 0

  return i

PRI readByte : ch

  if (ch := sock.rxtime(5000)) < 0
    abort -1

PRI readData(ptr, forcelen) : len | t
' Fills the whole buffer, waiting for the socket to fill before reading
' Built in 5 second timeout

  t := cnt
  repeat until sock.rxcount => forcelen or (cnt - t) / clkfreq > 5

  if sock.rxcount < forcelen
    abort -1
```

```
  len := sock.rxdata(ptr, forcelen)

  {if (len := sock.rxdatatime(ptr, maxlen, 5000)) < 0
    abort -1}

PRI conv_endianlong(in)
  return (in.byte[0] << 24) + (in.byte[1] << 16) + (in.byte[2] << 8) + (in.byte[3])

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

9.2 api_telnet_serial.spin

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

  Copyright (c) 2006-2009 Harrison Pham <harrison@harrisonpham.com>

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
  THE SOFTWARE.

  The latest version of this software can be obtained from
  http://hdpham.com/PropTCP and http://obex.parallax.com/
}}

'' NOTICE:  All buffer sizes must be a power of 2!

OBJ
  tcp : "driver_socket"

VAR
  long handle
  word listenport
  byte listening

  long ptrrxbuff, ptrtxbuff
  word rxlen, txlen

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

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

PUB stop

  tcp.stop

PUB connect(ipaddr, remoteport, _ptrrxbuff, _rxlen, _ptrtxbuff, _txlen)

  {if tcp.isValidHandle(handle)
    close}

  listening := false
  handle := -1
  handle := tcp.connect(ipaddr, remoteport, _ptrrxbuff, _rxlen, _ptrtxbuff, _txlen)

  return handle

PUB listen(port, _ptrrxbuff, _rxlen, _ptrtxbuff, _txlen)

  {if tcp.isValidHandle(handle)
    close}

  listenport := port
  ptrrxbuff := _ptrrxbuff
  rxlen := _rxlen
  ptrtxbuff := _ptrtxbuff
  txlen := _txlen
  listening := true
  handle := -1
  handle := tcp.listen(listenport, ptrrxbuff, rxlen, ptrtxbuff, txlen)

  return handle
```

```
PUB relisten
  if listening
    ifnot tcp.isValidHandle(handle)
      listen(listenport, ptrrxbuff, rxlen, ptrtxbuff, txlen)

PUB isConnected

  return tcp.isConnected(handle)

PUB rxcount

  return tcp.getReceiveBufferCount(handle)

PUB resetBuffers

  tcp.resetBuffers(handle)

PUB waitConnectTimeout(ms) : connected | t

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

PUB close

  tcp.close(handle)
  handle := -1

PUB rxflush

  repeat while rxcheck => 0

PUB rxcheck : rxbyte

  {if listening
    relisten
    rxbyte := tcp.readByteNonBlocking(handle)
  else}
    rxbyte := tcp.readByteNonBlocking(handle)
    if (not tcp.isConnected(handle)) and (rxbyte == -1)
      abort tcp#ERRSOCKETCLOSED

PUB rxtime(ms) : rxbyte | t

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

PUB rx : rxbyte

  repeat while (rxbyte := rxcheck) < 0

PUB rxdatatime(ptr, maxlen, ms) : len | t

  t := cnt
  repeat until (len := tcp.readDataNonBlocking(handle, ptr, maxlen)) => 0 or (cnt - t) / (clkfreq / 1000) > ms

PUB rxdata(ptr, maxlen)

  return tcp.readData(handle, ptr, maxlen)

PUB txflush

  tcp.flush(handle)

PUB txcheck(txbyte)

  {if listening
    relisten}

  ifnot tcp.isConnected(handle)
    abort tcp#ERRSOCKETCLOSED

  return tcp.writeByteNonBlocking(handle, txbyte)

PUB tx(txbyte)

  repeat while txcheck(txbyte) < 0

PUB txdata(ptr, len)
```

```
  {if listening
    relisten}

  tcp.writeData(handle, ptr, len)

