DAQPac

Automotive Data Logger

Project Number PC097232
Ryan David
February 26, 2010

Table of Contents

1. Project Description

DAQPac is an automotive data logger designed from the ground up for motorsports enthusiasts. It has all the features necessary for a driver to improve both their skill and vehicle performance. DAQPac was the result of two prior prototypes, each one evaluating the hardware for function at a reasonable cost. The end result is a well-balanced system, able to provide the features a motorsports enthusiast requires. Powered by a Propeller, the DAQPac interfaces to a powerful set of instrumentation, including:

- (18) 5-volt analog inputs for vehicle sensors
- (2) K-type thermocouple inputs
- (2) inputs for engine spark timing and RPM
- (3) outputs for controlling external relays (i.e. nitrous)
- Onboard X/Y/Z-axis accelerometer and Z-axis angular rate
- High sensitivity GPS with 10Hz update rate
- Secure Digital slot for recording
- Remote Telemetry with onboard long range wireless
- Communication to optional 4.3" touchscreen LCD

Due to the unique architecture of the Propeller, the DAQPac is able to process large amounts of information from several sources, simultaneously. A single cog can be assigned to each task, and the results shared amongst other cogs. For example, a cog is assigned to read the analog inputs at a high rate, another cog to write the data to the secure digital card, and yet another to control relays based on that same data. The counters available to each cog in this design were taken advantage of as well. Specifically, one of the cogs is assigned to poll two thermocouples. While the cog is performing the conversions, both counters are working in the background counting negative edges on different engine tachometer signals.

Robustness of the module was crucial in the design as well. Precision automotive electronics have the difficult task of isolating themselves from a noisy electrical environment. Special care was taken to minimize noise appearing in measurements, and protecting the system from unintentional damage. There is over and under voltage protection available on every analog and digital input, which also protects against transients. Digital outputs are current limited, and auxiliary power connections are protected with user replaceable fuses. The power supply is protected against reverse polarity, and the system can recognize over and under voltages. Finally, during system startup, the accelerometer and gyroscope are put into a self-test mode to verify proper bias and operation.

Mechanically, DAQPac was designed for a sealed, metal enclosure. Both the main connector and antenna connectors have a rubber seal, and the USB and Secure Digital socket can accept a rubber flap. Production boards will also receive a conformal coating to further protect against moisture.

For more information, please watch a walkthrough available on YouTube at
http://www.youtube.com/watch?v=dh-QhLt0Krg

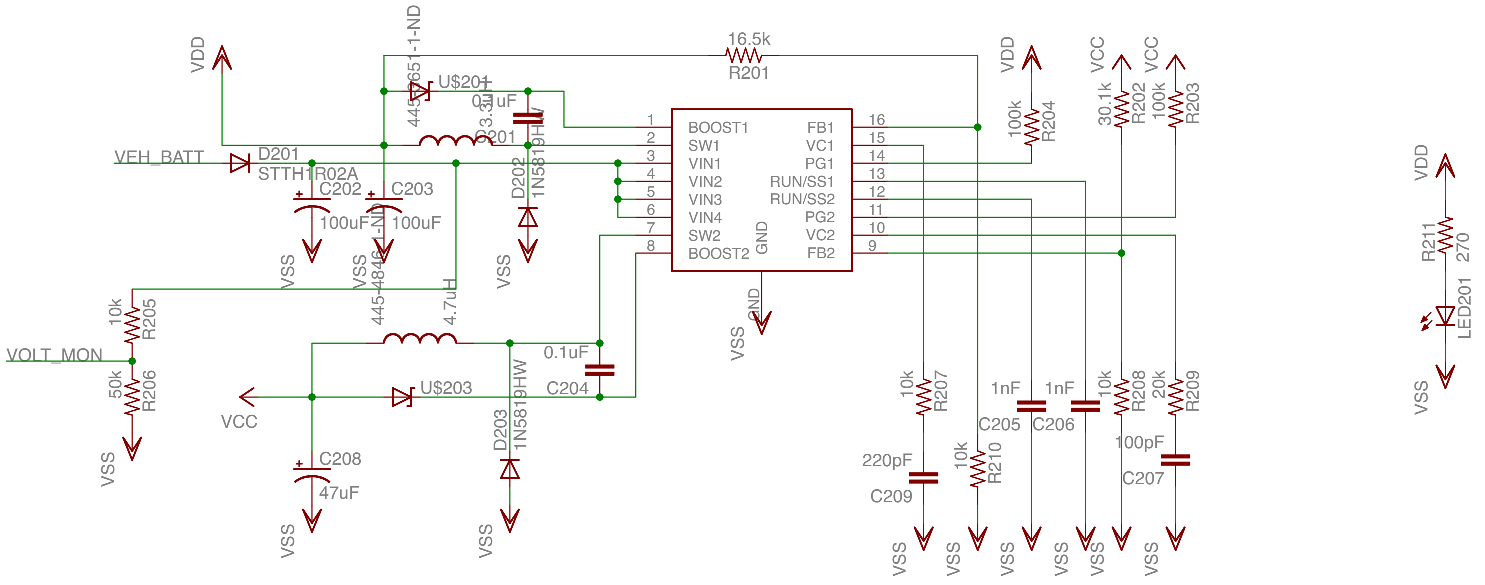
2.1 Schematic

DAQPac Schematic

Rev. 2 - February 26, 2010

TITLE: Datalogger_rev2	
Document Number:	REV:
Date: 2/15/10 9:35 PM	Sheet: 1/10

Power Supply



TITLE: Datalogger_rev2

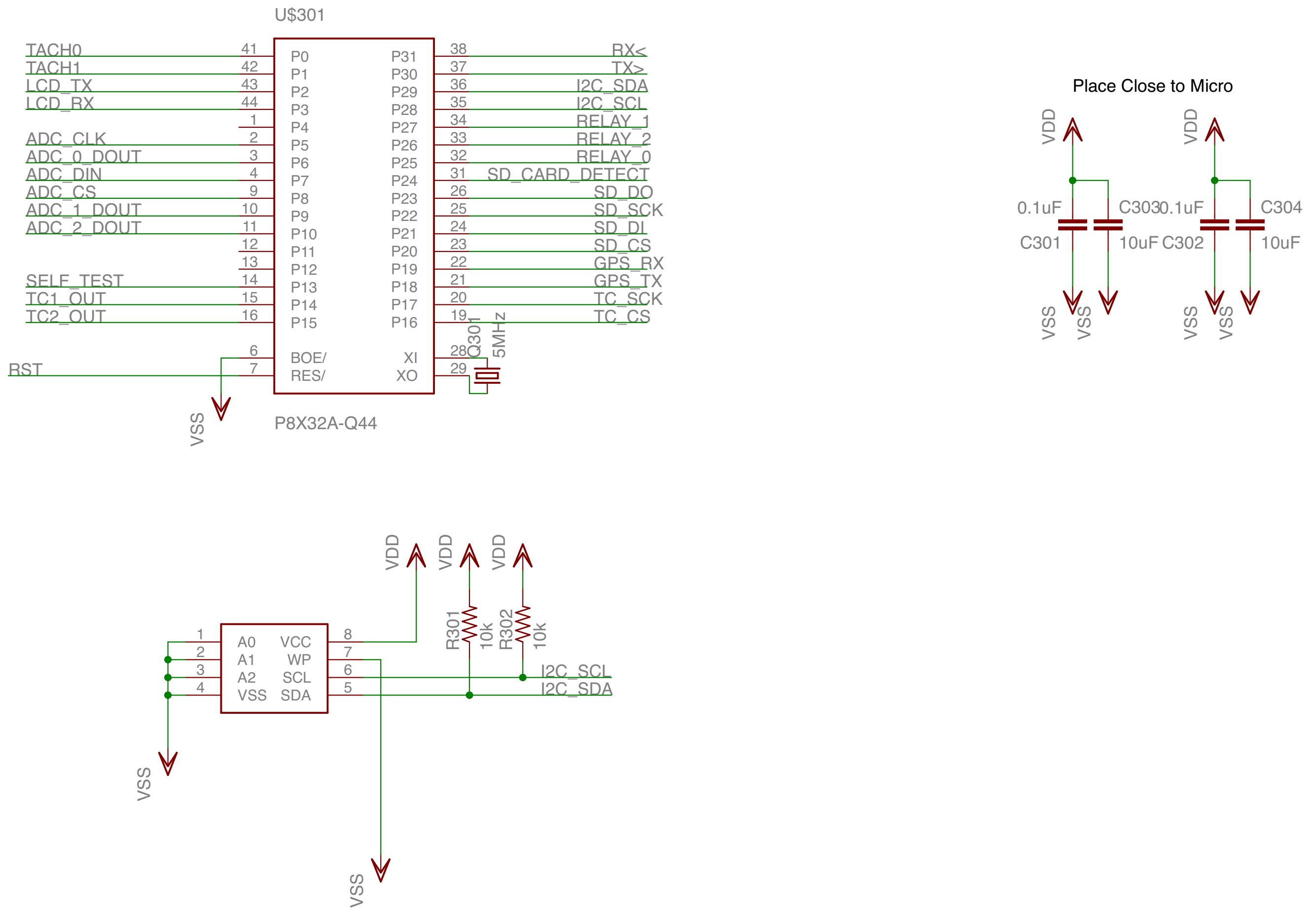
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 2/10

Microcontroller And EEPROM



TITLE: Datalogger_rev2

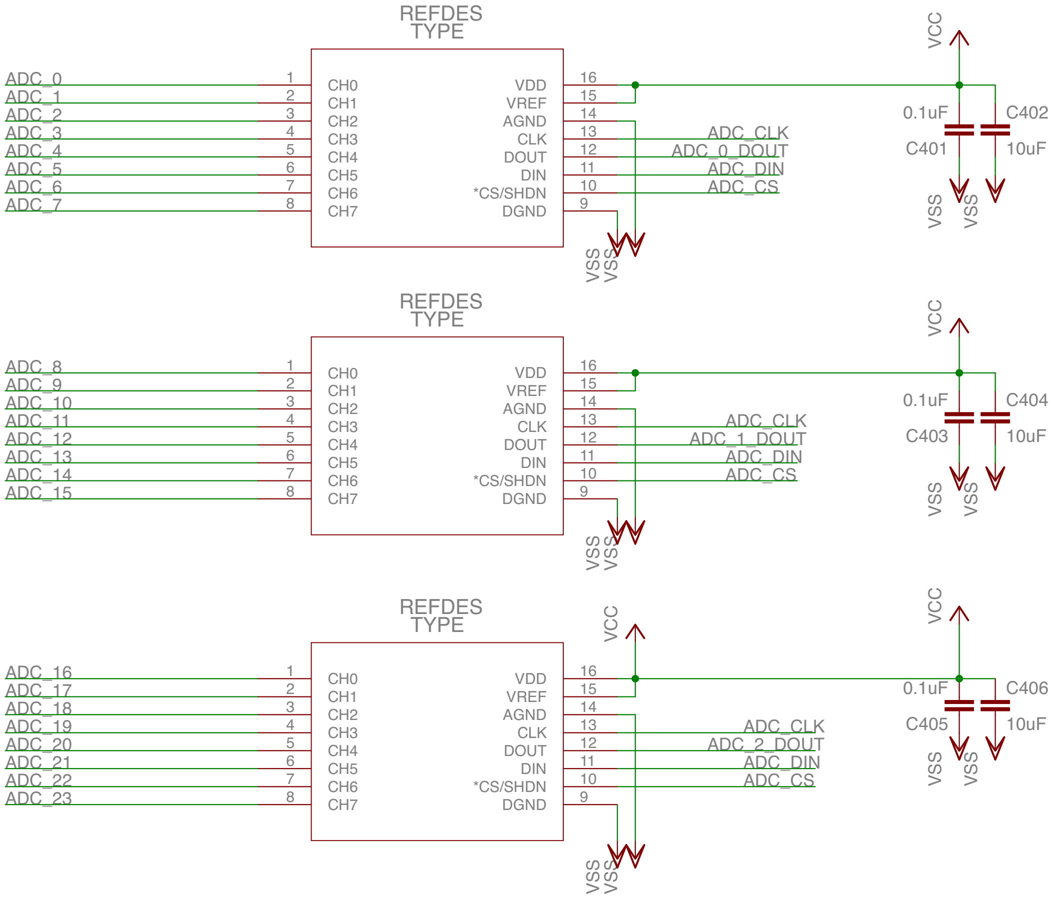
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 3/10

Analog Digital Converters



TITLE: Datalogger_rev2

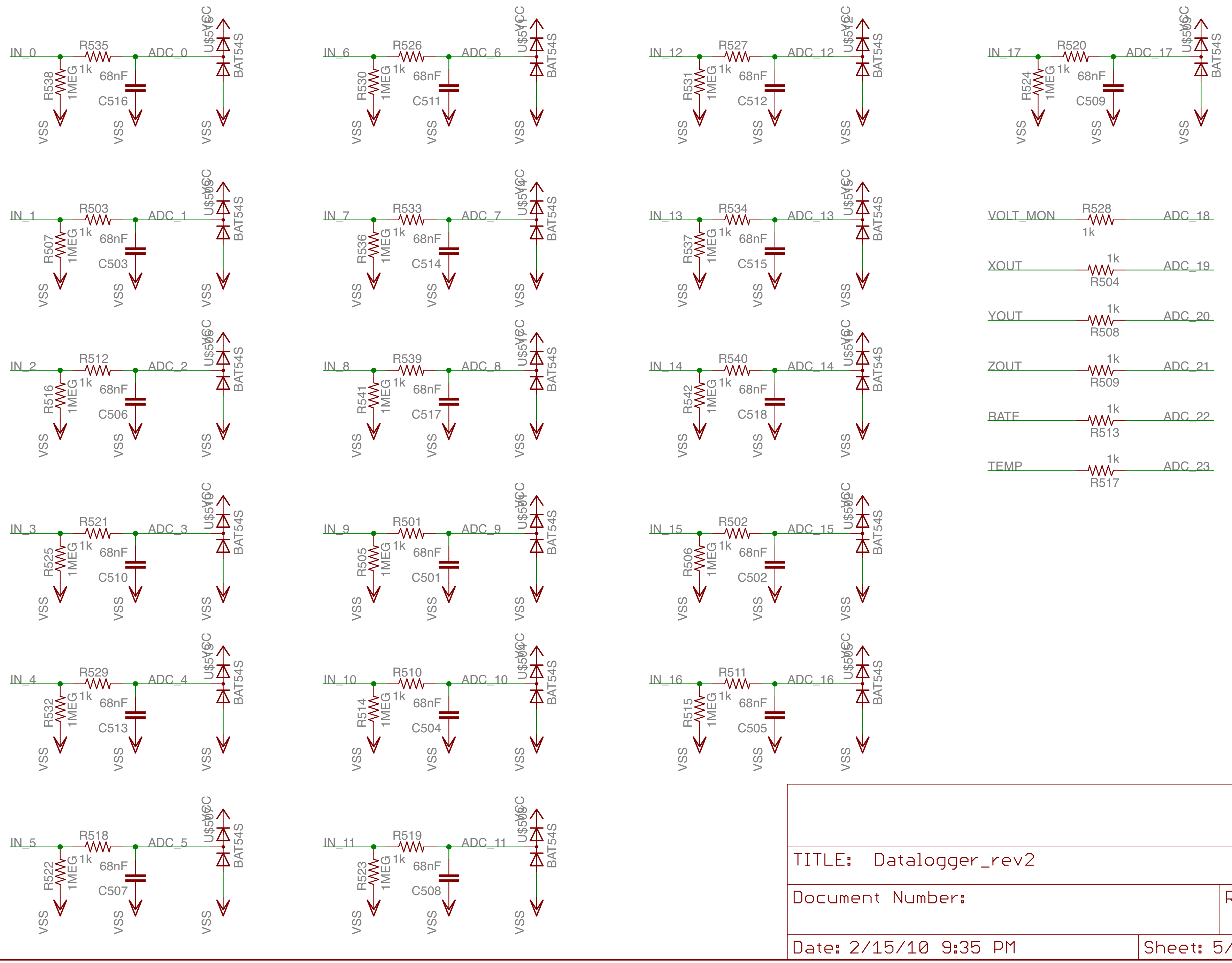
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 4/10

Input Filtering



TITLE: Datalogger_rev2

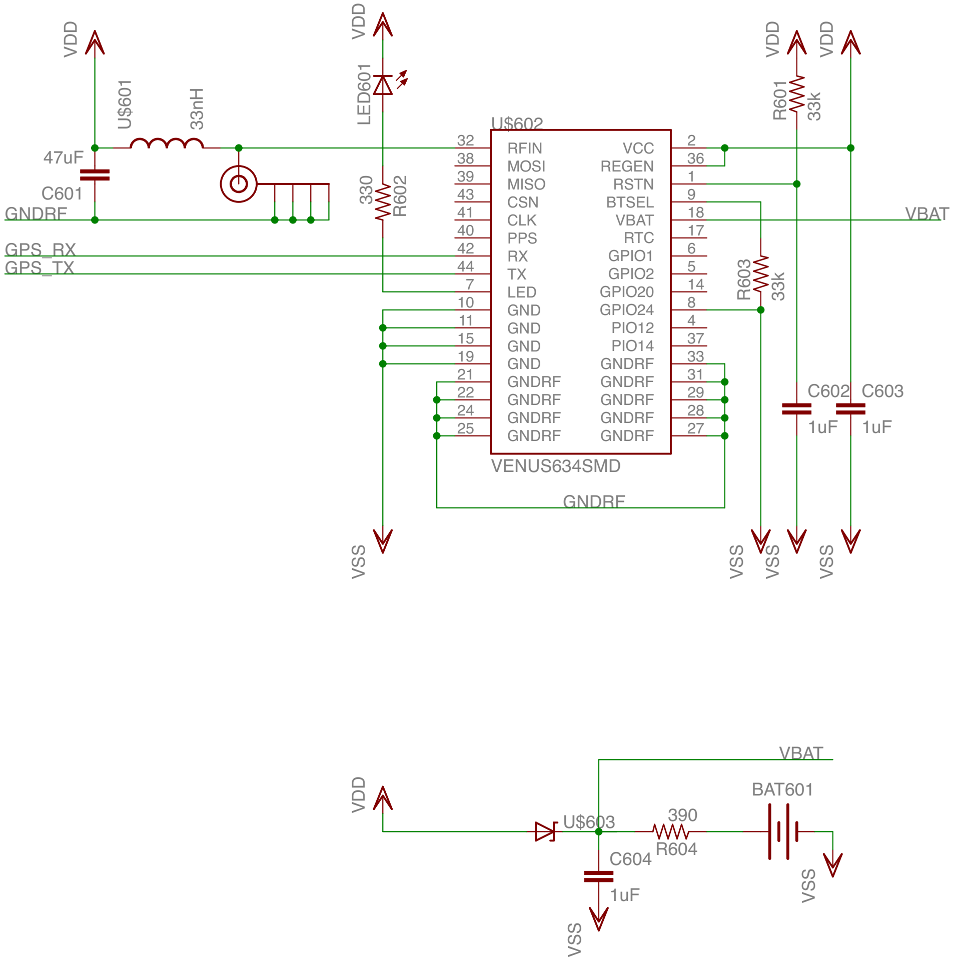
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 5/10