PUB str(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")
```

9.3 driver_socket.spin

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

  Copyright (c) 2006-2009 Harrison Pham <harrison@harrisonpham.com>

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
  THE SOFTWARE.

  The latest version of this software can be obtained from
  http://hdpham.com/PropTCP and http://obex.parallax.com/
}}

'' NOTICE:  All buffer sizes must be a power of 2!

CON
' ****************************************
' **      Versioning Information       **
' ****************************************
  version    = 5                 ' major version
  release    = 2                 ' minor version
  apiversion = 8                 ' api compatibility version

' ****************************************
' **     User Definable Settings       **
' ****************************************
  sNumSockets     = 4            ' max number of concurrent registered sockets (max of 255)

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

CON
' ****************************************
' **      Return Codes / Errors        **
' ****************************************

  RETBUFFEREMPTY  = -1           ' no data available
  RETBUFFERFULL   = -1           ' buffer full

  ERRGENERIC      = -1           ' generic errors

  ERR             = -100         ' error codes start at -100
  ERRBADHANDLE    = ERR - 1      ' bad socket handle
  ERROUTOFSOCKETS = ERR - 2      ' no free sockets available
  ERRSOCKETCLOSED = ERR - 3      ' socket closed, could not perform operation

OBJ
  nic : "driver_enc28j60"

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

CON
' ****************************************
' **   Socket Constants and Offsets    **
' ****************************************

' 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

' ****************************************
' **  TCP State Management Constants   **
' ****************************************
  TIMEOUTMS       = 500         ' (milliseconds) timeout before a retransmit occurs
  RSTTIMEOUTMS    = 2000        ' (milliseconds) timeout before a RST is sent to close the connection
  WINDOWUPDATEMS  = 25          ' (milliseconds) window advertisement frequency

  MAXUNACKS       = 6           ' max number of unacknowledged retransmits before the stack auto closes the socket
                                ' timeout = TIMEOUTMS * MAXUNACKS (default: 500ms * 5 = 3000ms)

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

  MAXPAYLOAD      = 1200        ' maximum TCP payload (data) in bytes, this only applies when your txbuffer_length >
payload size

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_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

  lock_id       byte 0                                  ' socket handle lock

  packet        byte 0[nic#MAXFRAME]                    ' the ethernet frame

' ****************************************
' **        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
  SocketArrayStart
  lMySeqNum     long    0[sNumSockets]
  lMyAckNum     long    0[sNumSockets]
  lSrcIp        long    0[sNumSockets]
  lTime         long    0[sNumSockets]

                word
  wSrcPort      word    0[sNumSockets]
  wDstPort      word    0[sNumSockets]
  wLastWin      word    0[sNumSockets]
  wLastTxLen    word    0[sNumSockets]
  wNotAcked     word    0[sNumSockets]

                byte
  bSrcMac       byte    0[sNumSockets * 6]
```

```
  bConState     byte    0[sNumSockets]
  SocketArrayEnd

' ****************************************
' **      Circular Buffer Arrays       **
' ****************************************
                        word
  FifoDataStart
  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_tailnew            word    0[sNumSockets]          ' the new tx_tail value (unacked data)

  rxbuffer_length       word    0[sNumSockets]          ' each socket's buffer sizes
  txbuffer_length       word    0[sNumSockets]

  rxbuffer_mask         word    0[sNumSockets]          ' each socket's buffer masks for capping buffer sizes
  txbuffer_mask         word    0[sNumSockets]

                        long
  tx_bufferptr          long    0[sNumSockets]          ' pointer addresses to each socket's buffer spaces
  rx_bufferptr          long    0[sNumSockets]
  FifoDataEnd

PUB start(cs, sck, si, so, xtalout, macptr, ipconfigptr)
'' 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(@SocketArrayStart, 0, @SocketArrayEnd - @SocketArrayStart)

  ' reset buffer pointers, zeros a contigous set of bytes, starting at rx_head
  bytefill(@FifoDataStart, 0, @FifoDataEnd - @FifoDataStart)

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

PUB stop
'' Stop the driver

  if cog
    cogstop(cog~ - 1)           ' stop the tcp engine
    nic.stop                    ' stop nic driver (kills spi engine)
    lockclr(lock_id)            ' clear lock before returning it to the pool
    lockret(lock_id)            ' return the lock to the lock pool

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

  lock_id := locknew                                                    ' checkout a lock from the HUB
  lockclr(lock_id)                                                      ' clear the lock, just in case it was
in a bad state

  ' Start the ENC28J60 driver in a new cog
  nic.start(cs, sck, si, so, 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

  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
      repeat while lockset(lock_id)
      tick_tcpsend                                                              '  occurs every 10 cycles, since
incoming packets more important
      lockclr(lock_id)

      i := 0
      nic.banksel(nic#EPKTCNT)                                                  ' re-select the packet count bank

PRI service_packet

  ' lets process this frame
  nic.get_frame(@packet)

  ' check for arp packet type (highest priority obviously)
  if packet[enetpacketType0] == $08 AND packet[enetpacketType1] == $06
    if packet[constant(arp_hwtype + 1)] == $01 AND packet[arp_prtype] == $08 AND packet[constant(arp_prtype + 1)] ==
$00 AND packet[arp_hwlen] == $06 AND packet[arp_prlen] == $04
      if packet[arp_tipaddr] == ip_addr[0] AND packet[constant(arp_tipaddr + 1)] == ip_addr[1] AND
packet[constant(arp_tipaddr + 2)] == ip_addr[2] AND packet[constant(arp_tipaddr + 3)] == ip_addr[3]
        case packet[constant(arp_op + 1)]
          $01 : handle_arp
          $02 : repeat while lockset(lock_id)
                handle_arpreply
                lockclr(lock_id)
        '++count_arp
  else
    if packet[enetpacketType0] == $08 AND packet[enetpacketType1] == $00
      if packet[ip_destaddr] == ip_addr[0] AND packet[constant(ip_destaddr + 1)] == ip_addr[1] AND
packet[constant(ip_destaddr + 2)] == ip_addr[2] AND packet[constant(ip_destaddr + 3)] == ip_addr[3]
        case packet[ip_proto]
          'PROT_ICMP : 'handle_ping
                       'ser.str(stk.GetLength(0, 0))
                       'stk.GetLength(30, 19200)
                       '++count_ping
          PROT_TCP :  repeat while lockset(lock_id)
                      \handle_tcp                                               ' handles abort out of tcp handlers
(no socket found)
                      lockclr(lock_id)
                      '++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(packet[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(packet[enetpacketSrc0 + i])

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

  return nic.send_frame

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

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

  found := false
  if ip == LONG[@ip_gateway]
    ' find a handle that wants gateway mac
    repeat handle from 0 to constant(sNumSockets - 1)
      if bConState[handle] == SCONNECTINGARP2G
        found := true
        quit
  else
    ' find the one that wants this arp
    repeat handle from 0 to constant(sNumSockets - 1)
      if bConState[handle] == SCONNECTINGARP2
        if lSrcIp[handle] == ip
          found := true
          quit

  if found
    bytemove(@bSrcMac[handle * 6], @packet + arp_shaddr, 6)
    bConState[handle] := SCONNECTING

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

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

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

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

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

  if (bConState[handle] == SSYNSENT OR bConState[handle] == SESTABLISHED) AND (packet[TCP_hdrflags] & TCP_ACK) AND
datain_len > 0
    ' ACK, without SYN, with data

    ' set socket state, established session
    bConState[handle] := SESTABLISHED

    i := packet[constant(TCP_seqnum + 3)] << 24 + packet[constant(TCP_seqnum + 2)] << 16 + packet[constant(TCP_seqnum
+ 1)] << 8 + packet[TCP_seqnum]
    if lMyAckNum[handle] == i
      if datain_len =< (rxbuffer_mask[handle] - ((rx_head[handle] - rx_tail[handle]) & rxbuffer_mask[handle]))
        ' we have buffer space
        ptr := rx_bufferptr[handle]
        if (datain_len + rx_head[handle]) > rxbuffer_length[handle]
          bytemove(ptr + rx_head[handle], @packet[TCP_data], rxbuffer_length[handle] - rx_head[handle])
          bytemove(ptr, @packet[TCP_data] + (rxbuffer_length[handle] - rx_head[handle]), datain_len -
(rxbuffer_length[handle] - rx_head[handle]))
        else
          bytemove(ptr + rx_head[handle], @packet[TCP_data], datain_len)
        rx_head[handle] := (rx_head[handle] + datain_len) & rxbuffer_mask[handle]
      else
        datain_len := 0
```

```
  else
    ' we had a bad ack number, meaning lost or out of order packet
    ' we have to wait for the remote host to retransmit in order
    datain_len := 0

  ' recalculate ack number
  lMyAckNum[handle] := conv_endianlong(conv_endianlong(lMyAckNum[handle]) + datain_len)

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

elseif (bConState[handle] == SSYNSENTCL) AND (packet[TCP_hdrflags] & TCP_SYN) AND (packet[TCP_hdrflags] & TCP_ACK)
  ' We got a server response, so we ACK it

  bytemove(@lMySeqNum[handle], @packet + TCP_acknum, 4)
  bytemove(@lMyAckNum[handle], @packet + TCP_seqnum, 4)

  lMyAckNum[handle] := conv_endianlong(conv_endianlong(lMyAckNum[handle]) + 1)

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

  ' set socket state, established session
  bConState[handle] := SESTABLISHED

elseif (bConState[handle] == SLISTEN) AND (packet[TCP_hdrflags] & TCP_SYN)
  ' Reply to SYN with SYN + ACK

  ' copy mac address so we don't have to keep an ARP table
  bytemove(@bSrcMac[handle * 6], @packet + enetpacketSrc0, 6)

  ' copy ip, port data
  bytemove(@lSrcIp[handle], @packet + ip_srcaddr, 4)
  bytemove(@wSrcPort[handle], @packet + TCP_srcport, 2)
  bytemove(@wDstPort[handle], @packet + TCP_destport, 2)

  ' get updated ack numbers
  bytemove(@lMyAckNum[handle], @packet + TCP_seqnum, 4)

  lMyAckNum[handle] := conv_endianlong(conv_endianlong(lMyAckNum[handle]) + 1)
  lMySeqNum[handle] := conv_endianlong(++pkt_isn)                ' Initial seq num (random)

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

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

  ' set socket state, waiting for establish
  bConState[handle] := SSYNSENT

elseif (bConState[handle] == SESTABLISHED OR bConState[handle] == SCLOSING2) AND (packet[TCP_hdrflags] & TCP_FIN)
  ' Reply to FIN with RST

  ' get updated sequence and ack numbers (gaurantee we have correct ones to kill connection with)
  bytemove(@lMySeqNum[handle], @packet + TCP_acknum, 4)
  bytemove(@lMyAckNum[handle], @packet + TCP_seqnum, 4)

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

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

  ' set socket state, now free
  bConState[handle] := SCLOSED
  return

elseif (bConState[handle] == SSYNSENT) AND (packet[TCP_hdrflags] & TCP_ACK)
  ' if just an ack, and we sent a syn before, then it's established
  ' this just gives us the ability to send on connect
  bConState[handle] := SESTABLISHED
```

```
  elseif (packet[TCP_hdrflags] & TCP_RST)
    ' Reset, reset states
    bConState[handle] := SCLOSED
    return

  if (bConState[handle] == SESTABLISHED OR bConState[handle] == SCLOSING) AND (packet[TCP_hdrflags] & TCP_ACK)
    wNotAcked[handle] := 0                                  ' reset retransmit counter
    ' check to see if our last sent data has been ack'd
    i := packet[TCP_acknum] << 24 + packet[constant(TCP_acknum + 1)] << 16 + packet[constant(TCP_acknum + 2)] << 8 +
packet[constant(TCP_acknum + 3)]
    if i == (conv_endianlong(lMySeqNum[handle]) + wLastTxLen[handle])
      ' we received an ack for our last sent packet, so we update our sequence number and buffer pointers
      lMySeqNum[handle] := conv_endianlong(conv_endianlong(lMySeqNum[handle]) + wLastTxLen[handle])
      tx_tail[handle] := tx_tailnew[handle]
      wLastTxLen[handle] := 0

      tcpsend(handle)                                       ' send data

PRI build_ipheaderskeleton(handle) | hdrlen, hdr_chksum

  bytemove(@packet + ip_destaddr, @lSrcIp[handle], 4)                          ' Set destination address

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

  bytemove(@packet + enetpacketDest0, @bSrcMac[handle * 6], 6)                 ' Set destination mac address

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

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

  packet[ip_vers_len] := $45
  packet[ip_tos] := $00

  ++pkt_id

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

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

  packet[ip_ttl] := $80                                                        ' TTL = 128

  packet[ip_proto] := $06                                                      ' TCP protocol

PRI build_tcpskeleton(handle, flags) | size

  bytemove(@packet + TCP_srcport, @wDstPort[handle], 2)                        ' Source port
  bytemove(@packet + TCP_destport, @wSrcPort[handle], 2)                       ' Destination port

  bytemove(@packet + TCP_seqnum, @lMySeqNum[handle], 4)                        ' Seq Num
  bytemove(@packet + TCP_acknum, @lMyAckNum[handle], 4)                        ' Ack Num

  packet[TCP_hdrlen] := $50                                                    ' Header length

  packet[TCP_hdrflags] := flags                                                ' TCP state flags

  ' we have to recalculate the window size often otherwise our stack
  ' might explode from too much data :(
  size := (rxbuffer_mask[handle] - ((rx_head[handle] - rx_tail[handle]) & rxbuffer_mask[handle]))
  wLastWin[handle] := size

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

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

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

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

  ' calc ip header checksum
  packet[ip_hdr_cksum] := $00
  packet[constant(ip_hdr_cksum + 1)] := $00
  hdrlen := (packet[ip_vers_len] & $0F) * 4
  hdr_chksum := calc_chksum(@packet[ip_vers_len], hdrlen)
```

```
  packet[ip_hdr_cksum] := hdr_chksum >> 8
  packet[constant(ip_hdr_cksum + 1)] := hdr_chksum

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

  lTime[handle] := cnt                      ' update last sent time (for timeout detection)

PRI find_socket(srcip, dstport, srcport) | handle, free_handle, listen_handle
  ' 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
  listen_handle := -1
  repeat handle from 0 to constant(sNumSockets - 1)
    if bConState[handle] <> SCLOSED
      if (lSrcIp[handle] == 0) OR (lSrcIp[handle] == conv_endianlong(srcip))
        ' ip match, ip socket srcip = 0, then will try to match dst port (find listening socket)
          if (wDstPort[handle] == conv_endianword(dstport)) {AND (WORD[handle_addr + sSrcPort] == 0 OR
WORD[handle_addr + sSrcPort] == conv_endianword(srcport))}
            if wSrcPort[handle] == conv_endianword(srcport)
              ' found exact socket match (established socket)
              return handle
            elseif wSrcPort[handle] == 0
              ' found a partial match (listening socket with no peer)
              listen_handle := handle
    elseif srcip == 0
      ' found a closed (unallocated) socket, save this as a free handle if we are searching for a free handle
      free_handle := handle     ' we found a free handle, may need this later

  if srcip <> 0
    ' return the listening handle we found
    if listen_handle <> -1
      return listen_handle
  else
    ' searched for a free handle
    if free_handle <> -1
      return free_handle

  ' could not find a matching socket / free socket...
  abort -1

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

  if tx_tail[handle] == tx_head[handle]
    ' no data in buffer, so just quit
    return

  ' we have data to send, so send it
  ptr := tx_bufferptr[handle]
```

```
  len := ((tx_head[handle] - tx_tail[handle]) & txbuffer_mask[handle]) <# MAXPAYLOAD
  if (len + tx_tail[handle]) > txbuffer_length[handle]
    bytemove(@packet[TCP_data], ptr + tx_tail[handle], txbuffer_length[handle] - tx_tail[handle])
    bytemove(@packet[TCP_data] + (txbuffer_length[handle] - tx_tail[handle]), ptr, len - (txbuffer_length[handle] -
tx_tail[handle]))
  else
    bytemove(@packet[TCP_data], ptr + tx_tail[handle], len)
  tx_tailnew[handle] := (tx_tail[handle] + len) & txbuffer_mask[handle]

  wLastTxLen[handle] := len

  build_ipheaderskeleton(handle)
  build_tcpskeleton(handle, TCP_ACK {constant(TCP_ACK | TCP_PSH)})
  send_tcpfinal(handle, len)                        ' send actual data

  send_tcpfinal(handle, 0)                          ' send an empty packet to force the other side to ACK (hack to
get around delayed acks)

  wNotAcked[handle]++                               ' increment unacked packet counter

PRI tick_tcpsend | handle, state, len

  repeat handle from 0 to constant(sNumSockets - 1)
    state := bConState[handle]

    if state == SESTABLISHED OR state == SCLOSING
      len := (rxbuffer_mask[handle] - ((rx_head[handle] - rx_tail[handle]) & rxbuffer_mask[handle]))
      if wLastWin[handle] <> len AND len => (rxbuffer_length[handle] / 2) AND ((cnt - lTime[handle]) / (clkfreq /
1000) > WINDOWUPDATEMS)
        ' update window size
        build_ipheaderskeleton(handle)
        build_tcpskeleton(handle, TCP_ACK)
        send_tcpfinal(handle, 0)

      if ((cnt - lTime[handle]) / (clkfreq / 1000) > TIMEOUTMS) OR wLastTxLen[handle] == 0
        ' send new data OR retransmit our last packet since the other side seems to have lost it
        ' the remote host will respond with another dup ack, and we will get back on track (hopefully)
        tcpsend(handle)

    if (state == SCLOSING)

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

      ' we now wait for the other side to terminate
      bConState[handle] := SCLOSING2

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

      arp_request_checkgateway(handle)

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

      lMySeqNum[handle] := conv_endianlong(++pkt_isn)
      lMyAckNum[handle] := 0

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

      bConState[handle] := SSYNSENTCL

    elseif (state == SFORCECLOSE) OR (state == SESTABLISHED AND wNotAcked[handle] => MAXUNACKS) OR (lookdown(state:
SCLOSING2, SSYNSENT, SSYNSENTCL, SCONNECTINGARP2, SCONNECTINGARP2G) {(state == SCLOSING2 OR state == SSYNSENT)} AND
((cnt - lTime[handle]) / (clkfreq / 1000) > RSTTIMEOUTMS))
      ' 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 exceeded MAXUNACKS tcp retransmits (remote host lost)
      '  3 - we get stuck in the SSCLOSING2 state
      '  4 - we don't get a response to our client SYNSENTCL state
      '  5 - we don't get an ARP response state SCONNECTINGARP2 or SCONNECTINGARP2G

      build_ipheaderskeleton(handle)
```

```
      build_tcpskeleton(handle, TCP_RST)
      send_tcpfinal(handle, 0)

      bConState[handle] := SCLOSED

PRI arp_request_checkgateway(handle) | ip_ptr

  ip_ptr := @lSrcIp[handle]

  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(conv_endianlong(LONG[ip_ptr]))
    bConState[handle] := SCONNECTINGARP2
  else
    arp_request(conv_endianlong(LONG[@ip_gateway]))
    bConState[handle] := SCONNECTINGARP2G

  lTime[handle] := cnt

PRI arp_request(ip) | 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
  repeat i from 3 to 0
    nic.wr_frame(ip.byte[i])

  ' send the request
  return nic.send_frame

' ******************************
' ** IP Packet Helpers (Calcs) **
' ******************************
PRI calc_chksum(ptr, 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(ptr, 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
  return (in.byte[0] << 24) + (in.byte[1] << 16) + (in.byte[2] << 8) + (in.byte[3])

PRI conv_endianword(in)
  'return ((in & $FF) << 8) + ((in & $FF00) >> 8)
  return (in.byte[0] << 8) + (in.byte[1])

PRI _handleConvert(userHandle, ptrHandle) | handle
' Checks to see if a handle index is valid
' Aborts if the handle is invalid

  handle := userHandle.byte[0]                          ' extract the handle index from the lower 8 bits

  if handle < 0 OR handle > constant(sNumSockets - 1)   ' check the handle index to make sure we don't go out of
bounds
    abort ERRBADHANDLE

  ' check handle to make sure it's the one we want (rid ourselves of bad user handles)
  ' the current check method is as follows:
  ' - compare sDstPort

  if wDstPort[handle] <> ((userHandle.byte[2] << 8) + userHandle.byte[1])
    abort ERRBADHANDLE

  ' if we got here without aborting then we can assume the handle is good
  LONG[ptrHandle] := handle

' ************************************
' ** Public Accessors (Thread Safe) **
' ************************************
PUB listen(port, _ptrrxbuff, _rxlen, _ptrtxbuff, _txlen) | handle
'' Sets up a socket for listening on a port
''    port = port number to listen on
''    ptrrxbuff = pointer to the rxbuffer array
''    rxlen = length of the rxbuffer array (must be power of 2)
''    ptrtxbuff = pointer to the txbuffer array
''    txlen = length of the txbuffer array (must be power of 2)
'' Returns handle if available, ERROUTOFSOCKETS if none available
'' Nonblocking

  repeat while lockset(lock_id)

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

  if handle < 0
    lockclr(lock_id)
    abort ERROUTOFSOCKETS

  rx_bufferptr[handle] := _ptrrxbuff
  tx_bufferptr[handle] := _ptrtxbuff
  rxbuffer_length[handle] := _rxlen
  txbuffer_length[handle] := _txlen
  rxbuffer_mask[handle] := _rxlen - 1
  txbuffer_mask[handle] := _txlen - 1
```

```
  lMySeqNum[handle] := 0
  lMyAckNum[handle] := 0
  lSrcIp[handle] := 0
  lTime[handle] := 0
  wLastTxLen[handle] := 0
  wNotAcked[handle] := 0
  bytefill(@bSrcMac[handle * 6], 0, 6)

  wSrcPort[handle] := 0                                 ' no source port yet
  wDstPort[handle] := conv_endianword(port)             ' we do have a dest port though

  wLastWin[handle] := rxbuffer_length[handle]

  tx_head[handle] := 0
  tx_tail[handle] := 0
  tx_tailnew[handle] := 0
  rx_head[handle] := 0
  rx_tail[handle] := 0

  ' it's now listening
  bConState[handle] := SLISTEN

  lockclr(lock_id)

  return ((port.byte[0] << 16) + (port.byte[1] << 8)) + handle

PUB connect(ipaddr, remoteport, _ptrrxbuff, _rxlen, _ptrtxbuff, _txlen) | handle, user_handle
'' 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
''   ptrrxbuff = pointer to the rxbuffer array
''   rxlen = length of the rxbuffer array (must be power of 2)
''   ptrtxbuff = pointer to the txbuffer array
''   txlen = length of the txbuffer array (must be power of 2)
'' Returns handle to new socket, ERROUTOFSOCKETS if no socket available
'' Nonblocking

  repeat while lockset(lock_id)

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

  if handle < 0
    lockclr(lock_id)
    abort ERROUTOFSOCKETS

  rx_bufferptr[handle] := _ptrrxbuff
  tx_bufferptr[handle] := _ptrtxbuff
  rxbuffer_length[handle] := _rxlen
  txbuffer_length[handle] := _txlen
  rxbuffer_mask[handle] := _rxlen - 1
  txbuffer_mask[handle] := _txlen - 1

  lMySeqNum[handle] := 0
  lMyAckNum[handle] := 0
  lTime[handle] := 0
  wLastTxLen[handle] := 0
  wNotAcked[handle] := 0
  bytefill(@bSrcMac[handle * 6], 0, 6)

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

  user_handle := ((ip_ephport.byte[0] << 16) + (ip_ephport.byte[1] << 8)) + handle

  ' copy in ip, port data (with respect to the remote host, since we use same code as server)
  lSrcIp[handle] := conv_endianlong(ipaddr)
  wSrcPort[handle] := conv_endianword(remoteport)
  wDstPort[handle] := conv_endianword(ip_ephport++)

  wLastWin[handle] := rxbuffer_length[handle]

  tx_head[handle] := 0
  tx_tail[handle] := 0
  tx_tailnew[handle] := 0
  rx_head[handle] := 0
  rx_tail[handle] := 0
```

```
  bConState[handle] := SCONNECTINGARP1

  lockclr(lock_id)

  return user_handle

PUB close(user_handle) | handle, state
'' Closes a connection

  _handleConvert(user_handle, @handle)

  repeat while lockset(lock_id)

  state := bConState[handle]

  if state == SESTABLISHED
    ' try to gracefully close the connection
    bConState[handle] := SCLOSING
  elseif state <> SCLOSING AND state <> SCLOSING2
    ' we only do an ungraceful close if we are not in ESTABLISHED, CLOSING, or CLOSING2
    bConState[handle] := SCLOSED

  lockclr(lock_id)

  ' wait for the socket to close, this is very important to prevent the client app from reusing the buffers
  repeat until (bConState[handle] == SCLOSING2) or (bConState[handle] == SCLOSED)

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

  if \_handleConvert(user_handle, @handle) <> 0
    return false

  return (bConState[handle] == SESTABLISHED)

PUB isValidHandle(user_handle) | handle
'' 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}

  if \_handleConvert(user_handle, @handle) < 0
    return false

  return (bConState[handle] <> SCLOSED)

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

  _handleConvert(user_handle, @handle)

  if rx_tail[handle] == rx_head[handle]
    return RETBUFFEREMPTY

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

  rxptr := rx_bufferptr[handle]

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

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

  return len

PUB readData(user_handle, ptr, maxlen) : len | handle
'' Reads bytes from the socket
'' Returns the number of read bytes
```

```
'' Will block until data is received

  _handleConvert(user_handle, @handle)

  repeat while (len := readDataNonBlocking(user_handle, ptr, maxlen)) < 0
    ifnot isConnected(user_handle)
      abort ERRSOCKETCLOSED

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

  _handleConvert(user_handle, @handle)

  rxbyte := RETBUFFEREMPTY
  if rx_tail[handle] <> rx_head[handle]
    ptr := rx_bufferptr[handle]
    rxbyte := BYTE[ptr][rx_tail[handle]]
    rx_tail[handle] := (rx_tail[handle] + 1) & rxbuffer_mask[handle]

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

  _handleConvert(user_handle, @handle)

  repeat while (rxbyte := readByteNonBlocking(user_handle)) < 0
    ifnot isConnected(user_handle)
      abort ERRSOCKETCLOSED

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

  _handleConvert(user_handle, @handle)

  if (txbuffer_mask[handle] - ((tx_head[handle] - tx_tail[handle]) & txbuffer_mask[handle])) < len
    return RETBUFFERFULL

  txptr := tx_bufferptr[handle]

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

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

  return len

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

  _handleConvert(user_handle, @handle)

  repeat while len > txbuffer_mask[handle]
    repeat while writeDataNonBlocking(user_handle, ptr, txbuffer_mask[handle]) < 0
      ifnot isConnected(user_handle)
        abort ERRSOCKETCLOSED
    len -= txbuffer_mask[handle]
    ptr += txbuffer_mask[handle]

  repeat while writeDataNonBlocking(user_handle, ptr, len) < 0
    ifnot isConnected(user_handle)
      abort ERRSOCKETCLOSED

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

  _handleConvert(user_handle, @handle)

  ifnot (tx_tail[handle] <> (tx_head[handle] + 1) & txbuffer_mask[handle])
    return RETBUFFERFULL
```

```
  ptr := tx_bufferptr[handle]
  BYTE[ptr][tx_head[handle]] := txbyte
  tx_head[handle] := (tx_head[handle] + 1) & txbuffer_mask[handle]

  return txbyte

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

  _handleConvert(user_handle, @handle)

  repeat while writeByteNonBlocking(user_handle, txbyte) < 0
    ifnot isConnected(user_handle)
      abort ERRSOCKETCLOSED

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

  _handleConvert(user_handle, @handle)

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

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

  _handleConvert(user_handle, @handle)

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

PUB getSocketState(user_handle) | handle
'' Gets the socket state (internal state numbers)
'' You can include driver_socket in any object and use the S... state constants for comparison

  _handleConvert(user_handle, @handle)

  return bConState[handle]

PUB getReceiveBufferCount(user_handle) | handle
'' Returns the number of bytes in the receive buffer

  _handleConvert(user_handle, @handle)

  return (rx_head[handle] - rx_tail[handle]) & rxbuffer_mask[handle]

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
```