GPS and Battery Backup



TITLE: Datalogger_rev2

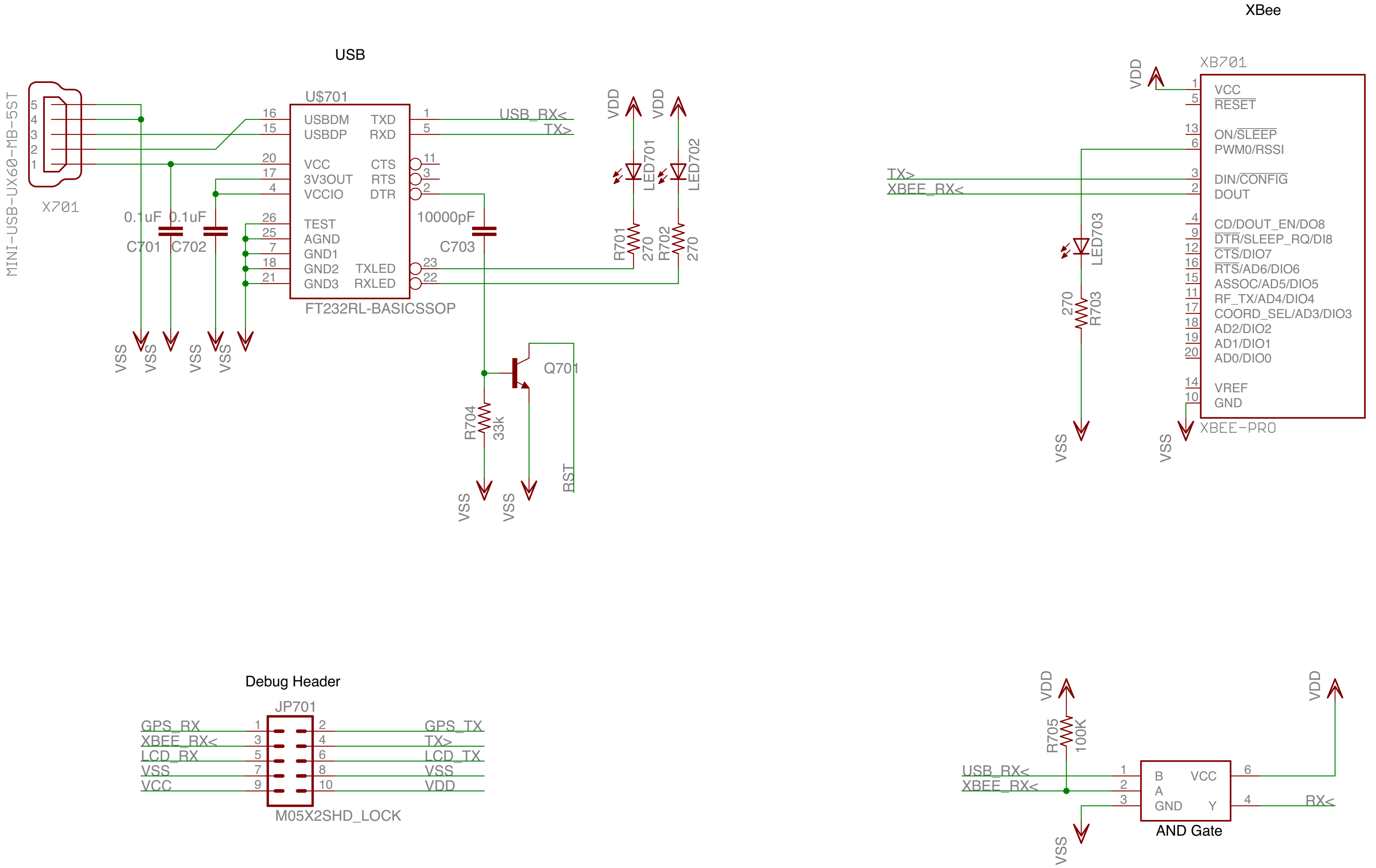
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 6/10

USB, Wireless, and LCD Communication



TITLE: Datalogger_rev2

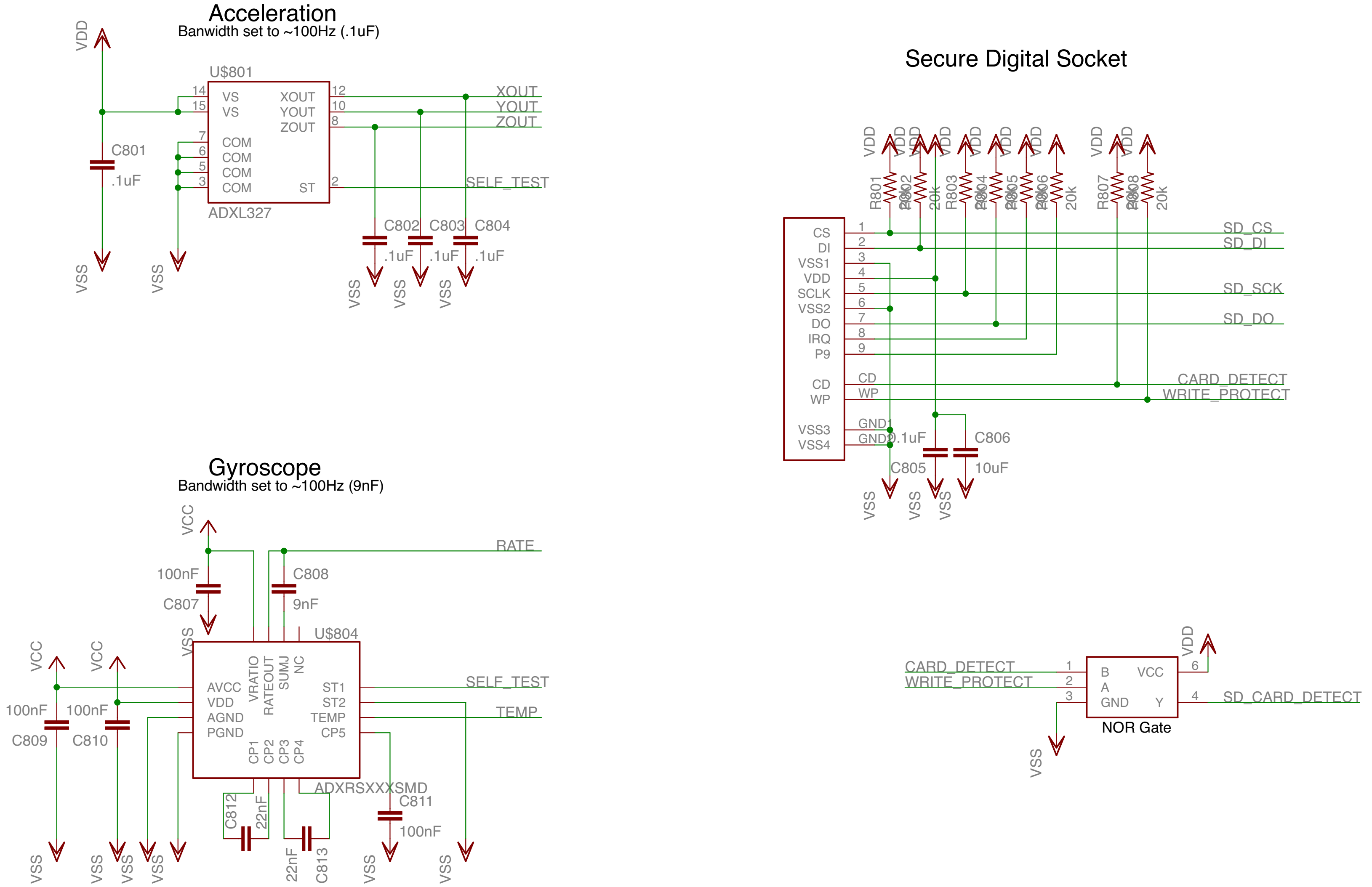
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 7/10

Acceleration and Gyroscope Sensors, and Secure Digital Socket



TITLE: Datalogger_rev2

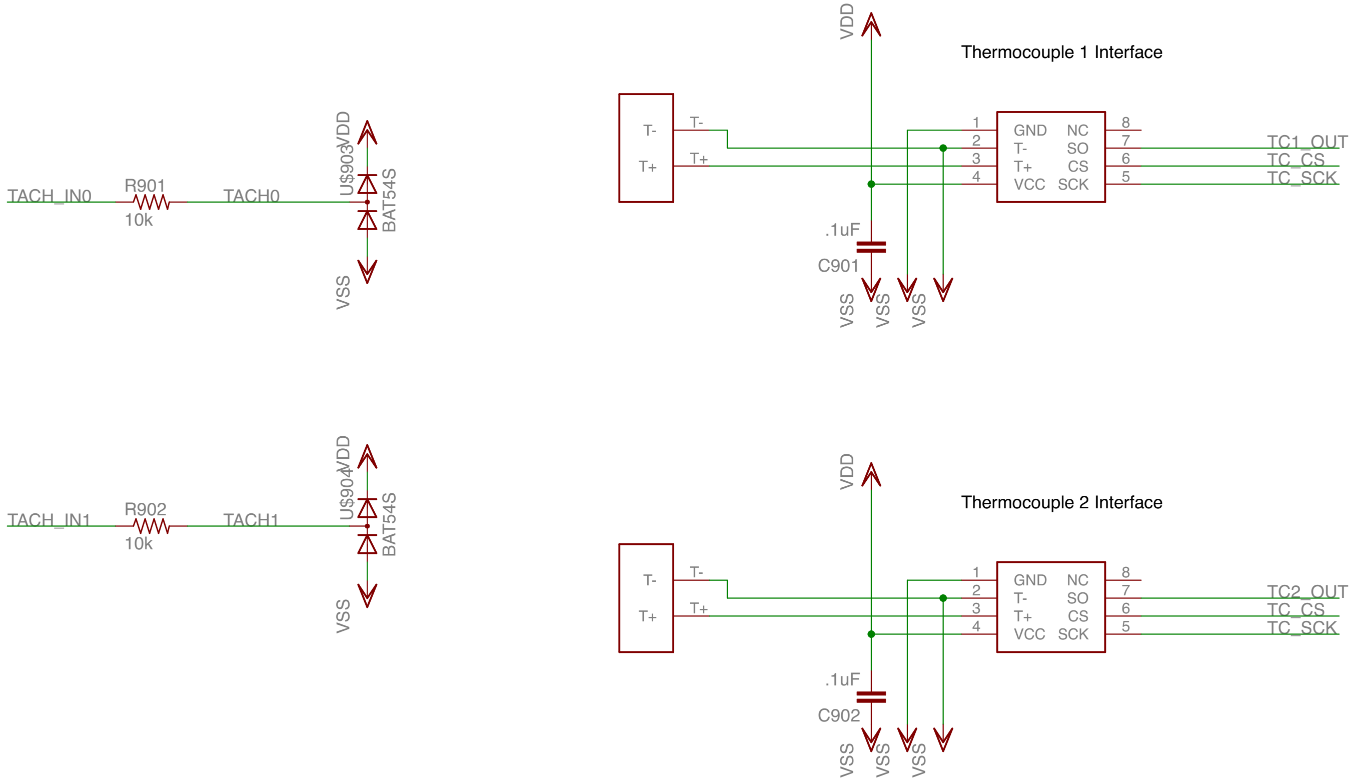
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 8/10

Thermocouple and Tachometer Inputs



TITLE: Datalogger_rev2

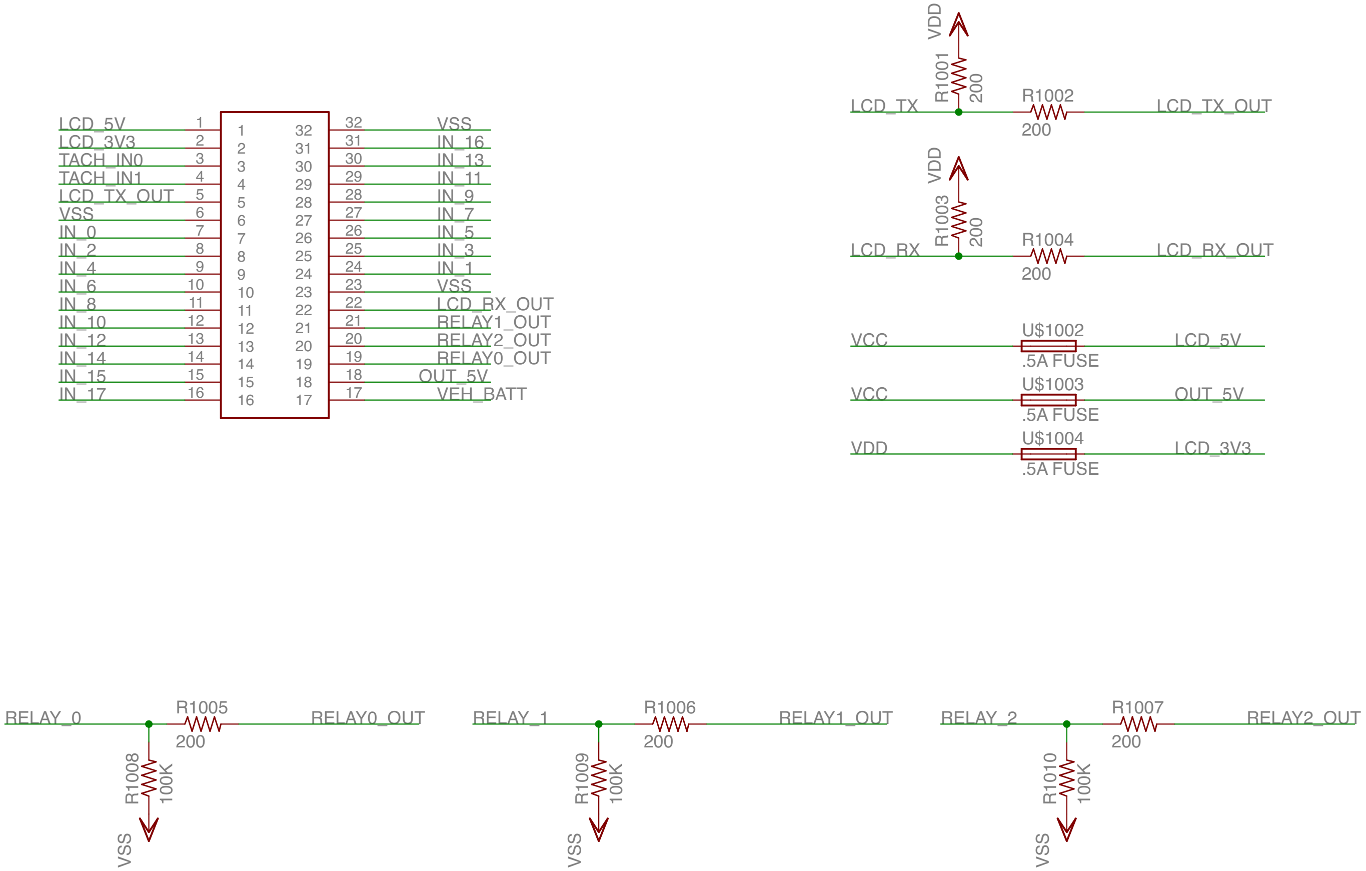
Document Number:

REV:

Date: 2/15/10 9:35 PM

Sheet: 9/10

Connector



TITLE: Datalogger_rev2

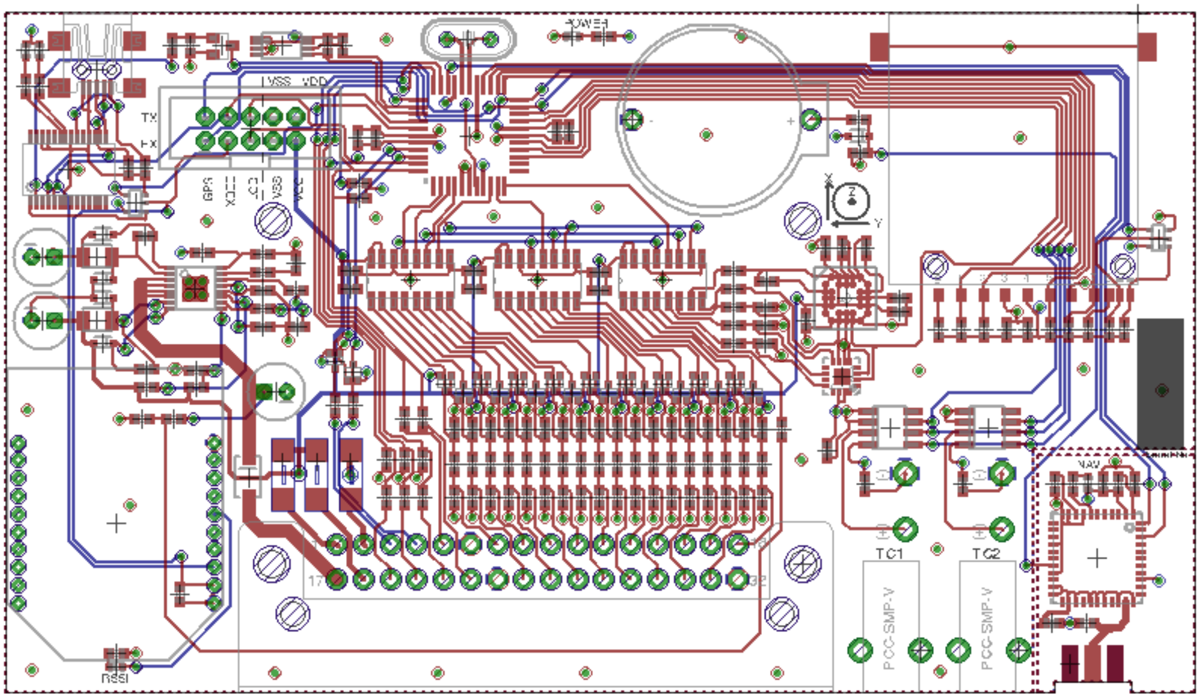
Document Number:

REV:

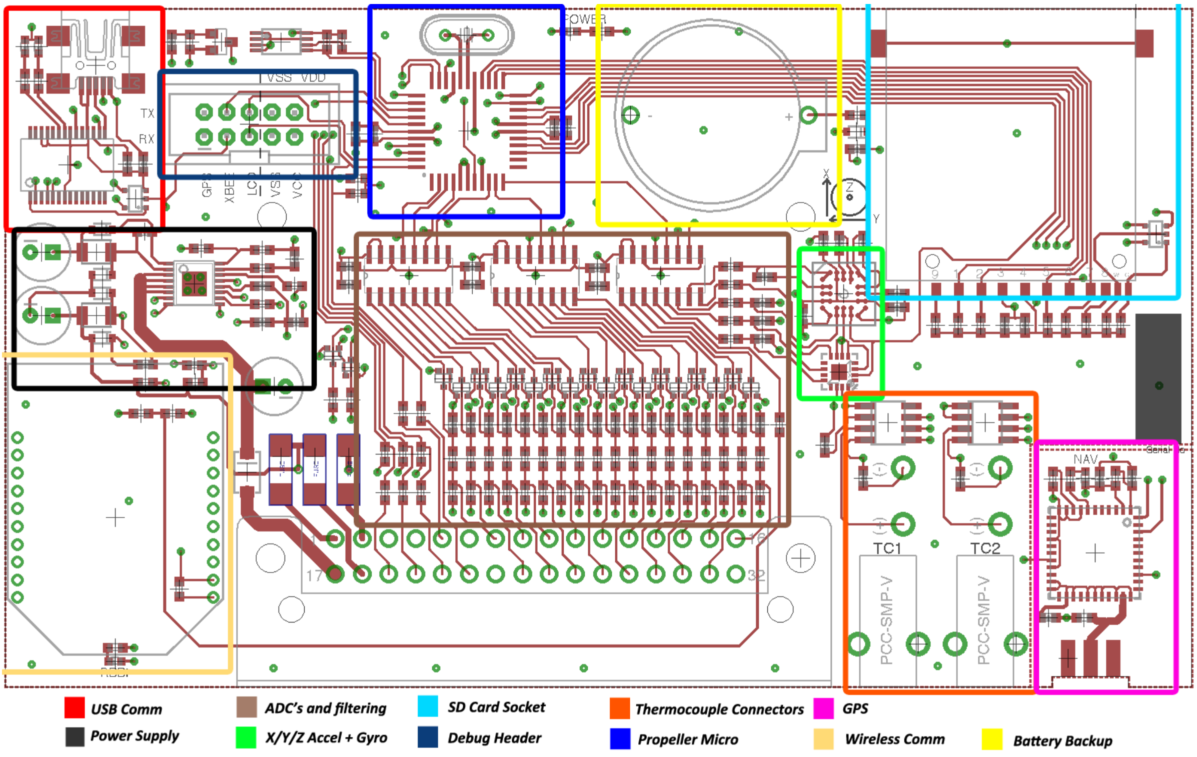
Date: 2/15/10 9:35 PM

Sheet: 10/10

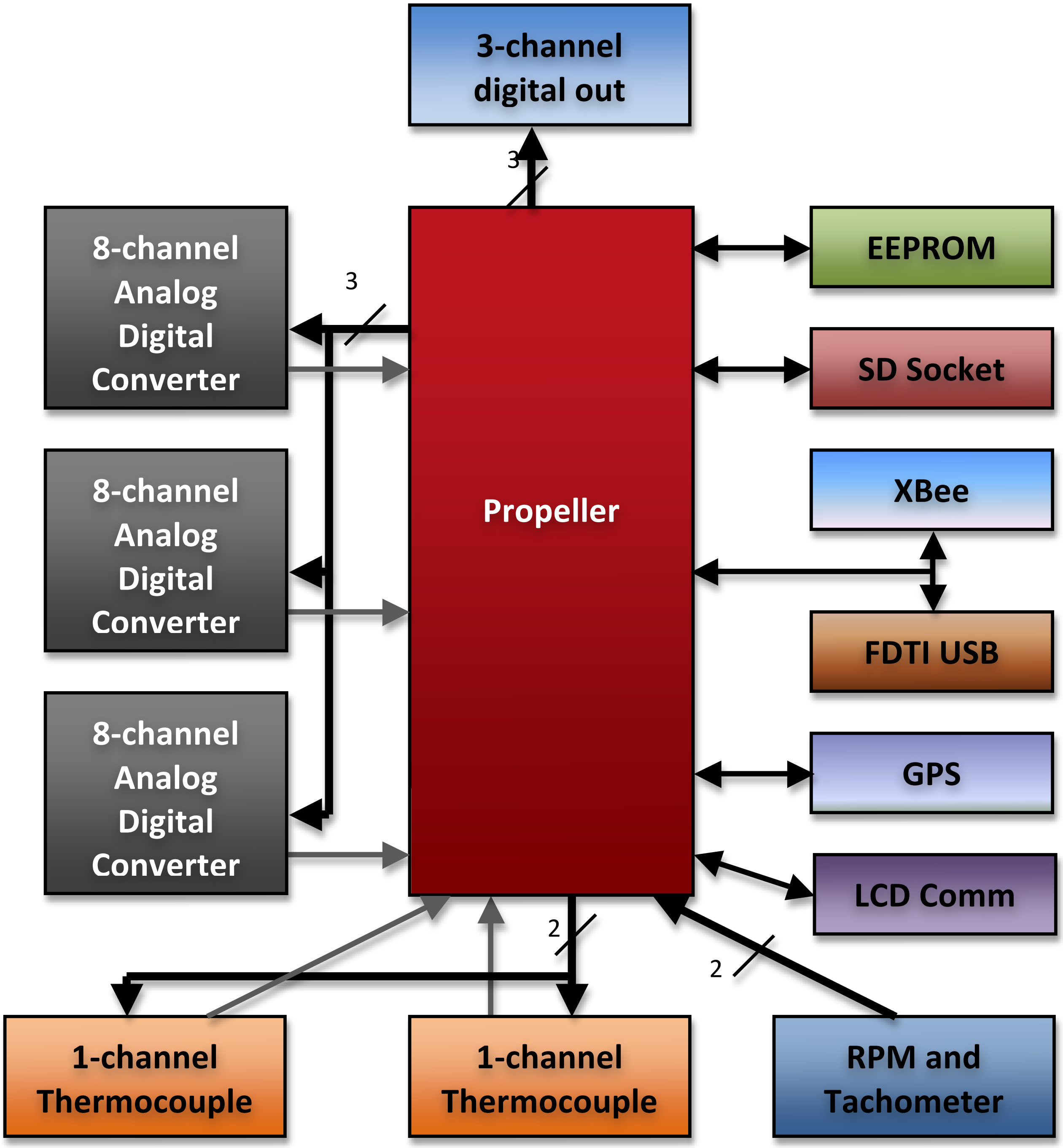
2.2 Board Layout



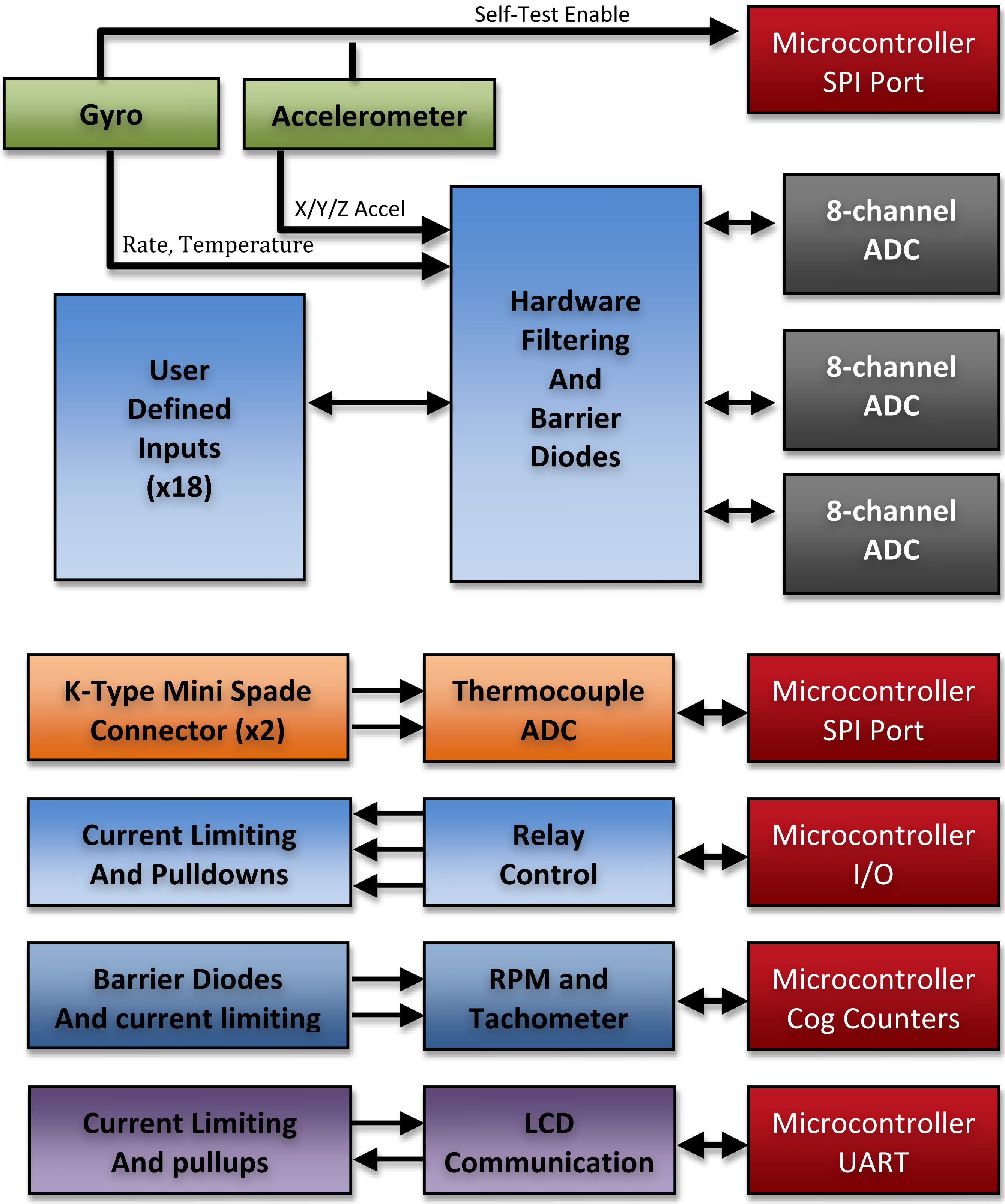
This figure shows the top layer in red, and the bottom layer in blue. The DAQPac is a 4-layer board, measuring 5.25” by 3”. The middle 2 layers are a ground plane and a 3.3-volt power plane. The top and bottom layers also have ground pours on empty areas, connected by vias throughout. The bottom right corner has it’s own isolated ground pour for the GPS receiver. The figure below highlights the different sections of the board.



3.1 Microcontroller Block Diagram



3.2 External I/O Block Diagram



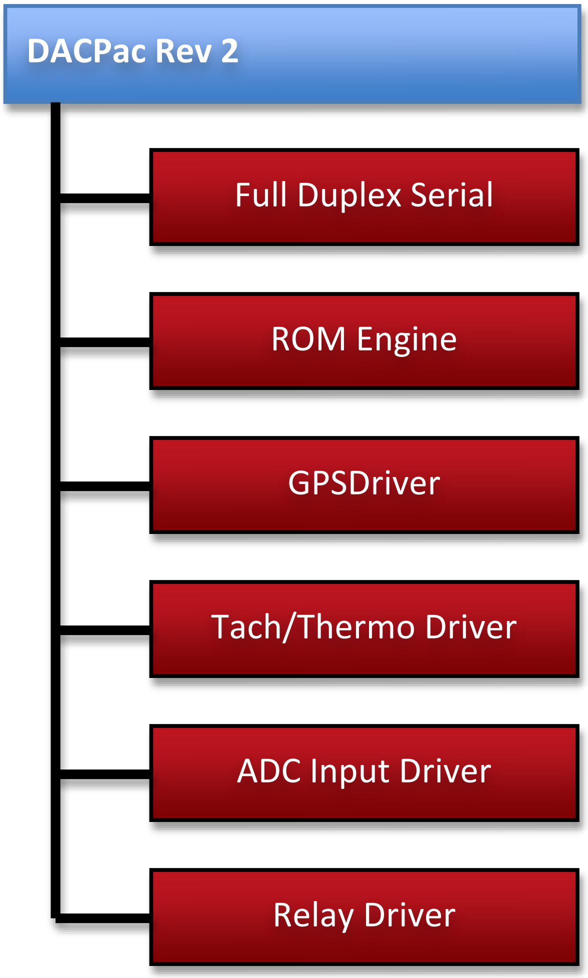
4. Bill Of Materials

DAQPac, Rev 2 BOM (02/15/10)					
Part	**Qty**	**Device**	**Value**	**Package**	**Manufacturer**
C207	1	Capacitor	100pF	0603	-
C209	1	Capacitor	220pF	0603	-
C205, C206	2	Capacitor	1nF	0603	-
C808	1	Capacitor	9nF	0603	-
C812, C813	2	Capacitor	22nF	0603	-
C501, C502, C503, C504, C505, C506, C507, C508, C509, C510, C511, C512, C513, C514, C515, C516, C517, C518	18	Capacitor	68nF	0603	-
C703	1	Capacitor	0.01uF	0603	-
C201, C204, C301, C302, C401, C403, C405, C701, C702, C801, C802, C803, C804, C805, C807, C809, C810, C811, C901, C902	20	Capacitor	0.1uF	0603	-
C602, C603, C604	3	Capacitor	1uF	0603	-
C303, C304, C402, C404, C406, C806	6	Capacitor	10uF	0603	-
C601	1	Capacitor	47uF	0603	-
C208	1	Capacitor	47uF	Radial	-
C202, C203	2	Capacitor	100uF	Radial	-
Q301	1	Crystal	5MHz	HC-49	-
D201	1	Diode	-	SMA	-
D202, D203	2	Diode	-	SOD-323	-
U$501, U$502, U$503, U$504, U$505, U$506, U$507, U$508, U$509, U$510, U$511, U$512, U$513, U$514, U$515, U$516, U$517, U$518, U$903, U$904	20	Diode; Schottky Barrier	-	SC70	-
U$201, U$203, U$603	3	Diode; Zener BZT52	-	SOD-323	-
U$804	1	IC; 150deg/sec Gyro	ADXRS613	BC-32	Analog
U$801	1	IC; 2g X/Y/Z Accelerometer	ADXL327	LFCSP-16	Analog
U$302	1	IC; 512-kbit EEPROM	AT24C512	TSSOP-8	Atmel
U$902, U$906	2	IC; ADC; 1-channel thermocouple; 12-bit	MAX6675ISA	SOIC-8	MAXIM
U$401, U$402, U$403	3	IC; ADC; 8-channel; 10-bit	MCP3008	SOIC-16	Microchip
U$202	1	IC; Dual Switching Regulator	LT1940EFE	TSSOP-16	Linear
U$602	1	IC; GPS Reciever	VENUS634	LGA-44	SkyTraq
U$301	1	IC; Propeller	P8X32A-Q44	QFN-44	Parallax
U$701	1	IC; USB to Serial UART	FT232RL	TSSOP-28	FTDI
U$601	1	Inductor	33nH	0603	-
	1	Inductor	3.3uH	1210	-
	1	Inductor	4.7uH	1210	-
LED201, LED601, LED701, LED701, LED703	5	LED; Green	-	0603	-
U$702	1	Logic; 2-input AND	74LVC1G08GV,125	SC74	-
U$803	1	Logic; 2-input NOR	74AHC1G02GV,125	SC74	-
BAT601	1	Mechanical; Battery Holder	-	Through-Hole	-
U$1001	1	Mechanical; Connector; 32-way	12186041	Through-Hole	Delphi
	1	Mechanical; Connector; SMA Female	-	PCB Edge	-
U$901, U$905	2	Mechanical; Connector; Thermocouple	PCC-SMP-V	Through-Hole	Omega

X701	1	Mechanical; Connector; USB	UX60-MB-5ST	Surface Mount	Hirose
U$1002, U$1003, U$1004	3	Mechanical; Fuse Holder	564 TR-5	Surface Mount	Littlefuse
JP701	1	Mechanical; Header; Shrouded 2x5	-	Through-Hole	-
U$703	2	Mechanical; Header; Xbee	NPPN101BFCN-RC	Through-Hole	Sullins
U$802	1	Mechanical; Secure Digital Socket	SD-RSMT-MQ	Surface Mount	3M
XB701	1	Module; XBee Pro 60mW	XBEE-PRO	Through-Hole	Digi
R1001, R1002, R1003, R1004, R1005, R1006, R1007	7	Resistor	200	0603	-
R211	1	Resistor	270	0603	-
R602	1	Resistor	330	0603	-
R604	1	Resistor	390	0603	-
R501, R502, R503, R504, R508, R509, R510, R511, R512, R513, R517, R518, R519, R520, R521, R526, R527, R528, R529, R533, R534, R535, R539, R540	24	Resistor	1K	0603	-
R205, R207, R208, R210, R301, R302, R901, R902	8	Resistor	10K	0603	-
R201, R701, R702, R703	4	Resistor	16.5k	0603	-
R209, R801, R802, R803, R804, R805, R806, R807, R808	9	Resistor	20K	0603	-
R202	1	Resistor	30.1K	0603	-
R601, R603, R704	3	Resistor	33K	0603	-
R206	1	Resistor	50K	0603	-
R203, R204, R705, R1008, R1009, R1010	3	Resistor	100K	0603	-
R505, R506, R507, R514, R515, R516, R522, R523, R524, R525, R530, R531, R532, R536, R537, R538, R541, R542	18	Resistor	1M	0603	-
Q701	1	Transistor; NPN	-	SOT-23	-