9.4 driver_enc28j60.spin

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

  Copyright (c) 2006-2009 Harrison Pham <harrison@harrisonpham.com>

  Permission is hereby granted, free of charge, to any person obtaining a copy
  of this software and associated documentation files (the "Software"), to deal
  in the Software without restriction, including without limitation the rights
  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
  copies of the Software, and to permit persons to whom the Software is
  furnished to do so, subject to the following conditions:

  The above copyright notice and this permission notice shall be included in
  all copies or substantial portions of the Software.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
  THE SOFTWARE.

  The latest version of this software can be obtained from
  http://hdpham.com/PropTCP and http://obex.parallax.com/

  Constant Names / Code Logic based on code from
  Microchip Technology, Inc.'s enc28j60.c / enc28j60.h source files
}}

CON
  version = 6     ' major version
  release = 0     ' minor version

CON
' ****************************************
' **       ENC28J60 SRAM Defines       **
' ****************************************
  ' ENC28J60 Frequency
  enc_freq = 25_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    $02, $00, $00, $00, $00, $01

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

  packetheader    byte 0[6]

  'packet          byte 0[MAXFRAME]
```

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

  ' 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

  ' return the chip silicon version
  banksel(EREVID)
  return rd_cntlreg(EREVID)

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
  wr_phy(PHLCON, $0742)         ' $0472 => ledA = link, ledB = tx/rx
                                ' $0742 => ledA = tx/rx, ledB = link

  ' enable packet reception
  bfs_reg(ECON1, ECON1_RXEN)

PUB get_frame(pktptr) | 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(pktptr, 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_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

  ctramode := %0_00100_00_0000_0000_0000_0000_0000_0000 + _sck
  ctrbmode := %0_00100_00_0000_0000_0000_0000_0000_0000 + _do

  spi_setupfreqsynth(_freqpin)

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

PRI spi_stop
  if cog
    cogstop(cog~ - 1)
  ctra := 0
  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
    return

  dira[pin] := 1

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

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

              mov       frqb, #0                'disable ctrb increment
              mov       ctrb, ctrbmode


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_      andn      outa, clkpin
              shl       arg0, #24
              mov       phsb, arg0              ' data to write
              mov       frqa, freqw             ' 20MHz write frequency
              mov       phsa, #0                ' start at clocking at 0

              mov       ctra, ctramode          ' send data @ 20MHz
              rol       phsb, #1
              rol       phsb, #1
              rol       phsb, #1
              rol       phsb, #1
              rol       phsb, #1
              rol       phsb, #1
              rol       phsb, #1
              mov       ctra, #0                ' disable
              andn      outa, clkpin

spi_out__ret  ret


spi_in_       andn      outa, clkpin
              mov       phsa, phsr              ' start phs for clock
```

```
              mov       frqa, freqr             ' 10MHz read frequency
              nop

              mov       ctra, ctramode          ' start clocking
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              rcl       arg0, #1
              test      dipin, ina wc
              mov       ctra, #0                ' stop clocking
              rcl       arg0, #1
              andn      outa, clkpin

spi_in__ret   ret

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 to 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)
ctramode                long    0                       ' ctr mode for CLK
ctrbmode                long    0                       ' ctr mode for SPI Out

hffff                   long    $FFFF

freqr                   long    $2000_0000              'frequency of SCK /8 for receive
freqw                   long    $4000_0000              'frequency of SCK /4 for send
phsr                    long    $6000_0000

                                                        '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)
```