5. Source Code

The code for DACPac is organized by cogs, with the main communication driver acting as the glue to tie them together. Each driver was designed to be self-sustaining, where the driver required no further action from the parent cog after initialization. The main communication behaves as a protocol that could be interacted by manually or by software. Currently, the protocol supports querying data, manually setting outputs and internal flags, pairing wireless transceivers, and reading and writing to the EEPROM.



```
''********************************************
''*       DAQPac Revision 2.0 Software       *
''*         By: Ryan David, 2/23/10          *
''********************************************
''*              Main Cog and                *
''*         Command Interpreter v0.1         *
''*                                          *
''*     - a derivative work based on -       *
''*                                          *
''*    Serial Terminal, bundled with FSRW    *
''*          By: Jonathan Dummer             *
''*                                          *
''*               - and -                    *
''*                                          *
''*            String Engine                 *
''*      Author: Kwabena W. Agyeman          *
''*                                          *
''*    See end of file for terms of use.     *
''********************************************

{-----------------REVISION HISTORY-----------------
 v0.1 - 2/23/2010, first official release
}

CON
  _clkmode = xtal1 + pll16x
   _xinfreq = 5_000_000

  ' command parsing
  cmd_length    = 256
  num_tokens    = 16

  'Pin assignments
  TACH0         = 0
  TACH1         = 1
  LCD_TX        = 2
  LCD_RX        = 3
  UNUSED_0      = 4
  ADC_CLK       = 5
  ADC_0_DOUT    = 6
  ADC_DIN       = 7
  ADC_CS        = 8
  ADC_1_DOUT    = 9
  ADC_2_DOUT    = 10
  UNUSED_1      = 11
  UNUSED_2      = 12
  SELF_TEST     = 13
  TC1_OUT       = 14
  TC2_OUT       = 15
  TC_CS         = 16
  TC_SCK        = 17
  GPS_RX        = 18
  GPS_TX        = 19
  SD_CS         = 20
  SD_DI         = 21
  SD_SCK        = 22
  SD_DO         = 23
  CARD_DETECT   = 24
  RELAY0        = 25
  RELAY1        = 26
  RELAY2        = 27
  I2C_SCL       = 28
  I2C_SDA       = 29
  MAIN_TX       = 30
  MAIN_RX       = 31

  'Baud rates for comm
  GPS_BAUD      = 4800
  MAIN_BAUD     = 57600
  LCD_BAUD      = 57600


OBJ
   term   : "FullDuplexSerial"
   EEPROM : "ROMEngine"
   GPS    : "GPSDriver"
   TT     : "Tach_Thermo_Driver"
   ADC    : "ADC_INPUT_DRIVER"
   Relay  : "RelayDriver"


VAR
  long tokens[num_tokens]
  long tindex
  byte tin[cmd_length]
  byte moduleStatus

  byte hexidecimalCharacters[9]

  long GPSBuff
  long TTBuff

  long chanstate[8]
  long chanval[8]
  long chanmax[8]
  long chanmin[8]

PUB Main | char_in, did_something
  term.start(MAIN_RX, MAIN_TX, 0, MAIN_BAUD)                                    'Start main
comm
  GPSBuff := GPS.start(GPS_RX, GPS_TX, 0, GPS_BAUD, @moduleStatus)              'Start self-
contained GPS comm and parser
  TTBuff := TT.start(TC1_OUT, TC2_OUT, TC_CS, TC_SCK, TACH0, TACH1)             'Start
Thermocouple and Tachometer driver
  Relay.Start(RELAY0, RELAY1, RELAY2)                                           'Start Relay
Driver
  ADC.start_pointed(7, 10, 5, 8, 8, 8, 10, 1, @chanstate, @chanval, @chanmax, @chanmin) 'Start
ADC driver

  tindex := 0
  bytefill( @tin, 0, cmd_length )

  waitcnt(clkfreq/2 + cnt)
  term.str(string(13,"DAQPac 2.0", 13))
  term.tx(">")

  moduleStatus |= %0000_0010                                                    'Set system-
wide status flags
```

```
  Relay.AddCondition(0, TTBuff + 8, 500_000, 0)                                 'Add condition
to trigger relay 0
  Relay.EnableRelays                                                            'Enable output
of relays

  repeat                                                                        'start
interpreting commands
    char_in := term.rx'time( ms )
    did_something := false
    repeat while char_in => 0
      ' we got some input!
      did_something := true
      term.tx( char_in ) ' echo it
      if char_in == $08 ' backspace
        tindex--
        tindex #>= 0
      elseif char_in == $0D ' [Enter] terminates
        char_in := -1
        byte[@tin][tindex] := 0
        tindex := 0 ' reset my character index
        execute_command( @tin ) ' do the whatever!
        term.tx(">")
      else
        byte[@tin][tindex] := char_in
        tindex++
        tindex <#= cmd_length-1
      ' continue with the next character
      char_in := term.rxcheck


PRI execute_command( cmd_str_ptr ) | ntok, tmp, a, eetemp, b
  dirA[23]~~

  ntok := tokenize( cmd_str_ptr, @tokens, num_tokens )
  if ntok < 1
    return


  'Display Version Info
  '-------------------------------------------------------------------------------
  if strcomp( tokens[0], string( "VER" ))
    term.str(string("DAQPac 8.0                  02/03/10", 13))
    term.str(string("------------------------------------", 13))
    term.str(string("Terminal        rev 0.1     02/26/10", 13))
    term.str(string("Comm Driver     rev 1.0     07/24/08", 13))
    term.str(string("GPS Driver      rev 0.1     02/20/10", 13))
    term.str(string("File System     rev 2.6     12/11/09", 13))
    term.str(string("ADC Driver      rev 2.0     11/11/09", 13))
    term.str(string("T/T Driver      rev 0.1     02/23/10", 13))
    term.str(string("Relay Driver    rev 0.1     02/26/10", 13))


  'Display Module Status
  '-------------------------------------------------------------------------------
  elseif strcomp(tokens[0], string("STAT"))
    term.str(string("Module Status                ")) 
    if(moduleStatus & %1000_0000 <> 0)
      term.str(string("Logging",13))
    else
      term.str(string("   Idle",13))
    term.str(string("------------------------------------", 13))

    term.str(string("Power Supply                    "))
    if(moduleStatus & %0001_0000 <> 0)
      term.str(string("   OK",13))
    else
      term.str(string("ERROR",13))

    term.str(string("IMU Bias                        "))
    if(moduleStatus & %0000_1000 <> 0)
      term.str(string("   OK",13))
    else
      term.str(string("ERROR",13))

    term.str(string("GPS Driver                    "))
    if(moduleStatus & %0000_0001 <> 0)
      term.str(string("Running",13))
    else
      term.str(string("    Off",13))

    term.str(string("Comms                         "))
    if(moduleStatus & %0000_0010 <> 0)
      term.str(string("Running",13))
    else
      term.str(string("    Off",13))

    term.str(string("ADC Driver                    "))
    if(moduleStatus & %0000_0100 <> 0)
      term.str(string("Running",13))
    else
      term.str(string("    Off",13))

    term.str(string("Temp Driver                   "))
    if(moduleStatus & %0010_0000 <> 0)
      term.str(string("Running",13))
    else
      term.str(string("    Off",13))

    term.str(string("File System                   "))
    if(moduleStatus & %0100_0000 <> 0)
      term.str(string("Running",13))
    else
      term.str(string("    Off",13))


  'Pair wireless interface
  '-------------------------------------------------------------------------------
  elseif strcomp( tokens[0], string("PAIR"))
    if ntok < 3
      term.str(@ERROR)
    else
      waitcnt(clkfreq*3 + cnt)
      term.str(string("+++"))   'Enter AT mode
      waitcnt(clkfreq + cnt)
      term.str(string("ATDH"))  'Set the upper destination address
      term.str(tokens[1])
      term.tx(13)
      waitcnt(clkfreq + cnt)
      term.str(string("ATDL"))  'Set the lower destination address
      term.str(tokens[2])
      term.tx(13)
      waitcnt(clkfreq + cnt)
      term.str(string("ATCN"))  'Exit AT mode
      term.tx(13)
      'term.str(string("OK"))


  'Set relay control line and logging status
  '-------------------------------------------------------------------------------
  elseif strcomp( tokens[0], string("S"))
    if ntok < 3
      term.str(@ERROR)
    else
      tokens[2] := decimalToNumber(tokens[2]) & 1

      if strcomp(tokens[1], string("R0"))
        outA[RELAY0] := tokens[2]
        term.str(@OK)
      elseif strcomp(tokens[1], string("R1"))
        outA[RELAY1] := tokens[2]
        term.str(@OK)
      elseif strcomp(tokens[1], string("R2"))
        outA[RELAY2] := tokens[2]
        term.str(@OK)
      elseif strcomp(tokens[1], string("LOG"))
        if tokens[2] == 1
          moduleStatus |= %1000_0000
        else
          moduleStatus &= %0111_1111
        term.str(@OK)
      else
        term.str(@ERROR)


  'Query data
  '-------------------------------------------------------------------------------
  elseif strcomp( tokens[0], string("Q"))
    if ntok < 2
      term.str(@ERROR)
    else

      if strcomp(tokens[1], string("R0"))               'Relay 0
        term.dec(inA[RELAY0])
      elseif strcomp(tokens[1], string("R1"))           'Relay 1
        term.dec(inA[RELAY1])
      elseif strcomp(tokens[1], string("R2"))           'Relay 2
        term.dec(inA[RELAY2])
      elseif strcomp(tokens[1], string("LOG"))          'Logging Status
        term.dec(moduleStatus >> 7)
      elseif strcomp(tokens[1], string("A"))            'Analog Inputs
        repeat a FROM 0 TO 7
          term.dec(chanval[a])
          term.tx(",")
      elseif strcomp(tokens[1], string("G"))            'GPS
        GPSSentence
      elseif strcomp(tokens[1], string("SD"))           'SD Card Detect
```

```
      term.dec(inA[CARD_DETECT])
    elseif strcomp(tokens[1], string("T"))              'Thermocouple
      if long[TTBuff] & 1
        term.str(string("N/A"))
      else
        term.dec(long[TTBuff] >> 2)
        term.tx(".")
        term.dec(((long[TTBuff] >> 1) & %11)*25)
      term.tx(",")
      if long[TTBuff+4] & 1
        term.str(string("N/A"))
      else
        term.dec(long[TTBuff+4] >> 2)
        term.tx(".")
        term.dec(((long[TTBuff+4] >> 1) & %11)*25)
    elseif strcomp(tokens[1], string("R"))              'RPM Inputs
      term.dec(long[TTBuff + 8])
      term.tx(",")
      term.dec(long[TTBuff + 13])
    else
      term.str(@ERROR)

  'Read EEPROM
'-------------------------------------------------------------------------------
elseif strcomp( tokens[0], string("READEE"))
  if ntok < 3
    term.str(@ERROR)
  else
    tokens[1] := hexidecimalToNumber(tokens[1]) <# 65534
    tokens[2] := hexidecimalToNumber(tokens[2]) >> 4

    term.str(numberToHexidecimal(tokens[1], 4))
    term.tx(" ")
    repeat tokens[2]
      repeat 2
        repeat 2
          eetemp := EEPROM.readLong(tokens[1])
          tokens[1] += 4

          term.tx(" ")
          term.hex(eetemp & %1111_1111,2 )
          term.tx(" ")
          term.hex((eetemp >> 8) & %1111_1111, 2)
          term.tx(" ")
          term.hex((eetemp >> 16) & %1111_1111, 2)
          term.tx(" ")
          term.hex(eetemp >> 24, 2)
        term.str(string("  "))

      term.tx(13)
      term.str(numberToHexidecimal(tokens[1], 4))
      term.tx(" ")



'Write Byte To EEPROM
'-------------------------------------------------------------------------------
elseif strcomp( tokens[0], string("WRITEEE"))
  if ntok < 3
```

```
      term.str(@ERROR)
    else
      tokens[1] := hexidecimalToNumber(tokens[1]) <# 65534
      tokens[2] := hexidecimalToNumber(tokens[2]) <# 256

      EEPROM.writeByte(tokens[1], tokens[2])
      term.str(@OK)

  'Perform a full hardware reboot
  '--------------------------------------------------------------------------------
  elseif strcomp( tokens[0], string("REBOOT"))
    term.str(string("Rebooting... now!",13))
    waitcnt(clkfreq/8 + cnt)
    reboot

  elseif strcomp( tokens[0], @OK)
    term.str(@OK)

  'Display error
  '--------------------------------------------------------------------------------
  else
    term.str(@ERROR)

  ' end with a CR
  term.tx( $0D )


PUB GPSSentence
  term.tx(long[GPSBuff+8] >> 24)                    'Output time
  term.tx((long[GPSBuff+8] >> 16) & %1111_1111)
  term.tx(":")
  term.tx((long[GPSBuff+8] >> 8) & %1111_1111)
  term.tx(long[GPSBuff+8] & %1111_1111)
  term.tx(":")
  term.tx((long[GPSBuff+12] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+12] >> 8) & %1111_1111)
  term.tx(".")
  term.tx(long[GPSBuff+12] & %1111_1111)

  term.tx(",")

  term.tx((long[GPSBuff+16] >> 16) & %1111_1111)  'Output GPS Quality

  term.tx(",")

  term.tx((long[GPSBuff+48] >> 8) & %1111_1111)  'Output Satellites
  term.tx(long[GPSBuff+48] & %1111_1111)

  term.tx(",")

  term.tx(long[GPSBuff+20] >> 24)                  'Output Latitude
  term.tx((long[GPSBuff+20] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+20] >> 8) & %1111_1111)
  term.tx(long[GPSBuff+20] & %1111_1111)
  term.tx(".")
  term.tx((long[GPSBuff+24] >> 24) & %1111_1111)
  term.tx((long[GPSBuff+24] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+24] >> 8) & %1111_1111)
  term.tx(long[GPSBuff+24] & %1111_1111)
  term.tx(",")

  term.tx((long[GPSBuff+16] >> 8) & %1111_1111)    'Output North Or South

  term.tx(",")

  term.tx(long[GPSBuff+28] >> 24)                  'Output Longitude
  term.tx((long[GPSBuff+28] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+28] >> 8) & %1111_1111)
  term.tx(long[GPSBuff+28] & %1111_1111)
  term.tx((long[GPSBuff+32] >> 24) & %1111_1111)
  term.tx(".")
  term.tx((long[GPSBuff+32] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+32] >> 8) & %1111_1111)
  term.tx(long[GPSBuff+32] & %1111_1111)
  term.tx(long[GPSBuff+36] & %1111_1111)

  term.tx(",")

  term.tx(long[GPSBuff+16] & %1111_1111)           'Output East Or West

  term.tx(",")

  term.tx(long[GPSBuff] >> 24)                     'Output course
  term.tx((long[GPSBuff] >> 16) & %1111_1111)
  term.tx((long[GPSBuff] >> 8) & %1111_1111)
  term.tx(".")
  term.tx(long[GPSBuff] & %1111_1111)

  term.tx(",")

  term.tx(long[GPSBuff+4] >> 24)                   'Output speed
  term.tx((long[GPSBuff+4] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+4] >> 8) & %1111_1111)
  term.tx(".")
  term.tx(long[GPSBuff+4] & %1111_1111)

  term.tx(",")

  term.tx((long[GPSBuff+52] >> 24) & %1111_1111)   'Altitude
  term.tx((long[GPSBuff+52] >> 16) & %1111_1111)
  term.tx((long[GPSBuff+52] >> 8) & %1111_1111)
  term.tx(".")
  term.tx(long[GPSBuff+52] & %1111_1111)
```

```
PRI tokenize( string_ptr, token_ptr, max_tokens ) : found_tokens | slen, was_ws, is_ws, i
  ' go through the string, storing the lead pointer to any
  ' non-whitespace tokens, and zero terminating each token
  slen := strsize( string_ptr )-1
  was_ws := true
  repeat i from 0 to slen
    ' is this important?
    is_ws := byte[string_ptr][i] < $21
    if is_ws
      ' make it a 0, so tokens will be terminated
      byte[string_ptr][i] := 0
    else
      ' well, it's not white-space...can I upper-case it?
```

```
   if (byte[string_ptr][i] > $60) and (byte[string_ptr][i] < $7B)
    byte[string_ptr][i] -= $20
   'it may be interesting
   if was_ws
    'yep, we just switched..store this token
    long[token_ptr][found_tokens] := string_ptr + i
    found_tokens++
    if found_tokens => max_tokens
     return max_tokens
  'and move on
  was_ws := is_ws
 'done, and the return value is in found_tokens


PUB decimalToNumber(characters) | buffer, counter
 buffer := byte[characters]

 counter := (strsize(characters) <# 11)

 repeat while(counter--)
  result *= 10
  result += lookdownz(byte[characters++]: "0".."9")

 if(buffer == "-")
  -result

PUB hexidecimalToNumber(characters) : buffer | counter
 counter := (strsize(characters) <# 8)

 repeat while(counter--)
  buffer <<= 4
  buffer += lookdownz(byte[characters++]: "0".."9", "A".."F")

PUB numberToHexidecimal(number, length)
 repeat result from 7 to 0
  hexidecimalCharacters[result] := lookupz((number & $F): "0".."9", "A".."F")
  number >>= 4

 return @hexidecimalCharacters[(8 - ((length <# 8) #> 0))]

DAT
OK      byte "OK",0
ERROR   byte "ERROR", 0



{{

┌───────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┐
│                                                     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.                         │
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┘
}}
```

```
''********************************************
''*  Full-Duplex Serial Driver v1.2          *
''*  Author: Chip Gracey, Jeff Martin        *
''*  Copyright (c) 2006-2009 Parallax, Inc.  *
''*  See end of file for terms of use.       *
''********************************************

{-----------------REVISION HISTORY-----------------
 v1.2 - 5/7/2009 fixed bug in dec method causing largest negative value (-2,147,483,648) to be output as -0.
 v1.1 - 3/1/2006 first official release.
}


VAR

  long  cog                     'cog flag/id

  long  rx_head                 '9 contiguous longs
  long  rx_tail
  long  tx_head
  long  tx_tail
  long  rx_pin
  long  tx_pin
  long  rxtx_mode
  long  bit_ticks
  long  buffer_ptr

  byte  rx_buffer[16]           'transmit and receive buffers
  byte  tx_buffer[16]


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

'' Start serial driver - starts a cog
'' returns false if no cog available
''
'' mode bit 0 = invert rx
'' mode bit 1 = invert tx
'' mode bit 2 = open-drain/source tx
'' mode bit 3 = ignore tx echo on rx

  stop
  longfill(@rx_head, 0, 4)
  longmove(@rx_pin, @rxpin, 3)
  bit_ticks := clkfreq / baudrate
  buffer_ptr := @rx_buffer
  okay := cog := cognew(@entry, @rx_head) + 1


PUB stop

'' Stop serial driver - frees a cog

  if cog
    cogstop(cog~ - 1)
  longfill(@rx_head, 0, 9)
```

```
PUB rxflush

'' Flush receive buffer

  repeat while rxcheck => 0


PUB rxcheck : rxbyte

'' Check if byte received (never waits)
'' returns -1 if no byte received, $00..$FF if byte

  rxbyte--
  if rx_tail <> rx_head
    rxbyte := rx_buffer[rx_tail]
    rx_tail := (rx_tail + 1) & $F


PUB rxtime(ms) : rxbyte | t

'' Wait ms milliseconds for a byte to be received
'' returns -1 if no byte received, $00..$FF if byte

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


PUB rx : rxbyte

'' Receive byte (may wait for byte)
'' returns $00..$FF

  repeat while (rxbyte := rxcheck) < 0


PUB tx(txbyte)

'' Send byte (may wait for room in buffer)

  repeat until (tx_tail <> (tx_head + 1) & $F)
  tx_buffer[tx_head] := txbyte
  tx_head := (tx_head + 1) & $F

  if rxtx_mode & %1000
    rx


PUB str(stringptr)

'' Send string

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


PUB dec(value) | i, x

'' Print a decimal number
```

```
  x := value == NEGX                                                            'Check for max
negative
  if value < 0
    value := ||(value+x)                                                        'If negative,
make positive; adjust for max negative
    tx("-")                                                                     'and output
sign

  i := 1_000_000_000                                                            'Initialize
divisor

  repeat 10                                                                     'Loop for 10
digits
    if value => i
      tx(value / i + "0" + x*(i == 1))                                          'If non-zero
digit, output digit; adjust for max negative
      value //= i                                                               'and digit
from value
      result~~                                                                  'flag non-zero
found
    elseif result or i == 1
      tx("0")                                                                   'If zero digit
(or only digit) output it
    i /= 10                                                                     'Update divisor


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")


DAT

'***********************************
'* Assembly language serial driver *
'***********************************

                        org
'
'
' Entry
'
entry                   mov     t1,par                'get structure address
                        add     t1,#4 << 2            'skip past heads and tails

                        rdlong  t2,t1                 'get rx_pin
                        mov     rxmask,#1
                        shl     rxmask,t2

                        add     t1,#4                 'get tx_pin
                        rdlong  t2,t1
                        mov     txmask,#1
                        shl     txmask,t2

                        add     t1,#4                 'get rxtx_mode
                        rdlong  rxtxmode,t1

                        add     t1,#4                 'get bit_ticks
                        rdlong  bitticks,t1

                        add     t1,#4                 'get buffer_ptr
                        rdlong  rxbuff,t1
                        mov     txbuff,rxbuff
                        add     txbuff,#16

                        test    rxtxmode,#%100  wz    'init tx pin according to mode
                        test    rxtxmode,#%010  wc
        if_z_ne_c       or      outa,txmask
        if_z            or      dira,txmask

                        mov     txcode,#transmit      'initialize ping-pong multitasking
'
'
' Receive
'
receive                 jmpret  rxcode,txcode         'run a chunk of transmit code, then
return

                        test    rxtxmode,#%001  wz    'wait for start bit on rx pin
                        test    rxmask,ina      wc
        if_z_eq_c       jmp     #receive

                        mov     rxbits,#9             'ready to receive byte
                        mov     rxcnt,bitticks
                        shr     rxcnt,#1
                        add     rxcnt,cnt

:bit                    add     rxcnt,bitticks        'ready next bit period

:wait                   jmpret  rxcode,txcode         'run a chuck of transmit code, then
return

                        mov     t1,rxcnt              'check if bit receive period done
                        sub     t1,cnt
                        cmps    t1,#0           wc
        if_nc           jmp     #:wait

                        test    rxmask,ina      wc    'receive bit on rx pin
                        rcr     rxdata,#1
                        djnz    rxbits,#:bit

                        shr     rxdata,#32-9          'justify and trim received byte
                        and     rxdata,#$FF
                        test    rxtxmode,#%001  wz    'if rx inverted, invert byte
        if_nz           xor     rxdata,#$FF
```

```
                        rdlong  t2,par                ' save received byte and inc head
                        add     t2,rxbuff
                        wrbyte  rxdata,t2
                        sub     t2,rxbuff
                        add     t2,#1
                        and     t2,#$0F
                        wrlong  t2,par

                        jmp     #receive              ' byte done, receive next byte