9.5 vs10xx_mp3.spin

```
{{
┌──────────────────────────────────────┐
│ vs1002 MP3 Decoder Driver            │
│ Author: Kit Morton                   │
│ Copyright (c) 2008 Kit Morton        │
│ See end of file for terms of use.    │
└──────────────────────────────────────┘


** Contains modifications by Harrison Pham for compatibility with VS1011e and VS1053 decoders.
** Also includes a external SPI SRAM FIFO Queue and DMA Transfer assembly driver for seamless buffers.
** Modifications (c) 2019 Harrison Pham.  Licensed under the MIT License.

This object provides high speed access to the vs10002 mp3 decoder from VLSI Solution (www.vlsi.fi)
Although this object was written for the vs1002 it should work for the vs1003 and vs1033.

This driver does not use the DREQ output from the vs1002 so be aware that your program needs to listen to this line.
If you are not using a 24.576 MHz crystal for the vs1002 then you have to set the clock multiplier. Refer to page 28
of the datasheet for more information.

Functions:
     Start                   This function starts the cog that the ASM driver runs in, and sets up the object of
                             operation. Always call this before using the object.
     WriteDataByte           Send one byte of mp3 data to decode.
     WriteDataBuffer         Send 32 bytes of mp3 data to the vs1002 from the memory address givin.
     Volume                  Returns the current volume of the chip
     SetVolume               Sets the Volume of playback. NewVol must be between 0 and 255. Balance must be between -
20 and 20, with -20
                             all the way to the left and 20 all the way to the right.
     SetBassBoost            Set how much the bass / treble is boosted (read the datasheet for more information on
the bass boost)
                             Values must be between 0 and 127
     SetFreqLimit            Set the lower frequency limit of your sound system, also used for bass boost. Value must
be between 0 and 15.
     Mode                    The current status of the Mode Control register. Bit is a bit mask for the bit of the
register you want.
     ReadReg                 This function allows you to read any register of the vs1002
     WriteReg                This function allows you to write any register of the vs1002
     Stop                    Kills the ASM driver cog.
}}

CON
  DIFF      = %00000000_00000001
  RESET     = %00000000_00000100
  OUTOFWAV  = %00000000_00001000
  PDOWN     = %00000000_00010000
  TESTS     = %00000000_00100000
  STREAM    = %00000000_01000000
  PLUSV     = %00000000_10000000
  DACT      = %00000001_00000000
  SDIORD    = %00000010_00000000
  SDISHARE  = %00000100_00000000
  SDINEW    = %00001000_00000000
  ADPCM     = %00010000_00000000
  ADPCM_HP  = %00100000_00000000

CON
  #2, WRITE_CMD,READ_CMD                                'RAM commands
  CMD_DONE  = $8000_0000

  rxbuffer_length = 32768
  rxbuffer_mask = rxbuffer_length - 1

VAR
  '     0          4        8         12        16    20     24     28
  Long  Operation, RegName, RegValue, DataAddr, Data, BHead, BTail, mp3buff[32/4]    ' Misc. hub variables
  Long  cog
  Byte  CurrentVolume
  Byte  CurrentBalance
  Word  ModeReg
  Byte  DREQPin

PUB start(_MOSIPin, _MISOPin, _CLKPin, _CSPin, _DCSPin, _DREQPin, _SRAM1CSPin, _SRAM2CSPin) : okay

  stop

  DREQPin := _DREQPin
```

```
  MOSI := |<_MOSIPin
  MISO := |<_MISOPin
  CLK  := |<_CLKPin
  CS   := |<_CSPin
  DCS  := |<_DCSPin
  DREQ := |<_DREQPin
  SRAM1CS := |<_SRAM1CSPin
  SRAM2CS := |<_SRAM2CSPin

  ctramode := %0_00100_00_0000_0000_0000_0000_0000_0000 + _CLKPin
  ctrbmode := %0_00100_00_0000_0000_0000_0000_0000_0000 + _MOSIPin

  dira[_DREQPin]~

  okay := cog := cognew(@entry, @Operation) + 1

  CurrentVolume := 0

  ModeReg := %0000_1000_0000_0000                       ' VS1002 mode

  repeat until ina[DREQPin] == 1
  WriteReg(0, ModeReg)
  WriteReg(3, 38912)                                    ' mp3 decoder xtal + pll settings

PUB stop
  if cog
    cogstop(cog~ - 1)

PUB WriteDataBuffer(OutputDataAddr)
  DataAddr := OutputDataAddr
  'repeat until Operation == 0
  Operation := 3
  repeat until Operation == 0

PUB WriteDataByte(OutputData)
  Data := OutputData
  'repeat until Operation == 0
  Operation := 4
  repeat until Operation == 0

PUB SetBassBoost(Bass, Treble) | NewBass
  ' formulas from http://www.vlsi.fi/player_vs1011_1002_1003/modularplayer/player_8c-source.html
  NewBass := 0
  if Bass > 0
    NewBass |= (Bass + 23) / 10
    NewBass |= (Bass >> 3) << 4
  if Treble > 0
    NewBass |= (((148 - Treble) >> 3) + 2) << 8
    NewBass |= ((Treble >> 3) - 8) << 12
  WriteReg(2, NewBass)

PUB SetFreqLimit(Value) | OldFreq
  OldFreq := ReadReg(2)
  OldFreq &= %00000000_00000000_00000000_11110000 'Clear Old Freq Vlaue
  OldFreq |= Value
  WriteReg(2,OldFreq)

PUB SetVolume(NewVol,NewBalance) | Output, Vol
  Vol := 255 - NewVol
  if NewBalance < 0
    Output := (Vol + NewBalance) << 8
    Output |= Vol - NewBalance
  else
    Output := (Vol + NewBalance) << 8
    Output |= Vol - NewBalance

  if CurrentVolume == NewVol and CurrentBalance == NewBalance
    return

  CurrentVolume := NewVol
  CurrentBalance := NewBalance

  WriteReg(11,Output)
  waitcnt(3773 + cnt)

PUB Mode(Bit)
  Return(ModeReg & Bit)
```

```
PUB SetMode(Bit, Value)
  if Value == 0
    ModeReg &= !Bit
  else
    ModeReg |= Bit

  WriteReg(0,ModeReg)

PUB ReadReg(CurrRegName)
  RegName := CurrRegName
  'repeat until Operation == 0
  Operation := 1
  repeat until Operation == 0
  waitcnt(3773 + cnt)

  return RegValue

PUB WriteReg(CurrRegName,CurrRegValue)
  RegName := CurrRegName
  RegValue := CurrRegValue
  'repeat until Operation == 0
  Operation := 2
  repeat until Operation == 0
  waitcnt(3773 + cnt)

' -------------------------------------------------------------------------------

PUB DMASet(en)

  Operation := 9 + (en & 1)
  repeat until Operation == 0

PUB SendZeros

  repeat constant(2048 * 2)
    repeat until ina[DREQPin] == 1
    WriteDataByte(0)

' -------------------------------------------------------------------------------
PUB SRAMEmpty

  'BHead := BTail := 0
  Operation := 11
  repeat until Operation == 0

{PUB SRAMWriteByte(_addr, _data)                          'Write 1 byte into RAM

  repeat until Operation == 0
  DataAddr := WRITE_CMD<<24 + _addr<<8 + _data           'start write
  Operation := 6
  repeat until Operation == 0

PUB SRAMReadByte(_addr)                                  'Read 1 byte from RAM

  repeat until Operation == 0
  DataAddr := READ_CMD<<24 + _addr<<8                    'start read
  Operation := 5
  repeat until Operation == 0
  return Data & $FF                                      'return data}

PUB SRAMWriteData(_ptr, _len)

  if SRAMFree < _len
    return -1

  'repeat until Operation == 0
  DataAddr := _ptr
  Data := _len
  Operation := 7
  repeat until Operation == 0

{PUB SRAMReadData(_ptr, _len)

  'repeat until Operation == 0
  DataAddr := _ptr
  Data := _len
  Operation := 8
```

```
  repeat until Operation == 0}

PUB SRAMFree

  return rxbuffer_mask - SRAMBytes

PUB SRAMBytes

  return ((BHead - BTail) & rxbuffer_mask)

{PUB SRAMToDecoder | i

  if SRAMBytes < 32
    return -1

  SRAMReadData(@mp3buff, 32)
  WriteDataBuffer(@mp3buff)}

DAT
              org
entry                                                   ' Setup pins
              or        dira,MOSI                           ' Make MOSI an output
              andn      outa,MOSI                           ' Set MOSI low

              andn      dira,MISO                           ' Make MISO an input

              or        dira,CLK                            ' Make CLK an output
              andn      outa,CLK                            ' Set CLK low

              or        dira,CS                             ' Make CS an output
              or        outa,CS                             ' Set CS High

              or        dira,DCS                            ' Make DCS an output
              or        outa,DCS                            ' Set DCS High

              andn      dira,DREQ                           ' DREQ input

              or        dira,SRAM1CS                        ' SRAM 1 CS output
              or        outa,SRAM1CS

              or        dira,SRAM2CS                        ' SRAM 2 CS output
              or        outa,SRAM2CS

              mov       frqb,#0


loop          wrlong    zero, par
subloop       rdlong    _Operation, par         wz        ' Wait for command
        if_z  jmp       #performdma
        'if_z  jmp       #:subloop

                                                        ' Jump to current operation
              cmp       _Operation,#1           wz          ' Compare _Operation to one and put the result in Z
        if_z  jmp       #_readreg                           ' If _Operation is one then jump to _readreg
              cmp       _Operation,#2           wz          ' Compare _Operation to two and put the result in Z
        if_z  jmp       #_writereg                          ' If _Operation is two then jump to _writereg
              cmp       _Operation,#3           wz          ' Compare _Operation to three and put the result in Z
        if_z  call      #_writedatabuffer                   ' If _Operation is three then jump to _writedatabuffer
              cmp       _Operation,#4           wz          ' Compare _Operation to four and put the result into Z
        if_z  jmp       #_writedatabyte                     ' If _Operation is four ten jump to _writedatabyte
              cmp       _Operation,#5           wz
        if_z  jmp       #sramread
              cmp       _Operation,#6           wz
        if_z  jmp       #sramwrite
              cmp       _Operation,#7           wz
        if_z  call      #sramblockw
              cmp       _Operation,#8           wz
        if_z  call      #sramblockr
              cmp       _Operation,#9           wz
        if_z  mov       AutoDMA, #0
              cmp       _Operation,#10          wz
        if_z  mov       AutoDMA, #1
              cmp       _Operation,#11          wz
        if_z  jmp       #resetbuffers
              jmp       #loop                               ' If _Operation is none of the above then loop around
again
```

```
resetbuffers  mov       t0, par
              add       t0, #20
              wrlong    zero, t0
              add       t0, #4
              wrlong    zero, t0
              jmp       #loop


performdma    cmp       AutoDMA, #0             wz
        if_z  jmp       #subloop                          ' no DMA op
              test      DREQ, ina               wz
        if_z  jmp       #subloop                          ' DREQ not asserted

              mov       t0, par
              add       t0, #28                         ' point to mp3buff
              mov       hubptr, t0
              mov       Outaddr, t0
              sub       t0, #8                          ' point to head
              rdlong    head, t0
              add       t0, #4                          ' point to tail
              rdlong    tail, t0

              mov       t1, head
              sub       t1, tail
              and       t1, srammask
              cmp       t1, #32                 wc
        if_c  jmp       #subloop                        ' not enough bytes in buff

              mov       LoopCount, #32
              call      #sramblockrdma
              call      #_writedatabufferdma

              jmp       #subloop

_readreg
              andn      outa,CS                         ' Set CS low, select the chip controll interface

              mov       TempAddr,par            wz      ' Move the address of the first hub variable to TempAddr
              add       TempAddr,#4                     ' Add offset for Output Variable
              rdlong    OutputBuffer,TempAddr   wc      ' Read Value of RegName form hub memory in to OutputBuffer

              or        OutputBuffer,ReadCmd            ' Or the read command onto the beginning of the buffer

              mov       BitMask,#%1                     ' Setup bit mask
              shl       BitMask,#16                     ' Move bitmask to 16th bit

              mov       LoopCount,#16                   ' Set number of bits for output loop counter

:output_loop
              shr       BitMask,#1                      ' Shift the bit mask to the right one bit
              test      OutputBuffer,BitMask    wc      ' Pull current bit out of OutputBuffer and put it on wc
              muxc      Outa,MOSI                       ' Set _MOSI pin the current bit
              call      #clock                          ' Send clock pulse

              djnz      LoopCount,#:output_loop         ' Decrement loop counter and loop back to
                                                        ' :input_loop, if LoopCount is zero then keep going

              andn      outa,MOSI                       ' Force MOSI low

              mov       LoopCount,#16                   ' Set number of bits for input loop couter

:input_loop
              test      MISO,ina                wc      ' Get the current bit form the MISO pin and put it in "C"
              rcl       InputBuffer,#1                  ' Rotate the current bit form "C" into InputBuffer
              call      #clock                          ' Send clock pulse

              djnz      LoopCount,#:input_loop          ' Decrement loop counter and loop back to
                                                        ' :input_loop, if LoopCount is zero then keep going

              mov       TempAddr,par            wz      ' Move the address of the first hub variable to TempAddr
              add       TempAddr,#8                     ' Add offset for Input
              wrlong    InputBuffer,TempAddr            ' Write InputBuffer to the hub ram

              or        outa,CS                         ' Set CS high, unselect the chip controll interface

              'mov       TempAddr,par            wz      ' Move the address of the first hub variable to TempAddr
              'wrlong    zero,TempAddr                   ' Clear Operation
```

```
              jmp       #loop                          ' Go back and wait for the next operation

_writereg
              andn      outa,CS                        ' Set CS low, select the chip controll interface

              mov       TempAddr,par            wz     ' Move the address of the first hub variable to TempAddr
              add       TempAddr,#4                    ' Add offset for Output Variable
              rdlong    OutputBuffer,TempAddr          ' Read Value of RegName form hub memory into OutputBuffer

              or        OutputBuffer,WriteCmd          ' Or the write command onto the beginning of the output
buffer

              mov       BitMask,#%1                    ' Setup bit mask
              shl       BitMask,#16                    ' Move bitmask to 16th bit

              mov       LoopCount,#16                  ' Set number of bits for output loop counter

:name_output_loop
              shr       BitMask,#1                     ' Shift the bit mask to the right one bit
              test      OutputBuffer,BitMask    wc     ' Pull current bit out of OutputBuffer and put it on wc
              muxc      Outa,MOSI                      ' Set _MOSI pin the current bit
              call      #clock                         ' Send clock pulse

              djnz      LoopCount,#:name_output_loop   ' Decrement loop counter and loop back to

              mov       TempAddr,par            wz     ' Move the address of the first hub variable to TempAddr
              add       TempAddr,#8                    ' Add offset for Output Variable
              rdlong    OutputBuffer,TempAddr          ' Read Value of Output form hub memory inot OutputBuffer

              mov       BitMask,#%1                    ' Setup bit mask
              shl       BitMask,#16                    ' Move bitmask to 16th bit

              mov       LoopCount,#16                  ' Set number of bits for output loop counter
                                                       ' :input_loop, if LoopCount is zero then keep going
:value_output_loop
              shr       BitMask,#1                     ' Shift the bit mask to the right one bit
              test      OutputBuffer,BitMask    wc     ' Pull current bit out of OutputBuffer and put it on wc
              muxc      Outa,MOSI                      ' Set _MOSI pin the current bit
              call      #clock                         ' Send clock pulse

              djnz      LoopCount,#:value_output_loop  ' Decrement loop counter and loop back to


              andn      outa,MOSI                      ' Force MOSI low

              or        outa,CS                        ' Set CS high, unselect the chip controll interface

              'mov       TempAddr,par            wz     ' Move the address of the first hub variable to TempAddr
              'wrlong    zero,TempAddr                  ' Clear Operation

              jmp       #loop                          ' Go back and wait for the next operation

_writedatabuffer
              mov       TempAddr,par            wz     ' Move the address of the first hub variable to TempAddr
              add       TempAddr,#12                   ' Add offset for DataAddr Variable
              rdlong    Outaddr,TempAddr        wc     ' Read Value of DataAddr form hub memory in to OutputAddr

_writedatabufferdma
              andn      outa,DCS                       ' Set DCS low, select the chip data interface
              mov       loopcount,#32                  ' Set the number of bytes to shift out

:Output
              rdbyte    OutputBuffer,Outaddr           ' Read the first byte from hub memory
              add       Outaddr,#1                     ' Incriment the address for the next byte

              mov       BitMask,#%1                    ' Setup bit mask
              shl       BitMask,#8                     ' Move bitmask to 16th bit

              mov       LoopCount2,#8                  ' Set number of bits for output loop counter

:output_loop
              shr       BitMask,#1                     ' Shift the bit mask to the right one bit
              test      OutputBuffer,BitMask    wc     ' Pull current bit out of OutputBuffer and put it on wc
              muxc      Outa,MOSI                      ' Set _MOSI pin the current bit
              call      #clock                         ' Send clock pulse

              djnz      LoopCount2,#:output_loop       ' Decrement loopcounter2 and loop back to
                                                       ' :output_loop, if LoopCount2 is zero then keep going
```

```
            djnz      LoopCount,#:Output              ' Decrement loopcounter and loop back to
                                                      ' :output, if LoopCount is zero then keep going

            andn      outa,MOSI                       ' Force MOSI low

            or        outa,DCS                        ' Set DCS high, unselect the chip data interface

            'mov       TempAddr,par            wz      ' Move the address of the first hub variable to TempAddr
            'wrlong    zero,TempAddr                   ' Clear Operation

_writedatabufferdma_ret
_writedatabuffer_ret
            ret                                       ' Go back and wait for the next operation

_writedatabyte
            andn      outa,DCS                        ' Set DCS low, select the chip data interface
            mov       TempAddr,par            wz      ' Move the address of the first hub variable to TempAddr
            add       TempAddr,#16                    ' Add offset for DataAddr Variable
            rdlong    OutputBuffer,TempAddr   wc      ' Read Value of Data form hub memory in to OutputBuffer

            mov       BitMask,#%1                     ' Setup bit mask
            shl       BitMask,#8                      ' Move bitmask to 16th bit

            mov       LoopCount,#8                    ' Set number of bits for output loop counter

:output_loop
            shr       BitMask,#1                      ' Shift the bit mask to the right one bit
            test      OutputBuffer,BitMask    wc      ' Pull current bit out of OutputBuffer and put it on wc
            muxc      Outa,MOSI                       ' Set _MOSI pin the current bit
            call      #clock                          ' Send clock pulse

            djnz      LoopCount,#:output_loop         ' Decrement loopcounter and loop back to
                                                      ' :output_loop, if LoopCount2 is zero then keep going

            andn      outa,MOSI                       ' Force MOSI low

            or        outa,DCS                        ' Set DCS high, unselect the chip data interface

            'mov       TempAddr,par            wz      ' Move the address of the first hub variable to TempAddr
            'wrlong    zero,TempAddr                   ' Clear Operation

            jmp       #loop                           ' Go back and wait for the next operation

clock
            mov       CLK,#0                  wz, nr  ' Load Z with 1
            mov       Time,cnt                        ' Move curent value of cnt to Time
            add       Time,#30                        ' Add wait time to Time
            muxz      outa,CLK                        ' Set CLK high
            waitcnt   Time,#30                        ' Wait 15 clock ticks
            muxnz     outa,CLK                        ' Set CLK low
clock_ret   ret



sramblockw              mov     t0, par               ' 12 = pointer, 16 = len
                        add     t0, #12
                        rdlong  hubptr, t0            ' hub ptr
                        add     t0, #4
                        rdlong  LoopCount, t0         ' LoopCount = len
                        add     t0, #4
                        rdlong  head, t0              ' get updated head ptr

:loop                   mov     TempData, head        ' address
                        shl     TempData, #8
                        add     TempData, cmdwrite    ' writecmd + address
                        rdbyte  t1, hubptr            ' byte to write
                        add     TempData, t1          ' writecmd + address + byte
                        call    #_sramwrite           ' write it

                        add     head, #1
                        and     head, srammask        ' increment head w/ wrap around
                        add     hubptr, #1            ' increment hub ptr
                        djnz    LoopCount, #:loop

                        wrlong  head, t0              ' write head back
sramblockw_ret          ret
```

```
sramblockr              mov     t0, par
                        add     t0, #12
                        rdlong  hubptr, t0              ' hub pointer to temporary buffer
                        add     t0, #4
                        rdlong  LoopCount, t0           ' LoopCount = len
                        add     t0, #8
                        rdlong  tail, t0                ' get updated tail

sramblockrdma
:loop                   mov     TempData, tail
                        shl     TempData, #8
                        add     TempData, cmdread       ' readcmd + address
                        call    #_sramread

                        and     TempData, #$FF          ' mask off everything but the byte
                        wrbyte  TempData, hubptr

                        add     tail, #1
                        and     tail, srammask
                        add     hubptr, #1
                        djnz    LoopCount, #:loop

                        wrlong  tail, t0                ' write tail back
sramblockrdma_ret
sramblockr_ret          ret

sramread                mov     TempAddr, par
                        add     TempAddr, #12           ' address
                        rdlong  TempData, TempAddr
                        call    #_sramread
                        add     TempAddr, #4
                        wrlong  TempData, TempAddr
                        jmp     #loop

_sramread               andn    outa,CLK                'send command and address to SRAM
                        andn    outa,SRAM1CS             'cs low
                        mov     phsb,TempData           '32bit data to be written

                        mov     frqa,freqw              'frequency of SCK
                        mov     phsa,#0                 'start clock

                        mov     ctrb,ctrbmode
                        mov     ctra,ctramode           'send cmd and address with 20MHz clock
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        mov     ctra,#0
                        mov     ctrb,#0
                        andn    outa,CLK

                        mov     phsa,phsr               'set counter to right value
                        mov     frqa,freqr              'frequency of SCK (10MHz)
                        nop

                        mov     ctra,ctramode           'read data byte
                        test    MISO,ina    wc          'L
                        rcl     TempData,#1                   '1 H
```

```
                        test    MISO,ina    wc
                        rcl     TempData,#1                   '2
                        test    MISO,ina    wc
                        rcl     TempData,#1                   '3
                        test    MISO,ina    wc
                        rcl     TempData,#1                   '4
                        test    MISO,ina    wc
                        rcl     TempData,#1                   '5
                        test    MISO,ina    wc
                        rcl     TempData,#1                   '6
                        test    MISO,ina    wc
                        rcl     TempData,#1                   '7
                        test    MISO,ina    wc
                        mov     ctra,#0
                        rcl     TempData,#1                   '8
                        andn    outa,CLK                'SCK low
                        or      outa,SRAM1CS             'disable SRAM

_sramread_ret           ret                             ' done


sramwrite               mov     TempAddr, par
                        add     TempAddr, #12           ' address
                        rdlong  TempData, TempAddr
                        call    #_sramwrite
                        jmp     #loop

_sramwrite              andn    outa,CLK                'send command and address to SRAM
                        andn    outa,SRAM1CS             'cs low
                        mov     phsb,TempData
                        mov     frqa,freqw              'frequency of SCK
                        mov     phsa,#0                 'start clock

                        mov     ctrb,ctrbmode
                        mov     ctra,ctramode           'send cmd,address and data with 20MHz clock
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
                        rol     phsb,#1
'                        rol     phsb,#1
                        mov     ctra,#0
                        mov     ctrb,#0
                        andn    outa,CLK
                        'mov     TempData,phsb
                        or      outa,SRAM1CS             'disable SRAM
                                                        'write data to ram
_sramwrite_ret          ret                             ' done

{
############################ Defined data #############################
}
```

```
zero          long      0                               ' Define as zero to 0
WriteCmd      long      %1000000000
ReadCmd       long      %1100000000

freqr                   long    $2000_0000              'frequency of SCK /8 for receive
freqw                   long    $4000_0000              'frequency of SCK /4 for send
phsr                    long    $6000_0000

'sramsize      long      rxbuffer_length
srammask      long      rxbuffer_mask

cmdwrite      long      WRITE_CMD<<24
cmdread       long      READ_CMD<<24

head          long      0
tail          long      0

AutoDMA       long      0

{
############################ Define Pins ############################
}

MOSI          long      0                               ' Pin number of MOSI
MISO          long      0                               ' Pin number of MISO
CLK           long      0                               ' Pin number of CLK
CS            long      0                               ' Pin number of CS
DCS           long      0                               ' Pin number of DCS

DREQ          long      0                               ' DREQ pin

SRAM1CS       long      0                               ' SRAM 1 CS Pin
SRAM2CS       long      0                               ' SRAM 2 CS Pin


ctramode      long      0                               ' ctr mode for CLK
ctrbmode      long      0                               ' ctr mode for SPI Out

{
########################## Other Variables ##########################
}

_Operation              res     1                       ' Variable that stores the current operaion
OutputBuffer            res     1                       ' Variable to store the data to be shifted out
InputBuffer             res     1                       ' Variable to store the data that has just been shifted in
TempAddr                res     1                       ' Temporary holder for address of first hub variable
Time                    res     1                       ' Use to store count to wait four
BitMask                 res     1                       ' Bit mask for geting bits out of OutputBuffer
LoopCount               res     1                       ' Used to keap track of how many times it has looped around
LoopCount2              res     1                       ' Used to keap track of how many times it has looped around
Outaddr                 res     1

TempData                res     1

sramptr                 res     1
hubptr                  res     1
t0                      res     1
t1                      res     1
t2                      res     1

              fit       496

{{
┌──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┐
│                                                    TERMS OF USE: MIT License                                                          │
├──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────┤
│Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation    │
│files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy,    │
│modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software│
│is furnished to do so, subject to the following conditions:
│
```

```
│
│
│The above copyright notice and this permission notice shall be included in all copies or substantial portions of the
Software.│
│
│
│THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
│
│WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS
OR         │
│COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE,    │
│ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
│
└──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
───────────┘
}}
```