'
'
' Transmit
'
transmit                jmpret  txcode,rxcode         ' run a chunk of receive code, then return

                        mov     t1,par                ' check for head <> tail
                        add     t1,#2 << 2
                        rdlong  t2,t1
                        add     t1,#1 << 2
                        rdlong  t3,t1
                        cmp     t2,t3           wz
        if_z            jmp     #transmit

                        add     t3,txbuff             ' get byte and inc tail
                        rdbyte  txdata,t3
                        sub     t3,txbuff
                        add     t3,#1
                        and     t3,#$0F
                        wrlong  t3,t1

                        or      txdata,#$100          ' ready byte to transmit
                        shl     txdata,#2
                        or      txdata,#1
                        mov     txbits,#11
                        mov     txcnt,cnt

:bit                    test    rxtxmode,#%100  wz    ' output bit on tx pin according to mode
                        test    rxtxmode,#%010  wc
        if_z_and_c      xor     txdata,#1
                        shr     txdata,#1       wc
        if_z            muxc    outa,txmask
        if_nz           muxnc   dira,txmask
                        add     txcnt,bitticks        ' ready next cnt

:wait                   jmpret  txcode,rxcode         ' run a chunk of receive code, then return

                        mov     t1,txcnt              ' check if bit transmit period done
                        sub     t1,cnt
                        cmps    t1,#0           wc
        if_nc           jmp     #:wait

                        djnz    txbits,#:bit          ' another bit to transmit?

                        jmp     #transmit             ' byte done, transmit next byte
'
'
' Uninitialized data
'
```

```
t1                        res     1
t2                        res     1
t3                        res     1

rxtxmode                  res     1
bitticks                  res     1

rxmask                    res     1
rxbuff                    res     1
rxdata                    res     1
rxbits                    res     1
rxcnt                     res     1
rxcode                    res     1

txmask                    res     1
txbuff                    res     1
txdata                    res     1
txbits                    res     1
txcnt                     res     1
txcode                    res     1

{{

┌───────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┐
│                                                    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.                          │
└───────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┘
}}
```

```
''********************************************
''*             GPS Driver v0.1              *
''*         By: Ryan David, 2/21/10          *
''*                                          *
''*     - a derivative work based on -       *
''*                                          *
''*     Full-Duplex Serial Driver v1.2       *
''*      By: Chip Gracey, Jeff Martin        *
''*                                          *
''*   Creates a single cog to receive and    *
''*    parse the data at the same time.      *
''*  Currently it only parses RMC and GGA,   *
''* but more messages could be easily added. *
''*    Tested at 115200 baud with a 10Hz     *
''*         position update rate             *
''*                                          *
''*    See end of file for terms of use.     *
''********************************************

{-----------------REVISION HISTORY-----------------
 v0.1 - 2/23/2010, first official release.
}

VAR
  long  cog

  long rx_pin
  long rxtx_mode
  long bit_ticks

  long Course
  long Speed
  long TimeStamp[2]
  long Flags
  long Latitude[2]
  long Longitude[3]
  long Date[2]
  long Satellites
  long Altitude

PUB start(rxpin, txpin, mode, baudrate, status)
'' mode bit 0 = invert rx
'' mode bit 1 = invert tx
'' mode bit 2 = open-drain/source tx
'' mode bit 3 = ignore tx echo on rx

  stop
  longmove(@rx_pin, @rxpin, 1)
  bit_ticks := clkfreq / baudrate
  cog := cognew(@entry, @rx_pin)  + 1

  byte[status] |= %0000_0001 'Set module status to running

  return @Course

PUB stop
  if cog
    cogstop(cog~ - 1)
DAT
```

```
                        org

'---------------------------------------------------------------------------------------
' Intialize Serial and GPS Hub pointers                                                -
'---------------------------------------------------------------------------------------
entry                   mov     t1,par                'get structure address

                        rdlong  t2,t1                 'get rx_pin
                        mov     rxmask,#1
                        shl     rxmask,t2

                        add     t1,#4                 'get rxtx_mode
                        rdlong  rxtxmode,t1

                        add     t1,#4                 'get bit_ticks
                        rdlong  bitticks,t1

                        add     t1,#4                 'get Course pointer
                        mov     t3,t1

                        add     t1,#4                 'get Speed pointer
                        mov     t4,t1

                        add     t1,#4                 'get Timestamp[0] pointer
                        mov     t5,t1

                        add     t1,#4                 'get Timestamp[1] pointer
                        mov     t6,t1

                        add     t1,#4                 'get Flags pointer
                        mov     t7,t1

                        add     t1,#4                 'get Latitude[0] pointer
                        mov     t8,t1

                        add     t1,#4                 'get Latitude[1] pointer
                        mov     t9,t1

                        add     t1,#4                 'get Longitude[0] pointer
                        mov     t10,t1

                        add     t1,#4                 'get Longitude[1] pointer
                        mov     t11,t1

                        add     t1,#4                 'get Longitude[2] pointer
                        mov     t12,t1

                        add     t1,#4                 'get Date[0] pointer
                        mov     t13,t1

                        add     t1,#4                 'get Date[1] pointer
                        mov     t14,t1

                        add     t1,#4                 'get Satellites pointer
                        mov     t15,t1

                        add     t1,#4                 'get Altitude pointer
                        mov     t16,t1
```

```
'---------------------------------------------------------------------------------------
' Serial Receive Code                                                                  -
'---------------------------------------------------------------------------------------
receive                 test    rxtxmode,#%001  wz    'wait for start bit on rx pin
                        test    rxmask,ina      wc
        if_z_eq_c       jmp     #receive

                        mov     rxbits,#9             'ready to receive byte
                        mov     rxcnt,bitticks
                        shr     rxcnt,#1
                        add     rxcnt,cnt

:bit                    add     rxcnt,bitticks        'ready next bit period

:wait                   mov     t1,rxcnt              'check if bit receive period done
                        sub     t1,cnt
                        cmps    t1,#0           wc
        if_nc           jmp     #:wait

                        test    rxmask,ina      wc    'receive bit on rx pin
                        rcr     rxdata,#1
                        djnz    rxbits,#:bit

                        shr     rxdata,#32-9          'justify and trim received byte
                        and     rxdata,#$FF
                        test    rxtxmode,#%001  wz    'if rx inverted, invert byte
        if_nz           xor     rxdata,#$FF


'---------------------------------------------------------------------------------------
' GPS Code                                                                             -
'---------------------------------------------------------------------------------------
                        add     sent_pos, #1            'increment position in GPS sentence

                        cmp     rxdata, #$24    wz      'is the recieved byte the start of a
GPS sentence?
        if_nz           jmp     #:det_msgID
                        mov     sent_pos, #0            'Reset position in GPS sentence
                        mov     msgID, #0               'change message ID to undetermined
                        jmp     #receive

:det_msgID              cmp     msgID, #0       wz      'did we already figure out what
message ID this is?
        if_nz           jmp     #process
                        cmp     sent_pos, #4    wc      'check to see if this is the middle
character of a message ID
        if_c            jmp     #receive

:checkG                 cmp     rxdata, #71     wz      'check to see if middle character is '
G'
        if_nz           jmp     #:checkM
                        mov     msgID, #2               'set message ID to 2, GGA
                        jmp     #receive

:checkM                 cmp     rxdata, #77     wz      'check to see if middle character is '
M'
        if_nz           jmp     #receive
                        mov     msgID, #3               'set message ID to 1, RMC
                        jmp     #receive
```

```
'----------------------------- Message Processing Code ---------------------------------
process                 cmp     msgID, #2       wz      'check to see if the message ID is GGA
        if_z            jmp     #:gga                   'jump to GGA processing
                        cmp     msgID, #3       wz      'check to see if the message ID is RMC
        if_z            jmp     #:rmc                   'jump to RMC processing
                        jmp     #receive                'Should never reach this!

:rmc  '-----------------------------------------------------------------------------------
                        cmp     sent_pos, #7    wz      'Jump To appropriate section
        if_z            jmp     #:r7
                        cmp     sent_pos, #8    wz
        if_z            jmp     #:r8
                        cmp     sent_pos, #9    wz
        if_z            jmp     #:r9
                        cmp     sent_pos, #10   wz
        if_z            jmp     #:r10
                        cmp     sent_pos, #11   wz
        if_z            jmp     #:r11
                        cmp     sent_pos, #12   wz
        if_z            jmp     #:r12
                        cmp     sent_pos, #14   wz
        if_z            jmp     #:r14
                        cmp     sent_pos, #20   wz
        if_z            jmp     #:r20
                        cmp     sent_pos, #21   wz
        if_z            jmp     #:r21
                        cmp     sent_pos, #22   wz
        if_z            jmp     #:r22
                        cmp     sent_pos, #23   wz
        if_z            jmp     #:r23
                        cmp     sent_pos, #25   wz
        if_z            jmp     #:r25
                        cmp     sent_pos, #26   wz
        if_z            jmp     #:r26
                        cmp     sent_pos, #27   wz
        if_z            jmp     #:r27
                        cmp     sent_pos, #28   wz
        if_z            jmp     #:r28
                        cmp     sent_pos, #30   wz
        if_z            jmp     #:r30
                        cmp     sent_pos, #32   wz
        if_z            jmp     #:r32
                        cmp     sent_pos, #33   wz
        if_z            jmp     #:r33
                        cmp     sent_pos, #34   wz
        if_z            jmp     #:r34
                        cmp     sent_pos, #35   wz
        if_z            jmp     #:r35
                        cmp     sent_pos, #36   wz
        if_z            jmp     #:r36
                        cmp     sent_pos, #38   wz
        if_z            jmp     #:r38
                        cmp     sent_pos, #39   wz
        if_z            jmp     #:r39
                        cmp     sent_pos, #40   wz
        if_z            jmp     #:r40
                        cmp     sent_pos, #41   wz
        if_z            jmp     #:r41
                        cmp     sent_pos, #43   wz
        if_z            jmp     #:r43
                        cmp     sent_pos, #45   wz
        if_z            jmp     #:r45
                        cmp     sent_pos, #46   wz
        if_z            jmp     #:r46
                        cmp     sent_pos, #47   wz
        if_z            jmp     #:r47
                        cmp     sent_pos, #49   wz
        if_z            jmp     #:r49
                        cmp     sent_pos, #51   wz
        if_z            jmp     #:r51
                        cmp     sent_pos, #52   wz
        if_z            jmp     #:r52
                        cmp     sent_pos, #53   wz
        if_z            jmp     #:r53
                        cmp     sent_pos, #55   wz
        if_z            jmp     #:r55
                        cmp     sent_pos, #57   wz
        if_z            jmp     #:r57
                        cmp     sent_pos, #58   wz
        if_z            jmp     #:r58
                        cmp     sent_pos, #59   wz
        if_z            jmp     #:r59
                        cmp     sent_pos, #60   wz
        if_z            jmp     #:r60
                        cmp     sent_pos, #61   wz
        if_z            jmp     #:r61
                        cmp     sent_pos, #62   wz
        if_z            jmp     #:r62
                        jmp     #receive

:r7                     shl     rxdata, #24             'Hours
                        mov     scratchpad, rxdata
                        jmp     #receive

:r8                     shl     rxdata, #16             'Hours
                        add     scratchpad, rxdata
                        jmp     #receive

:r9                     shl     rxdata, #8              'Minutes
                        add     scratchpad, rxdata
                        jmp     #receive

:r10                    add     scratchpad, rxdata      'Minutes
                        wrlong  scratchpad, t5
                        jmp     #receive

:r11                    shl     rxdata, #16             'Seconds
                        mov     scratchpad, rxdata
                        jmp     #receive

:r12                    shl     rxdata, #8              'Seconds
                        add     scratchpad, rxdata
                        jmp     #receive

:r14                    add     scratchpad, rxdata      'Tenths Of Seconds
                        wrlong  scratchpad, t6
                        jmp     #receive

:r20                    shl     rxdata, #24             'Latitude 0
                        mov     scratchpad, rxdata
                        jmp     #receive

:r21                    shl     rxdata, #16             'Laditude 0
                        add     scratchpad, rxdata
                        jmp     #receive

:r22                    shl     rxdata, #8              'Laditude 0
                        add     scratchpad, rxdata
                        jmp     #receive

:r23                    add     scratchpad, rxdata      'Laditude 0
                        wrlong  scratchpad, t8
                        jmp     #receive

:r25                    shl     rxdata, #24             'Laditude 1
                        mov     scratchpad, rxdata
                        jmp     #receive

:r26                    shl     rxdata, #16             'Laditude 1
                        add     scratchpad, rxdata
                        jmp     #receive

:r27                    shl     rxdata, #8              'Laditude 1
                        add     scratchpad, rxdata
                        jmp     #receive

:r28                    add     scratchpad, rxdata      'Laditude 1
                        wrlong  scratchpad, t9
                        jmp     #receive

:r30                    shl     rxdata, #8             'North Or South
                        mov     sharedpad, rxdata
                        'wrlong  rxdata, t7
                        jmp     #receive

:r32                    shl     rxdata, #24             'Longitude 0
                        mov     scratchpad, rxdata
                        jmp     #receive

:r33                    shl     rxdata, #16             'Longitude 0
                        add     scratchpad, rxdata
                        jmp     #receive

:r34                    shl     rxdata, #8              'Longitude 0
                        add     scratchpad, rxdata
                        jmp     #receive

:r35                    add     scratchpad, rxdata      'Longitude 0
                        wrlong  scratchpad, t10
                        jmp     #receive

:r36                    shl     rxdata, #24             'Longitude 1
                        mov     scratchpad, rxdata
                        jmp     #receive
```

```
:r38                    shl     rxdata, #16             'Longitude 1
                        add     scratchpad, rxdata
                        jmp     #receive

:r39                    shl     rxdata, #8              'Longitude 1
                        add     scratchpad, rxdata
                        jmp     #receive

:r40                    add     scratchpad, rxdata      'Longitude 1
                        wrlong  scratchpad, t11
                        jmp     #receive

:r41                    wrlong  rxdata, t12             'Longitude 2
                        jmp     #receive

:r43                    'shl     rxdata, #8              'East Or West
                        add     sharedpad, rxdata
                        'wrlong  rxdata, t7
                        jmp     #receive

:r45                    shl     rxdata, #24             'Speed
                        mov     scratchpad, rxdata
                        jmp     #receive

:r46                    shl     rxdata, #16             'Speed
                        add     scratchpad, rxdata
                        jmp     #receive

:r47                    shl     rxdata, #8              'Speed
                        add     scratchpad, rxdata
                        jmp     #receive

:r49                    add     scratchpad, rxdata      'Speed
                        wrlong  scratchpad, t4
                        jmp     #receive

:r51                    shl     rxdata, #24             'Course
                        mov     scratchpad, rxdata
                        jmp     #receive

:r52                    shl     rxdata, #16             'Course
                        add     scratchpad, rxdata
                        jmp     #receive

:r53                    shl     rxdata, #8              'Course
                        add     scratchpad, rxdata
                        jmp     #receive

:r55                    add     scratchpad, rxdata      'Course
                        wrlong  scratchpad, t3
                        jmp     #receive

:r57                    shl     rxdata, #24             'Date[0]
                        mov     scratchpad, rxdata
                        jmp     #receive

:r58                    shl     rxdata, #16             'Date[0]
                        add     scratchpad, rxdata
                        jmp     #receive
```

```
:r59                     shl     rxdata, #8             'Date[0]
                         add     scratchpad, rxdata
                         jmp     #receive

:r60                     add     scratchpad, rxdata     'Date[0]
                         wrlong  scratchpad, t13
                         jmp     #receive

:r61                     shl     rxdata, #8             'Date[1]
                         mov     scratchpad, rxdata
                         jmp     #receive

:r62                     add     scratchpad, rxdata     'Date[1]
                         wrlong  scratchpad, t14
                         jmp     #receive


:gga    '-----------------------------------------------------------------------------
                         cmp     sent_pos, #43    wz    'Jump to appropriate section
        if_z             jmp     #:g43
                         cmp     sent_pos, #45    wz
        if_z             jmp     #:g45
                         cmp     sent_pos, #46    wz
        if_z             jmp     #:g46
                         cmp     sent_pos, #52    wz
        if_z             jmp     #:g52
                         cmp     sent_pos, #53    wz
        if_z             jmp     #:g53
                         cmp     sent_pos, #54    wz
        if_z             jmp     #:g54
                         cmp     sent_pos, #56    wz
        if_z             jmp     #:g56
                         jmp     #receive

:g43                     shl     rxdata, #16            'Fix Type
                         add     rxdata, sharedpad
                         wrlong  rxdata, t7
                         jmp     #receive

:g45                     shl     rxdata, #8             'Satellites
                         mov     scratchpad, rxdata
                         jmp     #receive

:g46                     add     scratchpad, rxdata     'Satellites
                         wrlong  scratchpad, t15
                         jmp     #receive

:g52                     shl     rxdata, #24            'Altitude
                         mov     scratchpad, rxdata
                         jmp     #receive

:g53                     shl     rxdata, #16            'Altitude
                         add     scratchpad, rxdata
                         jmp     #receive

:g54                     shl     rxdata, #8             'Altitude
                         add     scratchpad, rxdata
                         jmp     #receive
```

```
:g56                    add     scratchpad, rxdata      'Altitude
                        wrlong  scratchpad, t16
                        jmp     #receive


'-----------------------------------------------------------------------------------------
' Variables                                                                              -
'-----------------------------------------------------------------------------------------
t1                      res     1
t2                      res     1

t3                      res     1 'Pointer for Course
t4                      res     1 'Pointer for Speed

t5                      res     1 'Pointer for Timestamp[0]
t6                      res     1 'Pointer for Timestamp[1]