9.6 driver_hd44780.spin

```
con
  'basepin = 16

  'rs= basepin+4                'Register select
  'en= basepin+5                'Enable
  'msb = basepin+3              'Highest dataline
  'lsb = basepin+0              'Lowest dataline
  'LCD Commands
  EightBitInit = 3    'Eight Bit mode
  FourBitInit = 2     'Four Bit mode
  ClearLcd = 1        'Clear the LCD
  CursorBlink = $0F   'Turn the cursor on and blink it
  NoCursor = $0C      'Turn the cursor off
  'LCD constants
  'lcdlines = 2        '# of lines on the LCD. Assumes the lines are a power of 2 (1,2,4,8,16,etc..)
  'linelength = 16     'LCD line length
  Line1 = $80         'Address of the First Line
  Line2 = $C0         'Address of the Second Line
  Line3 = $94         'Address of the Third Line
  Line4 = $D4         'Address of the Fourth Line
  lf = $0A            'Line Feed code
  cr = $0D            'Carriage Return code
  esc = $1B           'Escape code
  cgra = $40          'Address of the cgram
  off = 0             'Cursor State constant
  on = 1              'Cursor State constant
  space = $20

var
  byte CurrentLine    'Current Line position Value 0-3
  byte CurrentPos     'Current Column position Value 0-19
  byte CursorState
  byte ddra           'Screen Data address

  byte linelength, lcdlines
  byte rs, en, msb, lsb

pub start(_basepin, _linelength, _lcdlines)                                    'Initialize the LCD to four bit mode
and clear it

  rs := _basepin
  en := _basepin + 1
  msb := _basepin + 5
  lsb := _basepin + 2

  linelength := _linelength
  lcdlines := _lcdlines

  dira[msb..lsb]~~
  outa[msb..lsb]~
  dira[en..rs]~~
  outa[en..rs]~
  outa[msb..lsb] := EightBitInit
  enable
  uSdelay(5000)
  enable
  enable
  outa[msb..lsb] := FourBitInit
  enable
  commandOut(12)
  commandOut(6)
  commandOut(ClearLcd)
  CurrentLine := 0
  CurrentPos := 0
  CursorState := off
  uSdelay(5000)

pub str(stringptr)                                        'Write out a string to the LCD
  repeat while byte[stringptr] <> 0                       'test for end of string
    if byte[stringptr] > 7 AND byte[stringptr] < space
      case byte[stringptr]
        lf, cr:                                           'If a carriage return or line feed, go to a new line
          newline
      stringptr++
    else
      out(byte[stringptr++])
```

```
      'CurrentPos++
      if CurrentPos == linelength
        CurrentPos := 0

pub commandOut(char)            'Write out a command to the display controller
  outa[rs]~
  outa[msb..lsb] := char / 16
  enable
  outa[msb..lsb] := char & 15
  enable
  outa[rs]~~

pub out(character)              'Write out a single character to the display
  outa[msb..lsb] := character / 16
  enable
  outa[msb..lsb] := character & 15
  enable
  CurrentPos++

pub cls                         'Clear the display
  commandOut(ClearLcd)
  CurrentLine := 0
  CurrentPos := 0
  uSdelay(5000)
  if CursorState == on
    cursor_on
  else
    cursor_off

pub cll | lptr                  'Clear the current line
  home
  lptr := 0
  repeat while lptr < linelength
    out(" ")
    lptr++

pub clearRestOfLine
  repeat while CurrentPos < linelength
    out(" ")

pub cursor_on                   'Turn the cursor and blink on
  CursorState := on
  commandOut(CursorBlink)

pub cursor_off                  'Turn the cursor and blink off
  CursorState := off
  commandOut(NoCursor)

pub pos(line,column)            'Set the position
  CurrentLine := (line - 1) & (lcdlines-1)
  CurrentPos := column - 1
  checkddra
  ddra += CurrentPos
  commandOut(ddra)

pub home                        'Go back to the start of the line
  CurrentLine--
  newline

pub uSdelay(DelayuS)            'Delay for # of microseconds
  waitcnt((clkfreq/1_000_000) * DelayuS + cnt)

pri enable                      'Toggle the enable line
  outa[en]~~
  uSdelay(100)
  outa[en]~
  uSdelay(100)

pri newline                     'Go to the next line
  CurrentPos := 0
  CurrentLine++
  CurrentLine &= (lcdlines-1)
  checkddra
  commandOut(ddra)

pri checkddra                   'Generate the LCD line address
    case CurrentLine
      0: ddra := Line1          'Address of First Line
```

```
1: ddra := Line2          'Address of Second Line
2: ddra := Line3          'Address of Third Line
3: ddra := Line4          'Address of Fourth Line
```