t7                      res     1 'Pointer for Flags
t8                      res     1 'Pointer for Latitude[0]
t9                      res     1 'Pointer for Latitude[1]

t10                     res     1 'Pointer for Longitude[0]
t11                     res     1 'Pointer for Longitude[1]
t12                     res     1 'Pointer for Longitude[2]

t13                     res     1 'Pointer for Date[0]
t14                     res     1 'Pointer for Date[1]

t15                     res     1 'Pointer for Satellites

t16                     res     1 'Pointer for Altitude

rxtxmode                res     1
bitticks                res     1

rxmask                  res     1
rxdata                  res     1
rxbits                  res     1
rxcnt                   res     1
rxcode                  res     1

scratchpad              res     1
sharedpad               res     1

sent_pos                long    0 'Current Position In GPS sentence
msgID                   long    0 'Determining = 0, VTG = 1. GGA = 2, RMC = 3, GSA = 4


fit

{{

┌───────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────┐
│                                                    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.                          │
└─────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────┘
}}
```

```
''********************************************
''*   Tach Signal and Thermocouple Reader    *
''*         By: Ryan David, 2/25/10          *
''*                                          *
''*  Uses one cog to read two tach signals,  *
''*  and poll two MAX6675 thermocouple ADCs  *
''*                                          *
''*    See end of file for terms of use.     *
''********************************************

{-----------------REVISION HISTORY-----------------
 v0.1 - 2/25/2010, first official release.
}

CON
   _clkmode = xtal1 + pll16x
   _xinfreq = 5_000_000

VAR
  Long cog
  Long Pins[6]
  Long Temperature[2]
  Long RPM[2]

PUB Start(T1_OUT, T2_OUT, T_CS, T_SCK, RPM0, RPM1)
  Pins[0] := T1_OUT
  Pins[1] := T2_OUT
  Pins[2] := T_CS
  Pins[3] := T_SCK
  Pins[4] := RPM0
  Pins[5] := RPM1

  cog := cognew(@entry, @Pins) + 1

  return @Temperature[0]

DAT
                        org

entry                   mov     t1, par                 'set up T1 mask
                        rdlong  t2, t1
                        mov     T1_mask, #1
                        shl     T1_mask, t2

                        add     t1, #4                  'set up T2 mask
                        rdlong  t2, t1
                        mov     T2_mask, #1
                        shl     T2_mask, t2

                        add     t1, #4                  'set up CS mask
                        rdlong  t2, t1
                        mov     CS_mask, #1
                        shl     CS_mask, t2

                        add     t1, #4                  'set up SCK mas
                        rdlong  t2, t1
                        mov     SCK_mask, #1
                        shl     SCK_mask, t2
                        add     t1, #4                  'set up counter A
                        rdlong  t2, t1
                        add     ctra_, t2
                        mov     ctra, ctra_
                        mov     frqa, #1

                        add     t1, #4                  'set up counter B
                        rdlong  t2, t1
                        add     ctrb_, t2
                        mov     ctrb, ctrb_
                        mov     frqb, #1

                        add     t1, #4
                        mov     t3, t1                  'get Temperature 1 result pointer

                        add     t1, #4
                        mov     t4, t1                  'get Temperature 2 result pointer

                        add     t1, #4
                        mov     t5, t1                  'get RPM1 result pointer

                        add     t1, #4
                        mov     t6, t1                  'get RPM2 result pointer


                        or      outa, CS_mask           'set CS to high
                        or      outa, SCK_mask          'set SCK to high

                        or      dira, CS_mask           'set CS and SCK to outputs
                        or      dira, SCK_mask

                        mov     overallwait, cnt
                        add     overallwait, tenthsec


conversion              mov     wait, cnt
                        add     wait, hundredclk

                        mov     t1scratchpad, #0
                        mov     t2scratchpad, #0

                        andn    outa, CS_mask           'set CS to low
                        waitcnt wait, hundredclk

                        mov     bitpos, #14

:loop                   andn    outa, SCK_mask          'set SCK to low
                        waitcnt wait, hundredclk

                        test    T1_mask, ina    wc      'grab bit for T1
                        rcl     t1scratchpad, #1

                        test    T2_mask, ina    wc      'grab bit for T1
                        rcl     t2scratchpad, #1

                        or      outa, SCK_mask          'set SCK to high
                        waitcnt wait, hundredclk
                        djnz    bitpos, #:loop
                        or      outa, CS_mask           'set CS to high

                        wrlong  t1scratchpad, t3        'write out results
                        wrlong  t2scratchpad, t4

                        mov     bitpos, #1              'reuse

:counters               waitcnt overallwait, tenthsec   'go through this time delay twice

                        mov     rpm1scratchpad, phsa    'read counters
                        mov     rpm2scratchpad, phsb
                        mov     phsa, #0                'clear counters
                        mov     phsb, #0

                        wrlong  rpm1scratchpad, t5      'write out results
                        wrlong  rpm2scratchpad, t6

                        djnz    bitpos, #:counters      'repeat reading the counters once

                        jmp     #conversion




tenthsec                long    8_000_000              'quarter second to start conversion
hundredclk              long    100

ctra_                   long    %01110 << 26            'NEG EDGE mode
ctrb_                   long    %01110 << 26            'NEG EDGE mode

T1_mask                 res     1
T2_mask                 res     1
CS_mask                 res     1
SCK_mask                res     1

t1                      res     1                       'main pointer
t2                      res     1                       'temporary pointer
t3                      res     1                       'pointer for temperature 1 result
t4                      res     1                       'pointer for temperature 2 result
t5                      res     1                       'pointer for RPM1 result
t6                      res     1                       'pointer for RPM2 result

t1scratchpad            res     1
t2scratchpad            res     1
rpm1scratchpad          res     1
rpm2scratchpad          res     1
bitpos                  res     1
wait                    res     1
overallwait             res     1

fit

{{

┌────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┘
│                                                   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.                          │
└────────────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────────┘
}}
```

```
''*******************************************
''*               Relay Driver              *
''*          By: Ryan David, 2/26/10        *
''*                                         *
''*    Control up to three relays based on  *
''*   the data being collected and specified *
''*               conditions.               *
''*                                         *
''*     See end of file for terms of use.   *
''*******************************************

{-----------------REVISION HISTORY-----------------
 v0.1 - 2/23/2010, first official release.
}

CON

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

VAR
  long R0
  long R1
  long R2
  long Conditions[9]
  long Enable

  long cog
  long stack[100]

PUB Start(RELAY0, RELAY1, RELAY2)
  DisableRelays
  R0 := RELAY0
  R1 := RELAY1
  R2 := RELAY2

  cognew(Main, @stack)

PUB AddCondition(relay, signal, hi, lo) | a
  relay := 0 <# relay <# 2

  Conditions[(relay*3) + 0] := signal
  Conditions[(relay*3) + 1] := hi
  Conditions[(relay*3) + 2] := lo

PUB EnableRelays
  Enable := 1

PUB DisableRelays
  Enable := 0

PUB stop
  if cog
    cogstop(cog~ - 1)

PUB Main
  outA[R0] := 0
  outA[R1] := 0
  outA[R2] := 0
  dirA[R0]~~
  dirA[R1]~~
  dirA[R2]~~

  repeat
    if Enable & 1
      if (long[Conditions[0]] < Conditions[1]) & (long[Conditions[0]] > Conditions[2])
        outA[R0] := 1
      else
        outA[R0] := 0

      if (long[Conditions[3]] < Conditions[4]) & (long[Conditions[3]] > Conditions[5])
        outA[R1] := 1
      else
        outA[R1] := 0

      if (long[Conditions[6]] < Conditions[7]) & (long[Conditions[6]] > Conditions[8])
        outA[R2] := 1
      else
        outA[R2] := 0
    else
      outA[R0] := 0
      outA[R1] := 0
      outA[R2] := 0


{{
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┐
│                                                   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.                         │
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┘
}}
```

```
''*********************************************
''*  ADC Input Driver v2.0                    *
''*  Supports 8- to 16-bit, up to 8 channels  *
''*  Designed for the MCP3X0X series of ADCs  *
''*  Author: Brandon Nimon                    *
''*  Created: 16 July, 2009                   *
''*  Copyright (c) 2009 Parallax, Inc.        *
''*  See end of file for terms of use.        *
''***********************************************************************************
''* Driver features include support for Microchip ADCs with anywhere between 1 and  *
''* 8 ADC input channels and resolutions of 16 bits or less. Both single-ended and  *
''* differential modes are supported. The driver adds support for frequency reading *
''* on each of the channels and allows the ADC to react as a programmable Schmitt   *
''* trigger. Also available is gathering of maximum and minimum or average values   *
''* of the channels over time. The driver can wait for a channel to achieve a       *
''* specific state, or can even be put into standby to save power. Channel values   *
''* and states are available to multiple objects if the driver is supplied with     *
''* variables to store the information.                                             *
''*                                                                                 *
''* A channel is considered "high" only after the channel's value is above the high *
''* threshold. It is considered "low" when the value is below or equal to low       *
''* threshold. Thus making 0 a valid value for both high and low threshold, but     *
''* (for a 12-bit ADC) 4095 would not be a valid value as the "high" state would    *
''* never be achieved (4094 would be valid).                                        *
''*                                                                                 *
''* The waithigh, waitlow, getfreq, and average routines don't start until the      *
''* program cycles around to the channel that has been selected. Then the ADC       *
''* samples will be completely dedicated to the selected task (all other channels'  *
''* values are ignored).                                                            *
''*                                                                                 *
''* Due to channel sample speed changing between the different routines, the output *
''* values may vary slightly (read the ADC's datasheet for more information). For   *
''* example: during testing, displaying max/min values on a consistent input gave a *
''* max value of 3712 and a minimum of 3711, but when getting the average of the    *
''* same channel, it read 3702. This is due to the fact that the channel was being  *
''* sampled eight times more often during the average routine than during the       *
''* normal operation. This results will vary based on number of channels being      *
''* scanned.                                                                        *
''*                                                                                 *
''* For many of the routines with a watchdog value, zero can be put in its place to *
''* disable it. This will make the PASM cog attempt to fulfill the task             *
''* indefinitely. The controlling cog will wait indefinitely for the PASM cog to    *
''* complete. Normally, the watchdog value is the maximum amount of time to wait in *
''* milliseconds (1/1000th of a second).                                            *
''*                                                                                 *
''* Example wiring would be as follows:                                             *
''*               R1                                                                *
''*     ADC>─┳──┳─⋀⋀─>input                                                         *
''*       C1 ╪  ≷R2                                                                 *
''*          ⏚  ⏚                                                                   *
''* R1: 10K (high impedance input is helpful, 10K should be the minimum)            *
''* R2: 100K (this effectively creates a voltage divider, but also drives input to  *
''*     zero volts when not in use)                                                 *
''* C1: 0.01µF (10000pF is the about the maximum you would want to use. The         *
''* capacitor reduces jitter or spikes but also reduces resolution).                *
''***********************************************************************************
''
'' Note: single-channel ADCs will always use slow shift method.
''
```

```
'' Updates:
''      1.1 (28 July, 2009):
''              Added alternate start method: start_pointed. This allows for multiple objects
to access
''                current channel value, state, and max/min. The parent object supplies 4 8-
long
''                blocks which are used instead of this object's blocks.
''              Shifting in and out has been sped up. Fewer instructions are used per bit.
''      1.2 (29 July, 2009):
''              Created a "fast" and "slow" method of shifting in and out. This allows the
driver to
''                function closer to the ADC's maximum sample rate at 5V. For 12-bit ADCs, it
still runs
''                faster than is specified in the MCP32XX datasheet, almost all ADCs should be
able to
''                run at the provided speeds. Clock speeds of 80MHz or faster uses the "slow"
method,
''                while slower speeds will use the "fast" method.
''              Updated documentation with tables and better descriptions of sample speed.
''      1.3 (30 July, 2009):
''              Fixed freeze when standby_disable was called when not in standby.
''              Fixed problem where waitlow would always return 0 (whether it timed out or not)
.
''              Renamed curval to getval for uniformity in method names.
''              Wait methods can now be put into a mode to ignore or read current state before
the
''                driver actually starts waiting.
''      1.31 (3 August, 2009):
''              Add lines of code to make sure threshold values are valid.
''      2.0 (22 October, 2009):
''              Added support for 2 and 1 channel ADCs (now supports 8, 4, 2, and 1 channel
ADCs).
''              Added parameters to main method calls to allow for multiple types of ADCs in
the same
''                program (specific varues are parameters instead of constants).
''              Period meassuring for frequency method is now averaged (in PASM)
''              Slight optimizations in SPIN coding.
''      2.01 (5 November, 2009(
''              Added wait for driver to fully initialize (so commands don't get sent before
driver is
''                ready).
''
'' Future additions:

''      Add ability to alter each channel's threshold separately (may slow down the driver too
much).
''      Instead of limiting the total number of channels, enable/disable the channels based on
a bit of
''        a byte value (also make it so the channels could be changed on the fly).
''      Reduce the variable space (combine variables, use 4 longs instead of 8, etc.).

''
```

```
CON

  default_low_threshold  = 100  ' default "low" threshold value
 (min: 0, max: 2^bits_s_in - 2)
  default_high_threshold = 500  ' default "high" threshold value
 (min: 0, max: 2^bits_s_in - 2)
```

```
OBJ

VAR

  BYTE cogon, cog
  BYTE in_standby
  LONG timescale

  ''===[ 49 longs, DO NOT alter order! ]===
  LONG tx_pin
  LONG rx_pin
  LONG ck_pin
  LONG cs_pin
  LONG adcch
  LONG channels
  LONG bits_s_in
  LONG mode
  LONG par1
  LONG par2
  LONG ptr_start
  LONG fastslow
  LONG done
  LONG channel
  LONG duration
  LONG retval
  LONG count
  LONG command
  LONG chanstate[8]
  LONG chanval[8]
  LONG chanmax[8]
  LONG chanmin[8]


PUB start (DTPin, INPin, ClkPin, RSPin, ChCount, ActChannels, BitCount, SingDiff)
'' Starts driver.
'' DTPin: outgoing (from µController) serial data pin (0-31); ignored if 1-channel ADC
'' INPin: incoming (to µController) serial data pin (0-31)
'' ClkPin: serial clock pin (0-31)
'' RSPin: reset, CS pin (0-31)
'' ChCount: Number of channels on ADC (3208/3008 = 8; 3204/3004 = 4; 3202/3002 = 2; 3201/3001 =
 1)
'' ActChannels: number of channels to scan (1-8)
'' BitCount: number of bits the ADC outputs (2-16)
'' SingDiff: ADC single or differential mode (1 single, 0 differential)


'=====[ SETTINGS ]==============
  par1 := default_low_threshold ' default "low" threshold value
 (min: 0, max: 2^bits_s_in - 2)
  par2 := default_high_threshold' default "high" threshold value
 (min: 0, max: 2^bits_s_in - 2)
'==============================

  stop
  longfill(@done, 0, 38)
  longmove(@tx_pin, @DTPin, 8)
  ptr_start := 0
  in_standby := 0
  timescale := clkfreq / 1000   ' milliseconds -- could be changed to any scale

  IF (((BitCount > 10 AND clkfreq => 80_000_000) OR (BitCount > 12 AND clkfreq => 64_000_000))
OR ChCount == 1)
    fastslow := 0               ' slow mode

  ELSE
    fastslow := -1              ' fast mode

  cogon := (cog := cognew(@entry, @tx_pin))

  waitcnt(7000 + cnt)           ' wait for driver to fully initialize

  RETURN cogon

PUB start_pointed (DTPin, INPin, ClkPin, RSPin, ChCount, ActChannels, BitCount, SingDiff,
chanstate_ptr, chanval_ptr, chanmax_ptr, chanmin_ptr)
'' Starts the driver, but with 4 supplied 8-long blocks.
'' DTPin: outgoing (from µController) serial data pin (0-31); ignored if 1-channel ADC
'' INPin: incoming (to µController) serial data pin (0-31)
'' ClkPin: serial clock pin (0-31)
'' RSPin: reset, CS pin (0-31)
'' ChCount: Number of channels on ADC (3208/3008 = 8; 3204/3004 = 4; 3202/3002 = 2; 3201/3001 =
 1)
'' ActChannels: number of channels to scan (1-8)
'' BitCount: number of bits the ADC outputs (2-16)
'' SingDiff: ADC single or differential mode (1 single, 0 differential)
''
'' Note: This type of start grants the ability to access channels states and values in a
faster method (e.g.
''       adcstate[7] -- similar to ina[7]). It also allows for multiple objects to get the
same information from
''       this object. Of course, only the object that started this driver has access to the
normal functions
''       (threshold, freq, average, etc.).
''       If this method is used to start the driver, the getmax, getmin, getstate, and curval
functions will not
''       operate as expected. Use the supplied variables for the values given by those
functions.

'=====[ SETTINGS ]==============
  par1 := default_low_threshold ' default "low" threshold value
 (min: 0, max: 2^bits_s_in - 2)
  par2 := default_high_threshold' default "high" threshold value
 (min: 0, max: 2^bits_s_in - 2)
'===============================


  stop
  longfill(@done, 0, 38)
  longmove(@tx_pin, @DTPin, 8)
  ptr_start := -1
  in_standby := 0
  timescale := clkfreq / 1000   ' milliseconds -- could be changed to any scale

  chanstate := chanstate_ptr
  chanval := chanval_ptr
  chanmax := chanmax_ptr
  chanmin := chanmin_ptr
  IF (((BitCount => 10 AND clkfreq => 80_000_000) OR (BitCount > 12 AND clkfreq => 64_000_000))
 OR ChCount == 1)
    fastslow := 0                ' slow mode

  ELSE
    fastslow := -1               ' fast mode

  cogon := (cog := cognew(@entry, @tx_pin))

  waitcnt(7000 + cnt)            ' wait for driver to fully initialize

  RETURN cogon

PUB stop
'' Stops cog if running

  IF (cogon~)
    cogstop(cog)

PUB setthreshold (low, high)
'' sets the high/low thresholds for all channels

  par1 := low
  par2 := high
  command := 1

PUB resetmaxminall
'' reset maximum and minimum values on all channels (min set to 0 and max set to max ADC value
based on bits_s_in)

  command := 2

PUB resetmax (ch)
'' reset maximum value on this channel (set to 0)

  channel := ch
  command := 3

PUB resetmin (ch)
'' reset minimum value on this channel (set to max ADC value based on bits_s_in)

  channel := ch
  command := 4

PUB waithigh (ch, watchdog, waitmode)
'' wait until this channel is in high state (returns channel value at end of wait in case of
watchdog timeout)
'' waitmode enables or disables acknolegment of current channel state. Meaning, if waitmode is
0 and the channel
''         is currently high, but the actual value is floating between the two thresholds it is
not considered
''         high. In this mode, the method will only return when the channel's value has exceeded
the high
''         threshold level. waitmode 1 will read current channel state to see if it is high, if
it is it will
''         return immediately.

  IF (_checkchannel(ch) == false)
    RETURN false
  done := 0
  par1 := waitmode
  duration := timescale * watchdog
  channel := ch
  command := 5
  REPEAT UNTIL (done)

  RETURN retval

PUB waitlow (ch, watchdog, waitmode)
'' wait until this channel is in low state (returns channel value at end of wait in case of
watchdog timeout)
'' waitmode enables or disables acknolegment of current channel state. Meaning, if waitmode is
0 and the channel
''        is currently low, but the actual value is floating between the two thresholds it is
not considered
''        low. In this mode, the method will only return when the channel's value is eqaul or
below the threshold
''        level. waitmode 1 will read current channel state to see if it is low, if it is it
will return
''        immediately.

  IF (_checkchannel(ch) == false)
    RETURN false

  done := 0
  par1 := waitmode
  duration := timescale * watchdog
  channel := ch
  command := 6
  REPEAT UNTIL (done)

  RETURN retval

PUB getfreq (ch, watchdog, precision, highhold)
'' return frequency on this channel
'' Note: This function waits for a high-to-low transition, then starts a timer. Once 2 to-the-
power-of precision
''        high-to-low transitions have been achieved, it averages the period lengths between
the high-to-low
''        transitions and returns the period length in retval. The frequency is determined by
dividing current
''        clock speed by the period length. 0 precision means it wait for only one frequency
cycle, while 5 will
''        make it wait for 32 cycles then average the results.
''        highhold can reduce the function's resolution by requiring the channel's high-state
to be held for a
''        certain number of cycles. 0 disables the feature.

  IF (_checkchannel(ch) == false)
    RETURN false

  done := 0
  par1 := precision
  par2 := highhold
  duration := timescale * watchdog
  channel := ch
  command := 7
  REPEAT UNTIL (done)

  RETURN clkfreq / retval

PUB average (ch, samples)
'' return average value of a channel for a certain number of samples

  IF (_checkchannel(ch) == false)
    RETURN false

  done := 0
  par1 := (samples #> 1)
  channel := ch
  command := 8
  REPEAT UNTIL (done)

  RETURN retval / samples

PUB average_time (ch, watchdog)
'' return average value of a channel for a certain period of time

  IF (_checkchannel(ch) == false)
    RETURN false

  done := 0
  duration := timescale * (watchdog #> 1)                                       ' do not allow
0 as a value
  channel := ch
  command := 9
  REPEAT UNTIL (done)

  RETURN retval / count

PUB standby_enable (waitlength)
'' puts the ADC into standby mode, the ADC isn't sampled, and the cog is in waitcnt most of
the time
'' Note: checks for standby disable command every waitlength cycles (higher values use
slightly less power, but
''       the cog may have to wait before the next command can be issued

  in_standby := -1
  par1 := waitlength #> 27
  command := 10

PUB standby_disable
'' pulls the ADC out of standby (just call send any command)
'' Note: The first command sent to the driver (after standby_enable) is ignored, but it will
pull the driver out
''       of standby

  IF (in_standby)
    done := 0
    command := -1
    REPEAT UNTIL (done)
    in_standby := 0

PUB getmax (ch)
'' return maximum value on this channel since last reset
```

```
  RETURN chanmax[ch]

PUB getmin (ch)
'' return minimum value on this channel since last reset

  RETURN chanmin[ch]

PUB getstate (ch)
'' returns current state of this channel (-1 == high or 0 == low)

  RETURN chanstate[ch]

PUB getval (ch)
'' returns current value of this channel

  RETURN chanval[ch]

PUB getsamples
'' returns number of samples taken during last operation

  RETURN count

PRI _checkchannel (ch)
'' Check if cannel being accessed is being monitored (without this check, driver locks up)

  IF (ch => channels)
    RETURN false
  RETURN true

DAT
                        ORG
'=====[ START ]=========================================
entry
'-----[ SETUP VALUES, PINS, AND TIMER ]-----------------
                        MOV     p1, PAR
                        RDLONG  p2, p1                  ' get data-out (TX) pin
                        MOV     DPin2, p2
                        MOV     DPin, #1
                        SHL     DPin, p2

                        ADD     p1, #4
                        RDLONG  p2, p1                  ' get data-in (RX) pin
                        MOV     NPin, #1
                        SHL     NPin, p2

                        ADD     p1, #4
                        RDLONG  p2, p1                  ' get clock (CLK) pin
                        MOV     CPin2, p2
                        MOV     CPin, #1
                        SHL     CPin, p2

                        ADD     p1, #4
                        RDLONG  p2, p1                  ' get reset (CS) pin
                        MOV     CSPin, #1
                        SHL     CSPin, p2

                        ADD     p1, #4
                        RDLONG  adcchs, p1              ' get ADC being used (8/4/2/1-channel
ADC)
```

```
                        CMP     adcchs, #1      WZ      ' if 1 channel ADC
              IF_NZ     JMP     #:skip1
                        MOV     sval, #0
                        MOV     dval, #0
                        MOV     valplus1, #0
                        MOV     nullbits, #3
                        JMP     #:vdone

:skip1                  CMP     adcchs, #2      WZ      ' if 2 channel ADC
              IF_NZ     JMP     #:skip2
                        MOV     sval, sval_2
                        MOV     dval, dval_2
                        MOV     valplus1, valplus1_2
                        MOV     nullbits, #0
                        JMP     #:vdone

:skip2                  MOV     sval, sval_8            ' if nothing else...assumed 8/4
channel ADC
                        MOV     dval, dval_8
                        MOV     valplus1, valplus1_8
                        MOV     nullbits, #2
:vdone

                        ADD     p1, #4
                        RDLONG  chs, p1                 ' get number of channels to monitor

                        MAX     chs, adcchs             ' limit channels scanned to channels
on ADC

                        ADD     p1, #4
                        RDLONG  bitssin, p1             ' get number bits to shift in
                        MOV     bitssin1, bitssin
                        SUB     bitssin1, #1            ' number of bits to ignore (usually
shift-in-bits minus 1)


                        ADD     p1, #4
                        RDLONG  p2, p1          WZ      ' get mode: single/differential
              IF_NZ     MOV     mval, sval              ' single
              IF_Z      MOV     mval, dval              ' differential

                        ADD     p1, #4
                        MOV     pr1_addr, p1            ' get parameter1 address
                        RDLONG  chlow, p1               ' set low threshold

                        ADD     p1, #4
                        MOV     pr2_addr, p1            ' get parameter2 address
                        RDLONG  chhigh, p1              ' set high threshold


                        ADD     p1, #4
                        RDLONG  p3, p1                  ' get start type (-1 == pointer, 0 ==
normal)

                        ADD     p1, #4
                        RDLONG  fs, p1                  ' get speed mode (-1 == fast, 0 ==
slow)
```

```
                        ADD     p1, #4
                        MOV     done_addr, p1           ' get "completed" mark address

                        ADD     p1, #4
                        MOV     chl_addr, p1            ' output value address

                        ADD     p1, #4
                        MOV     wd_addr, p1             ' watchdog timeout address

                        ADD     p1, #4
                        MOV     out_addr, p1            ' output value address

                        ADD     p1, #4
                        MOV     count_addr, p1          ' get sample count value address

                        ADD     p1, #4
                        MOV     cmd_addr, p1            ' get input command address

                        TJNZ    p3, #:pointer_start     ' pointers

                        ADD     p1, #4
                        MOV     state_addr, p1          ' get state address

                        ADD     p1, #32
                        MOV     chval_addr, p1          ' get state address

                        ADD     p1, #32
                        MOV     max_addr, p1            ' get channel max address

                        ADD     p1, #32
                        MOV     min_addr, p1            ' get channel min address

                        JMP     #:cont

:pointer_start          ADD     p1, #4
                        RDLONG  state_addr, p1          ' get state address

                        ADD     p1, #32
                        RDLONG  chval_addr, p1          ' get state address

                        ADD     p1, #32
                        RDLONG  max_addr, p1            ' get channel max address

                        ADD     p1, #32
                        RDLONG  min_addr, p1            ' get channel min address

:cont

                        MOV     OUTA, #0                ' set all low
                        MOV     DIRA, #0                ' set all input
                        MOV     OUTA, CSPin             ' set CS pin high (inactive)
                        OR      DIRA, CPin              ' set pins we use to output
                        OR      DIRA, CSPin             ' set pins we use to output

                        MOV     val, mval
                        MOV     val2, val               ' backup

                        MOV     chmin, #1
                        TEST    chmin, #1     WC        ' set C
                        RCL     chmin, bitssin1         ' rotate 1's into val_min to set 1 to
all bits bitssin deep
                        MOV     adcmax, chmin           ' store maximum value

                        MOVD    :setmax, #chmax         ' probably not necessary, but just in
case (only happens during setup)
                        MOV     idx, #8                 ' do to all eight channels
:setmax                 MOV     chmax, #0               ' set to zero
                        ADD     :setmax, dplus1         ' move pointer
                        DJNZ    idx, #:setmax

                        MOVD    :setmin, #chmin         ' probably not necessary, but just in
case (only happens during setup)
                        MOV     idx, #8                 ' do to all eight channels
:setmin                 MOV     chmin, adcmax           ' set to max value
                        ADD     :setmin, dplus1         ' move pointer
                        DJNZ    idx, #:setmin

                        MINS    chlow, #0               ' make sure low value is not below 0
                        SUB     adcmax, #1              ' reduce max by one so "high" is
possible
                        MAXS    chhigh, adcmax          ' make sure high value is not above
max ADC value
                        ADD     adcmax, #1              ' return adcmax back to original
location

                        MOV     idx, #0                 ' clear any possible values
                        MOV     cmd, #0                 ' clear any possible values
                        MOV     output, #0              ' clear any possible values
                        MOV     clkready, #0            ' clear any possible values
                        MOV     strt, #0                ' clear any possible values
                        MOV     roll, #0                ' clear any possible values
                        MOV     curchl, #0              ' clear any possible values



                        CMP     adcchs, #1      WZ
              IF_NZ     MOV     CTRA, nco
              IF_NZ     ADD     CTRA, DPin2             ' NCO on this pin number

                        TJZ     fs, #mainloop           ' if in slow mode, skip CTRB setup
                        MOV     CTRB, nco
                        ADD     CTRB, CPin2             ' NCO on this pin number
                        MOV     FRQB, #1


'-----[ MAIN LOOP ]-------------------------------------
mainloop
                        MOV     bits_in, bitssin        ' set to reset value

                        MOV     PHSA, val2              ' get backup
                        TJNZ    fs, #fast_shift         ' if fast mode, go to fast shift

                        MOV     Bits, #4                ' 5 bit output

                        ANDN    OUTA, CSPin             ' set CS pin low (active)


'-----[ SHIFT COMMAND OUT ]-----------------------------
```

```
                        CMP     adcchs, #1      WZ      ' if single channel ADC, no info to
give
              IF_Z      JMP     #:skip

                        OR      DIRA, DPin              ' set pins we use to output

                        OR      OUTA, CPin              ' start clock cycle
                        ANDN    OUTA, CPin              ' end clock cycle
:shift_out_slow



                        SHL     PHSA, #1                ' shift output value
                        OR      OUTA, CPin              ' start clock cycle
                        ANDN    OUTA, CPin              ' end clock cycle

                        DJNZ    Bits, #:shift_out_slow

                        ANDN    DIRA, DPin              ' set pins we use to input (so same IO
can be used for RX and TX)

:skip
'-----[ NULL BITS ]-------------------------------------
                        TJZ     nullbits, #:cont        ' if zero ignore bits, skip this
                        MOV     emptyclk, nullbits      ' ignore bits
:empty                                                  ' generate empty clocks to ditch
unwanted bits
                        OR      OUTA, CPin              ' start clock cycle
                        ANDN    OUTA, CPin              ' end clock cycle

                        DJNZ    emptyclk, #:empty

:cont
'-----[ SHIFT MSB VALUE IN ]----------------------------
                        MOV     val_out, #0             ' set to reset value (0)
shift_in_slow
                        TEST    NPin, INA       WC      ' if data input pin is high
                        RCL     val_out, #1             ' add input pin value to output

                        OR      OUTA, CPin              ' start clock cycle
                        ANDN    OUTA, CPin              ' end clock cycle

                        ''NOP                             ' slow down input (slowest part of
ADC) add this NOP if not reading information at 80MHz (or faster)

                        DJNZ    bits_in, #shift_in_slow ' continue to end of input value


'-----[ SHIFT LSB IN AND IGNORE ]-----------------------
                        CMP     adcchs, #2      WZ      ' if 2-channel ADC
              IF_NZ     CMP     adcchs, #1      WZ      ' or 1-channel ADC
              IF_Z      JMP     #:skip                  ' skip the ignore bits

                        MOV     emptyclk, bitssin1
:empty                                                  ' generate empty clocks to ditch
unwanted bits
                        OR      OUTA, CPin              ' start clock cycle
                        ANDN    OUTA, CPin              ' end clock cycle
                        DJNZ    emptyclk, #:empty

:skip                   OR      OUTA, CSPin              ' set CS pin high (inactive)
                        JMP     #maxch1
'=====[ END OF SLOW SHIFT ]==============================

fast_shift

                        ANDN    OUTA, CSPin              ' set CS pin low (active)


'-----[ SHIFT COMMAND OUT ]------------------------------
                        OR      DIRA, DPin               ' set pins we use to output
shift_out

                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long

                        SHL     PHSA, #1                 ' shift output value
                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long

                        SHL     PHSA, #1                 ' shift output value
                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long

                        SHL     PHSA, #1                 ' shift output value
                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long

                        SHL     PHSA, #1                 ' shift output value
                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long

                        ANDN    DIRA, DPin               ' set pins we use to input (so same IO
can be used for RX and TX)


'-----[ NULL BITS ]--------------------------------------
                        TJZ     nullbits, #:cont         ' if zero ignore bits, skip this
                        MOV     emptyclk, nullbits       ' ignore bits
:empty                                                   ' generate empty clocks to ditch
unwanted bits
                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long
                        DJNZ    emptyclk, #:empty

:cont
'-----[ SHIFT MSB VALUE IN ]-----------------------------
                        MOV     val_out, #0              ' set to reset value (0)
shift_in
                        TEST    NPin, INA       WC       ' if data input pin is high
                        RCL     val_out, #1              ' add input pin value to output
                        NEG     PHSB, #3                 ' Send a pulse 3 clocks long

                        DJNZ    bits_in, #shift_in       ' continue to end of input value


'-----[ SHIFT LSB IN AND IGNORE ]------------------------
                        CMP     adcchs, #2      WZ       ' if 2-channel ADC
              IF_Z      JMP     #:skip                   ' skip the ignore bits

                        MOV     emptyclk, bitssin1
:empty                                                   ' generate empty clocks to ditch
unwanted bits
```

```
                        NEG     PHSB, #3                ' Send a pulse 3 clocks long
                        DJNZ    emptyclk, #:empty

:skip                   OR      OUTA, CSPin             ' set CS pin high (inactive)
'=====[ END OF FAST SHIFT ]=============================


'-----[ DETERMINE MAX/MIN VALUE ]-----------------------
maxch1                  MIN     chmax, val_out          ' set val_max to whichever is highest
minch1                  MAX     chmin, val_out          ' set val_min to whichever is lowest


'-----[ READ COMMANDS ]---------------------------------
                        TJNZ    cmd, #check_cmd         ' check if command already is set
                        RDLONG  cmd, cmd_addr   WZ      ' read input command
              IF_Z      JMP     #no_mode

                        RDLONG  p1, pr1_addr            ' get applicable parameter1
                        RDLONG  p2, pr2_addr            ' get applicable parameter2
                        RDLONG  sectime, wd_addr        ' get applicable watchdog timer limit
                        RDLONG  chl, chl_addr           ' get applicable channel

                        WRLONG  zero, cmd_addr          ' clear input command value


'-----[ EXCECUTE COMMANDS ]-----------------------------
check_cmd
                        CMP     cmd, #1         WZ
              IF_Z      JMP     #set_thresh
                        CMP     cmd, #2         WZ
              IF_Z      JMP     #reset_maxmin
                        CMP     cmd, #3         WZ
              IF_Z      JMP     #reset_max
                        CMP     cmd, #4         WZ
              IF_Z      JMP     #reset_min
                        CMP     cmd, #10        WZ
              IF_Z      JMP     #standby

                        CMP     chl, curchl     WZ      ' if current channel does not match
command channel skip the rest
              IF_NZ     JMP     #no_mode

                        CMP     cmd, #5         WZ
              IF_Z      JMP     #wait_high
                        CMP     cmd, #6         WZ
              IF_Z      JMP     #wait_low
                        CMP     cmd, #7         WZ
              IF_Z      JMP     #det_freq
                        CMP     cmd, #8         WZ
              IF_Z      JMP     #avg_samp
                        CMP     cmd, #9         WZ
              IF_Z      JMP     #avg_time

no_mode
end_mode


'-----[ END OF LOOP ]-----------------------------------
                        WRLONG  val_out, chval_addr     ' put current value in channel's value
                        ADD     curchl, #1                ' move current channel one position
                        CMP     val_out, chlow  WZ, WC
              IF_BE     WRLONG  zero, state_addr
                        ADD     val2, valplus1            ' add one channel to value
                        CMP     val_out, chhigh WZ, WC
              IF_A      WRLONG  negone, state_addr
                        ADD     state_addr, #4            ' move one long
                        ADD     chval_addr, #4            ' move one long
maxch2                  WRLONG  chmax, max_addr           ' max voltage over the second-long
sample
                        ADD     max_addr, #4              ' move one long
                        ADD     maxch1, dplus1            ' move one destination
minch2                  WRLONG  chmin, min_addr           ' min voltage over the second-long
sample
                        ADD     min_addr, #4              ' move one long
                        ADD     minch1, dplus1            ' move one destination
                        ADD     maxch2, dplus1            ' move one destination
                        ADD     minch2, dplus1            ' move one destination
                        ADD     roll, #4

                        CMP     curchl, chs     WZ, WC    ' if it hasn't exceded the number of
channels to be scanned
              IF_B      JMP     #mainloop

                        MOV     curchl, #0                ' set current channel to 0
                        MOV     val2, mval                ' add one channel to value
                        SUB     state_addr, roll           ' move back eight longs
                        SUB     chval_addr, roll           ' move back eight longs
                        SUB     max_addr, roll             ' move back eight longs
                        SUB     min_addr, roll             ' move back eight longs
                        MOVD    maxch1, #chmax            ' move back to original destination
                        MOVD    maxch2, #chmax            ' move back to original destination
                        MOVD    minch1, #chmin            ' move back to original destination
                        MOVD    minch2, #chmin            ' move back to original destination
                        MOV     roll, #0
                        JMP     #mainloop                 ' do it again!




'=====[ SUBROUTNIES ]===================================

'-----[ SET ALL CHANNELS' THRESHOLD ]-------------------
set_thresh
                        MINS    p1, #0                    ' make sure low value is not below 0
                        SUB     adcmax, #1                ' reduce max by one so "high" is
possible
                        MAXS    p2, adcmax                ' make sure high value is not above
max ADC value
                        ADD     adcmax, #1                ' return adcmax back to original
location