9.7 util_strings.spin

```
'' String Utilities
'' ----------------
'' Copyright (C) 2006-2009 Harrison Pham

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)

PUB concat(dst, src1, src2) | len1
  '' Concats two strings

  len1 := strsize(src1)
  bytemove(dst, src1, len1)
  bytemove(dst + len1, src2, strsize(src2) + 1)
```

9.8 softrtc.spin

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

  (c) 2008 Harrison Pham.
}}


OBJ
  clsock : "api_telnet_serial"
  dt : "date_time_epoch"

CON
  #0,MODE_COUNTER,MODE_COG

VAR
  long timeset        ' stores the value to add to the counter time to get a unix timestamp
  byte tcp_rxbuff[32]
  byte tcp_txbuff[2]

  long stack[16]
  long cogseconds
  byte mode

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

  timeset := 0
  mode := _mode

  if mode == MODE_COUNTER
    initCounters(pin)
  else
    cognew(worker, @stack)

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
      \clsock.close
      return 1
    \clsock.close
    delay_ms(500)

  return -1

PRI worker | t

  t := cnt
  cogseconds := 0

  repeat
    waitcnt(t += clkfreq)
    cogseconds++

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

  clsock.connect(constant((132 << 24) + (163 << 16) + (4 << 8) + 102), 13, @tcp_rxbuff, 32, @tcp_txbuff, 2)
  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 mode == MODE_COUNTER
    if clkfreq == 80_000_000
      return phsb ** $35AFE535                            ' timer value is stored in phsb (seconds)
    else
      return phsb
  else
    return cogseconds

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)
```

9.9 date_time_epoch.spin

```
{{ date_time_ts.spin

  Bob Belleville

  2007/03/29 - essentially from scratch
          30 - ts_bump, maxm, adj
  2007/04/18 - extract to a single object
          19 - cleanup, test, and complete

  Historically personal computers have used a single
  32 bit integer to count the total seconds from a
  beginning time or epoch and to have routines to
  convert to/from calendar notation.  Time values
  can be easily compared.  Adding or subtracting
  intervals up to many days is easy.  (Although
  adding a month is harder.)

  Various epochs have been used.  Unix used Jan 1, 1970
  and MS-DOS used Jan 1, 1980.  The range of dates
  that can be represented is limited to 2^31 seconds
  for signed arithmetic (as in this implementation.)
  This a bit more than 68 years.

  Astronomers use a so called Julian Day to measure
  calendar time.  It is simply a count of days since
  a base year a long time ago.

  (Note: The term Julian Date is used to represent
  the day number in a given year and is often seen
  in date codes on food and other items.  This is
  not what we use here.)

  Julian Day number 0 was at noon GMT January 1, 4713 BC.
  As I write this the value is 2_454_209.

  This is big number and of little interest to most
  users.  Unix for example subtracts 2_440_588 from the
  JDN to get a 'Epoch Day.'

  Multiply the Epoch Day number by 86_400 (the number
  of seconds in a day) and add the number of elapse
  seconds in the current day since midnight and you
  have the current date and time coded in a long.

  Here is a worked example of the use of this object:

  May 2007 has two full moons.  The second is called
  a blue moon.  This definition of a blue moon is an
  error introduced by Sky and Telescope Magazine a
  long time ago which they will never undo.

  (http://en.wikipedia.org/wiki/Blue_moon)

  May 2, 2007 at  3:09 hours (am)
     31, 2007 at 18:04 hours (6:04pm) are both full
  moons.  These are Pacific Daylight time.

  (http://www.griffithobs.org/skyfiles/skymoonphases2007.html)

  jd := toJD(2007,5,2)
    will return 2_454_223
  spd := toSPD(3,9,0)
    will return 11_340
  using the unix epoch (check using http://www.csgnetwork.com/unixds2timecalc.html)

    2_454_223 - 2_440_588 -> 13_635 days x 86400 -> 1_178_064_000 + 11_340 -> 1_178_075_340
  so
  tv1 := toETV(2007,5,2,3,9,0)
    will return 1_178_075_340
  and
  tv2 := toETV(2007,5,31,18,4,0)
    will return 1_180_634_640

  The mean period from full moon to full moon is called the
  synodic month and is 29.530_588_853 days (Jean Meeus 1991)
```

```
  This is 2_551_443 seconds.  tv1 + 2_551_443 -> 1_180_626_783

  so
  date := dateETV( 1_180_626_783 )
  will return
    date>>16      -> 2007
    date>>8 & $FF -> 5
    date & $FF    -> 31

  and
  time := timeETV( 1_180_626_783 )
    time>>16      -> 15
    time>>8 & $FF -> 53
    time & $FF    -> 3

  which means that the mean moon is about 2 hours earlier
  than the true full moon.

  This shows how the methods are used and provides a test
  case.

  The advantages of this object are:

        Only 79 longs and no data used.

        Like other PC date/time systems.

        Easy to compute strange intervals.

        Easy and very fast to update and compare
        values.

        Only 4 bytes to store a full date/time.

  The disadvantages are:

        Somewhat complex conversion to and from ordinary
        human calendar values.

        Short span of valid years --- about 68 from epoch
        chosen.

  Timing:

        The pair of routines dateETV and timeETV take
        460 microseconds to execute.

        The routine toETV takes 270 microseconds to
        convert a calendar date and time to a long.

}}

CON

        _eunix  = 2_440_588     'Julian Day (+0.5) of Jan 1, 1970 the unix epoch
        _edos   = 2_444_240     'Julian Day (+0.5) of Jan 1, 1980 the ms-dos epoch
        _eprop  = 2_451_545     'Julian Day (+0.5) of Jan 1, 2000 the Propeller epoch?

        _epoch  = _eunix        'take your choice

PUB toJD(y,m,d) | jd, lc

{  Henry F. Fliegel and Thomas C. Van Flandern

     jd = ( 1461 * ( y + 4800 + ( m - 14 ) / 12 ) ) / 4 +
          ( 367 * ( m - 2 - 12 * ( ( m - 14 ) / 12 ) ) ) / 12 -
          ( 3 * ( ( y + 4900 + ( m - 14 ) / 12 ) / 100 ) ) / 4 +
          d - 32075

     converts calendar year, month and day to a Julian Day number
}

  lc~
  if m =< 2
    lc := -1
  return (1461*(y+4800+lc))/4+(367*(m-2-12*lc))/12-(3*((y+4900+lc)/100))/4+d-32075

PUB toSPD(h,m,s)
```

```
''  convert hour, minute and second to seconds per day
''  0..86399

  return h*3600 + m*60 + s

PUB timeETV(etv) | spd, h, m

''  return the time of a epoch time variable as three bytes
''  in a long H:M:S

  spd := etv // 86400
  h   := spd / 3600
  spd -= h*3600
  m   := spd / 60
  spd -= m*60
  return h<<16 | m<<8 | spd

PUB dateETV(etv)

''  return the date of a epoch time variable as a word and
''  two bytes Y/M/D

  return toCal((etv/86400) + _epoch)

PUB toETV(y,mo,d,h,m,s) : n

''  create a epoch time value for the given date and time

  return ( (toJD(y,mo,d) - _epoch) * 86400 ) + toSPD(h,m,s)

PUB toCal(jd) | l, n, i, j, d, m, y

{  Henry F. Fliegel and Thomas C. Van Flandern

        l = jd + 68569
        n = ( 4 * l ) / 146097
        l = l - ( 146097 * n + 3 ) / 4
        i = ( 4000 * ( l + 1 ) ) / 1461001
        l = l - ( 1461 * i ) / 4 + 31
        j = ( 80 * l ) / 2447
        d = l - ( 2447 * j ) / 80
        l = j / 11
        m = j + 2 - ( 12 * l )
        y = 100 * ( n - 49 ) + i + l

   converts a Julian Day Number to year, month and day
}

  l := jd + 68569
  n := ( 4 * l ) / 146097
  l := l - ( 146097 * n + 3 ) / 4
  i := ( 4000 * ( l + 1 ) ) / 1461001
  l := l - ( 1461 * i ) / 4 + 31
  j := ( 80 * l ) / 2447
  d := l - ( 2447 * j ) / 80
  l := j / 11
  m := j + 2 - ( 12 * l )
  y := 100 * ( n - 49 ) + i + l

  return y<<16 | m<<8 | d
```

9.10 IR_Remote.spin

```
{{
      IR_Remote_NewCog.spin
      Tom Doyle
      2 March 2007

      Panasonic IR Receiver - Parallax #350-00014

      Receive and display codes sent from a Sony TV remote control.
      See "Infrared Decoding and Detection appnote" and "IR Remote for the Boe-Bot Book v1.1"
      on Parallax website for additional info on TV remotes.

      The procedure uses counter A to measure the pulse width of the signals received
      by the Panasonic IR Receiver. The procedure waits for a start pulse and then decodes the
      next 12 bits. The entire 12 bit result is returned. The lower 7 bits contain the actual
      key code. The upper 5 bits contain the device information (TV, VCR etc.) and are masked off
      for the display.

      Most TV Remotes send the code over and over again as long as the key is pressed.
      This allows auto repeat for TV operations like increasing volume. The volume continues to
      increase as long as you hold the 'volume up' key down. Even if the key is pressed for a
      very short time there is often more than one code sent. The best way to eliminate the
      auto key repeat is to look for an idle gap in the IR receiver output. There is a period of
      idle time (20-30 ms) between packets. The getSonyCode procedure will wait for an idle period
      controlled by the gapMin constant. This value can be adjusted to eliminate auto repeat
      while maintaining a fast response to a new keypress. If auto repeat is desired the indicated
      section of code at the start of the getSonyCode procedure can be commented out.

      The procedure sets a tolerance for the width of the start bit and the logic level 1 bit to
      allow for variation in the pulse widths sent out by different remotes. It is assumed that a
      bit is 0 if it is not a 1.

      The procedure to read the keycode ( getSonyCode ) is run in a separate cog. This allows
      the main program loop to continue without waiting for a key to be pressed. The getSonyCode
      procedure writes the NoNewCode value (255) into the keycode variable in main memory to
      indicate that no new keycode is available. When a keycode is received it writes the keycode
      into the main memory variable and terminates. With only 8 cogs available it seems to be a
      good idea to free up cogs rather than let them run forever. The main program can fire off
      the procedure if and when it is interested in a new keycode.

}}


CON
  NoNewCode    =   255              ' indicates no new keycode received

  gapMin       =   2000             ' minimum idle gap - adjust to eliminate auto repeat
  startBitMin  =   2000             ' minimum length of start bit in us (2400 us reference)
  startBitMax  =   2800             ' maximum length of start bit in us (2400 us reference)
  oneBitMin    =   1000             ' minimum length of 1 (1200 us reference)
  oneBitMax    =   1400             ' maximum length of 1 (1200 us reference)

  ' Sony TV remote key codes
  ' http://www.hifi-remote.com/sony/Sony_tv.htm

  one   =  0
  two   =  1
  three =  2
  four  =  3
  five  =  4
  six   =  5
  seven =  6
  eight =  7
  nine  =  8
  zero  =  9

  chUp  = 16
  chDn  = 17
  volUp = 18
  volDn = 19
  mute  = 20
  power = 21
  last  = 59

  select = 101
  select2 = 11
  right  = 51
```

```
  right2 = 97
  left   = 52
  left2  = 98
  up     = 116
  up2    = 66
  down   = 117
  down2  = 67

  tvvideo = 37
  tvvideo2 = 42
  skipback = 87
  skipfwd = 86

  play = 26
  bstop = 24

  dvddsply = 90
  discmenu = 35


VAR

  byte  cog
  long  Stack[20]

  byte  currircode

PUB start(Pin)
'' Pin - propeller pin connected to IR receiver
'' addrMainCode - address of keycode variable in main memory

  stop
  return (cog := cognew(_getSonycode(Pin), @Stack) + 1)

PUB stop
'' stop cog if in use

  if cog
    cogstop(cog~ -1)

PUB getIrCode : ircode

  ircode := currircode
  currircode := NoNewCode

PRI _getSonyCode(pin) | irCode, index, pulseWidth, lockID
'' Decode the Sony TV Remote key code from pulses received by the IR receiver

   ' wait for idle period (ir receiver output = 1 for gapMin)
   ' comment out "auto repeat" code if auto key repeat is desired

  currircode := NoNewCode

  dira[pin]~

  repeat

    ' start of "auto repeat" code section
    'dira[pin]~
    index := 0
    repeat
      if ina[Pin] == 1
        index++
      else
        index := 0
    while index < gapMin
    ' end of "auto repeat" code section

    frqa := 1
    ctra := 0
    'dira[pin]~

    ' wait for a start pulse ( width > startBitMin and < startBitMax  )
    repeat
      ctra := (%10101 << 26 ) | (PIN)                      ' accumulate while A = 0
      waitpne(0 << pin, |< Pin, 0)
      phsa:=0                                              ' zero width
      waitpeq(0 << pin, |< Pin, 0)                         ' start counting
```

```
    waitpne(0 << pin, |< Pin, 0)                          ' stop counting
    pulseWidth := phsa  / (clkfreq / 1_000_000) + 1
  while ((pulseWidth < startBitMin) OR (pulseWidth > startBitMax))

  ' read in next 12 bits
  index := 0
  irCode := 0
  repeat
    ctra := (%10101 << 26 ) | (PIN)                       ' accumulate while A = 0
    waitpne(0 << pin, |< Pin, 0)
    phsa:=0                                               ' zero width
    waitpeq(0 << pin, |< Pin, 0)                          ' start counting
    waitpne(0 << pin, |< Pin, 0)                          ' stop counting
    pulseWidth := phsa  / (clkfreq / 1_000_000) + 1

    if (pulseWidth > oneBitMin) AND (pulseWidth < oneBitMax)
      irCode := irCode + (1 << index)
    index++
  while index < 11

  irCode := irCode & $7f                                    ' mask off upper 5 bits

  currircode := irCode
```

9.11 thumper_index.htm

```
<html><head><title>Thumper :: Home</title>
<META HTTP-EQUIV=Refresh CONTENT=2>
<style>
body {
        font: 12px Verdana;
        background: #dddddd;
}
h1 {
        font: bold 22px Verdana;
        width: 600px;
        background: #679fd4;
        border-top: 1px solid #043a6b;
        font-family: Helvetica, Arial, sans-serif;
}
th {
        font: bold 14px Verdana;
        text-align: left;
        background: #cccccc;
        border-width: 1px;
        border-style: dotted;
}
td {
        font: 14px Verdana;
        text-align: left;
        background: #cccccc;
        border-width: 1px;
        border-style: dotted;
}
hr {
        text-align: left;
        margin: 50 auto 0 0;
        width: 600px;
}
</style>
<script>
function pr(a,b) {
        document.write('<tr><th>'+a+'</th><td>'+b+'</td></tr>');
}
function pr3(a,b,c) {
        document.write('<tr><th>'+a+'</th><td>'+b+'</td><td>'+c+'</td></tr>');
}
function bt(a,b) {
        return '<input type="submit" name="'+a+'" value="'+b+'">';
}
</script>
</head>
<body>
<h1>Currently Playing</h1>
<table><script>
pr('Title','~01');
pr('Song Time','~03');
pr('Total Time','~04');
</script></table>
<h1>Control</h1>
<form method="GET" action="r.cgi">
<table><script>
pr3('Power','~05',bt('pwr','Power'));
pr3('Station','http://~06',bt('cup','Channel Up')+bt('cdn','Channel Down'));
pr3('Volume','~07',bt('vup','Volume Up')+bt('vdn','Volume Down'));
</script></table>
</form>
<h1>System Stats</h1>
<table><script>
pr('Time','~08');
pr('IP Address','~09');
pr('Buffer Underflows','~10');
pr('TCP Buffer','~11 KB / 8KB');
pr('SRAM Buffer','~12 KB / 64KB');
pr('Web Hits','~13');
</script></table>
<hr>
<i>&copy;2010 Harrison Pham.</i>
</body></html>
```