                        CMP     p1, p2          WZ, WC    ' make sure low is not above high
              IF_A      MOV     p3, p1
              IF_A      MOV     p1, p2
              IF_A      MOV     p2, p3

                        MOV     chlow, p1                 ' set low
                        MOV     chhigh, p2                ' set high
                        MOV     cmd, #0
                        JMP     #end_mode


'-----[ RESET ALL MAX/MINS ]----------------------------
reset_maxmin
                        MOVD    :setmax, #chmax         ' clear any previous movement
                        MOV     idx, #8                 ' repeat for all 8 locations
:setmax                 MOV     chmax, #0               ' clear max in current address
                        ADD     :setmax, dplus1         ' move up one address
                        DJNZ    idx, #:setmax

                        MOVD    :setmin, #chmin         ' clear any previous movement
                        MOV     idx, #8                 ' repeat for all 8 locations
:setmin                 MOV     chmin, adcmax           ' clear min in current address
                        ADD     :setmin, dplus1         ' move up one address
                        DJNZ    idx, #:setmin

                        MOV     cmd, #0
                        JMP     #end_mode


'-----[ RESET CHANNEL'S MAX ]---------------------------
reset_max
                        MOVD    :setmax, #chmax         ' clear any previous movement
                        MOV     p1, chl                 ' get selected channel
                        SHL     p1, #9
                        ADD     :setmax, p1             ' move address to selected channel
                        MOV     cmd, #0                 ' need one instruction between setting
and using modified instruction
:setmax                 MOV     chmax, #0               ' clear max in current address

                        JMP     #end_mode


'-----[ RESET CHANNEL'S MIN ]---------------------------
reset_min
                        MOVD    :setmin, #chmin         ' clear any previous movement
                        MOV     p1, chl                 ' get selected channel
                        SHL     p1, #9
                        ADD     :setmin, p1             ' move address to selected channel
                        MOV     cmd, #0                 ' need one instruction between setting
and using modified instruction
:setmin                 MOV     chmin, adcmax           ' clear min in current address

                        JMP     #end_mode


'-----[ WAIT FOR CHANNEL HIGH ]-------------------------
wait_high
                        TJNZ    strt, #:arstarted       ' already started

                        CMP     p1, #0          WZ      ' if mode is not 0
              IF_NZ     RDLONG  p3, state_addr  WZ      ' if current state is -1 (high)
              IF_NZ     JMP     #:done                  ' say it is done

                        MOV     output, #0              ' default low output
                        MOV     idx, #0                 ' set index count to 0
                        MOV     strt, cnt               ' start timer
                        NEG     strt, strt              ' get negative (used in place of SUB
later on)
:arstarted
                        CMP     val_out, chhigh WZ, WC
              IF_BE     JMP     #check_watchdog         ' if still low, check watchdog timer

:done                   MOV     output, negone          ' make output current value
                        JMP     #alldone                ' continue normal operation


'-----[ WAIT FOR CHANNEL LOW ]--------------------------
wait_low
                        TJNZ    strt, #:arstarted       ' already started

                        CMP     p1, #0          WZ      ' if mode is not 0
              IF_NZ     RDLONG  p3, state_addr  WZ      ' if current state is 0 (low)
              IF_Z      JMP     #:done                  ' say it is done

                        MOV     output, negone          ' default high output
                        MOV     idx, #0                 ' set index count to 0

                        MOV     strt, cnt               ' start timer
                        NEG     strt, strt              ' get negative (used in place of SUB
later on)
:arstarted
                        CMP     val_out, chlow  WZ, WC
              IF_A      JMP     #check_watchdog         ' if still low, check watchdog timer


:done                   MOV     output, #0              ' make output current value
                        JMP     #alldone                ' continue normal operation


'-----[ DETERMINE FREQUENCY ]---------------------------
det_freq
                        TJNZ    strt, #:arstarted       ' already started

                        MOV     cumul, #0
                        MOV     idx, #0
                        MOV     clkready, #0
                        MOV     track, #0
                        MOV     tmstrt, #0

                        MOV     freqcycls, p1           ' get exponent of times to double
check frequency
                        MOV     cyclesshl, #1
                        SHL     cyclesshl, freqcycls    ' move exponent value to left
                        MOV     highhld, p2

                        MOV     strt, cnt               ' start timer
                        NEG     strt, strt              ' get negative (used in place of SUB
later on)
:arstarted

                        CMP     val_out, chhigh WZ, WC  ' if current value is below or equal
to "threshold value"
              IF_BE     JMP     #vbelow
```

```
vabove
                        CMP     track, highhld  WZ, WC  ' if val_out has been above "threshold
value" for this many times
              IF_AE     MOV     clkready, #1            ' set value to 1 so when val_out goes
below "threshold value" we can clock a Hz
              IF_AE     JMP     #check_watchdog
                        ADD     track, #1
                        JMP     #check_watchdog

vbelow
                        MOV     track, #0               ' clear tracking value
                        CMP     val_out, chlow  WZ, WC
              IF_A      JMP     #check_watchdog

                        CMP     clkready, #0    WZ, WC
                        MOV     clkready, #0            ' reset clock ready value
              IF_BE     JMP     #check_watchdog
                        ADD     cumul, #1               ' add a clock to Hz value

                        CMP     cumul, cyclesshl WZ, WC
              IF_BE     JMP     #:again
                        MOV     p1, cnt                 ' stop timer
                        SUB     p1, tmstrt              ' get difference between timer start
and end
                        'SHR     p1, freqcycls           ' divide to get average period (clock
speed divided by period == frequency)

                        TJZ     freqcycls, #:skip
                        SUB     freqcycls, #1
                        SHR     p1, freqcycls           ' divide to get average period (clock
speed divided by period == frequency)
                        SHR     p1, #1          WC      ' put "half" bit in c
                        ADDX    p1, #0                  ' if "half" bit is set, round up
:skip

                        MOV     output, p1
                        JMP     #alldone
:again
                        TJNZ    tmstrt, #check_watchdog

                        MOV     tmstrt, cnt             ' start first clock cycle timer
                        JMP     #check_watchdog


'-----[ COLLECT SUM OF VALUES TO AVERAGE ]--------------
avg_samp
                        TJNZ    strt, #:arstarted       ' already started

                        MOV     cumul, #0
                        MOV     idx, #0

:arstarted
                        ADD     cumul, val_out         ' add current value to existing output
                        ADD     strt, #1
                        CMP     strt, p1         WZ, WC ' if enough samples have been taken
                        ADD     idx, #1
              IF_AE     MOV     output, cumul
              IF_AE     JMP     #alldone                ' then we are all done
```

```
                        JMP     #mainloop


'-----[ COLLECT SUM OF VALUES TO AVERAGE ]--------------
avg_time
                        TJNZ    strt, #:arstarted        ' already started

                        MOV     cumul, #0
                        MOV     idx, #0

                        MOV     strt, cnt                ' start timer
                        NEG     strt, strt               ' get negative (used in place of SUB
later on)

:arstarted
                        ADD     cumul, val_out
                        ADD     idx, #1

                        MOV     waitlen, cnt             ' get now
                        ADDS    waitlen, strt            ' difference of start and now
                        CMP     waitlen, sectime WZ, WC  ' if below one second...do another loop

              IF_B      JMP     #mainloop                ' do it again!
                        MOV     output, cumul
                        JMP     #alldone


'-----[ GO INTO STANDBY, WAIT FOR EXIT COMMAND ]--------
standby
                        MOV     FRQB, #0                 ' zero FRQB to prevent pin toggling
                        MOV     waitlen, cnt
                        ADD     waitlen, p1
:wait                   RDLONG  cmd, cmd_addr   WZ       ' if no command yet
              IF_Z      WAITCNT waitlen, p1              ' wait for a time (longer the less
power)
              IF_Z      JMP     #:wait                   ' look for command again

                        CMP     fs, #0          WZ
              IF_NZ     MOV     FRQB, #1                 ' if fast mode, re-retup FRQB
                        WRLONG  zero, cmd_addr           ' clear input command value
                        MOV     cmd, #0                  ' space out wrlong (little bit faster)
                        WRLONG  negone, done_addr
                        JMP     #end_mode


'-----[ WATCHDOG ]--------------------------------------
check_watchdog
                        ADD     idx, #1
                        TJZ     sectime, #mainloop       ' if watchdog is disabled...skip it
                        MOV     waitlen, cnt             ' get now
                        ADDS    waitlen, strt            ' difference of start and now
                        CMP     waitlen, sectime WZ, WC  ' if below one second...do another loop

              IF_B      JMP     #mainloop                ' do it again!

alldone
                        WRLONG  output, out_addr         ' output value
                        MOV     strt, #0                 ' clear any timer
                        WRLONG  idx, count_addr          ' number of loops to address
                        MOV     cmd, #0
                        WRLONG  negone, done_addr       ' tell method, PASM command is done
                        MOV     output, #0              ' if timed out, no output
                        JMP     #end_mode               ' move on to next channel


negone                  LONG    -1                      ' $FF_FF_FF_FF
zero                    LONG    0                       ' used for cog memory writes
dplus1                  LONG    1 << 9                  ' destination plus one value
nco                     LONG    %00100 << 26            ' numerically controlled oscillator
counter setting

sval_8                  LONG    %11000 << 27            ' single-ended channel 0 output value (
for 8/4 channel ADC)
dval_8                  LONG    %10000 << 27            ' differential channel 0 output value (
for 8/4 channel ADC)
valplus1_8              LONG    1 << 27                 ' add one channel (for 8/4 channel ADC)
sval_2                  LONG    %1101 << 28             ' single-ended channel 0 output value (
for 2 channel ADC)
dval_2                  LONG    %1001 << 28             ' differential channel 0 output value (
for 2 channel ADC)
valplus1_2              LONG    1 << 29                 ' add one channel (for 2 channel ADC)

sval                    RES                             ' single-ended channel 0 output value
dval                    RES                             ' differential channel 0 output value
valplus1                RES                             ' add one channel
nullbits                RES                             ' null bits between output and input

DPin                    RES                             ' tx
DPin2                   RES                             ' tx
CPin                    RES                             ' clock
CPin2                   RES                             ' clock
CSPin                   RES                             ' cs (reset)
NPin                    RES                             ' rx

sectime                 RES                             ' watchdog period
waitlen                 RES                             ' tmp time
tmstrt                  RES                             ' start timer at first clock
emptyclk                RES                             ' number of empty clocks to excecute

out_addr                RES                             ' output address
max_addr                RES                             ' output address
min_addr                RES                             ' output address
done_addr               RES                             ' output address
count_addr              RES                             ' output address
state_addr              RES                             ' output address
chval_addr              RES                             ' output address

cmd_addr                RES                             ' input address
chl_addr                RES                             ' input address
pr1_addr                RES                             ' input address
pr2_addr                RES                             ' input address
wd_addr                 RES                             ' input address

cmd                     RES                             ' command value
chl                     RES                             ' channel value
curchl                  RES                             ' current cycle's channel

output                  RES                             ' output value
```

```
cumul                   RES                             ' output value cumulator
clkready                RES                             ' ready to add one to frequency ("high"
 criteria met)
track                   RES                             ' store number of times above "
threshold value"
strt                    RES                             ' watchdog start time
idx                     RES
adcmax                  RES                             ' maximum ADC value based on number of
bits_s_in
chs                     RES                             ' number of channels to scan (1-8)
roll                    RES                             ' number of bytes to roll back (when
looping back to channel 0)
fs                      RES                             ' setting for runnnig fast or slow mode
adcchs                  RES                             ' track which ADC is being used (for
different dataschemes)

Bits                    RES                             ' number of bits to shift out
bits_in                 RES                             ' number of value bits to shift in (10
or 12)
mval                    RES                             ' stored value for single/differential
changes
val                     RES                             ' output value (channel number plus
mval)
val2                    RES                             ' "backup" of val
val_out                 RES                             ' shifted in value


freqcycls               RES                             ' number of cycles to count for
frequency
cyclesshl               RES                             ' number of shifts to act as a fast
divider
highhld                 RES                             ' number of cycles to see as "high"
before alowing a "low" value to count a Hz
bitssin                 RES                             ' number of value bits to shift in (10
or 12)
bitssin1                RES                             ' number of value bits to shift in
minus 1 (9 or 11)

p1                      RES                             ' address pointer (for value/pin/
address setup) and parameter1
p2                      RES                             ' temperary value read from address
and parameter2
p3                      RES                             ' tmp value storrage

chhigh                  RES                             ' all channels' threshold before
considered "high"
chlow                   RES                             ' all channels' threshold before
considered "low"
chmax                   RES     8                       ' channel's maxinum value since last
reset
chmin                   RES     8                       ' channel's minimum value since last
reset

                        FIT

{{
┌───────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┐
│                                                    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.                         │
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────┘
}}
```

```
{{
┌──────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────┐
│ Read Only Memory Engine
                              │
│
                              │
│ Author: Kwabena W. Agyeman
                              │
│ Updated: 11/22/2009
                              │
│ Designed For: P8X32A
                              │
│
                              │
│ Copyright (c) 2009 Kwabena W. Agyeman
                              │
│ See end of file for terms of use.
                              │
│
                              │
│ Driver Info:
                              │
│
                              │
│ The ROMEngine interfaces with any standard I2C EEPROM.
                              │
│
                              │
│ The driver is only guaranteed and tested to work at an 80Mhz system clock or higher. The
driver is designed for the P8X32A  │
│ so port B will not be operational.
                              │
│
                              │
│ Nyamekye,
                              │
└──────────────────────────────────────────────────────────────────────────────────────────────
──────────────────────────────┘
}}

CON
                              ''
                              ''    3.3V
                              ''
                              ''     ┬
                              ''     │
                              ''     ≷ RΩ
                              ''     │
  Data_Pin              = 29  ''  ├──•──┤ SDA
                              ''
                              ''    3.3V
                              ''
                              ''     ┬
                              ''     │
                              ''     ≷ RΩ
                              ''     │
  Clock_Pin             = 28  ''  ├──•──┤ SCL

  EEPROM_Address        = %1010 ' EEPROM Address for device selection.
  EEPROM_Max_Write_Time = 5     ' EEPROM Max Write Time in milliseconds so that write and read
page know when to quit.

  EEPROM_Size           = 65536 ' EEPROM Size in bytes so that read page can properlly prevent
wrap arrounds.

  EEPROM_Page_Size      = 32    ' EEPROM Page Size in bytes so that write page can properlly
prevent wrap arrounds.

PUB readByte(EEPROMaddress) '' 18 Stack Longs

''
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┐
'' │ Reads a byte from the EEPROM.
                               │
'' │
                               │
'' │ Returns the byte on success and false on failure. Could return a byte of value 0.
                               │
'' │
                               │
'' │ EEPROMaddress - Starting byte address of the data to access.
                               │
''
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┘

  result &= readPage(EEPROMaddress, @result, 1)

PUB writeByte(EEPROMaddress, value) '' 19 Stack Longs

''
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┐
'' │ Writes a byte to the EEPROM.
                               │
'' │
                               │
'' │ Returns true on success and false on failure. Writes over page boundaries will be
truncated.                             │
'' │
                               │
'' │ EEPROMaddress - Starting byte address of the data to access.
                               │
'' │ Value         - Value to write to the EEPROM.
                               │
''
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┘

  return writePage(EEPROMaddress, @value, 1)

PUB readWord(EEPROMaddress) '' 18 Stack Longs

''
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┐
'' │ Reads a word from the EEPROM.
```

```
                                        │
'' │
                                        │
'' │ Returns the word on success and false on failure. Could return a word of value 0.
                                        │
'' │
                                        │
'' │ EEPROMaddress - Starting byte address of the data to access.
                                        │
''
└────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────┘

  result &= readPage(EEPROMaddress, @result, 2)

PUB writeWord(EEPROMaddress, value) '' 19 Stack Longs

''
┌────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────┐
'' │ Writes a word to the EEPROM.
                                        │
'' │
                                        │
'' │ Returns true on success and false on failure. Writes over page boundaries will be
truncated.                                        │
'' │
                                        │
'' │ EEPROMaddress - Starting byte address of the data to access.
                                        │
'' │ Value         - Value to write to the EEPROM.
                                        │
''
└────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────┘

  return writePage(EEPROMaddress, @value, 2)

PUB readLong(EEPROMaddress) '' 18 Stack Longs

''
┌────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────┐
'' │ Reads a long from the EEPROM.
                                        │
'' │
                                        │
'' │ Returns the long on success and false on failure. Could return a long of value 0.
                                        │
'' │
                                        │
'' │ EEPROMaddress - Starting byte address of the data to access.
                                        │
''
└────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────────────┘

  result &= readPage(EEPROMaddress, @result, 4)
```

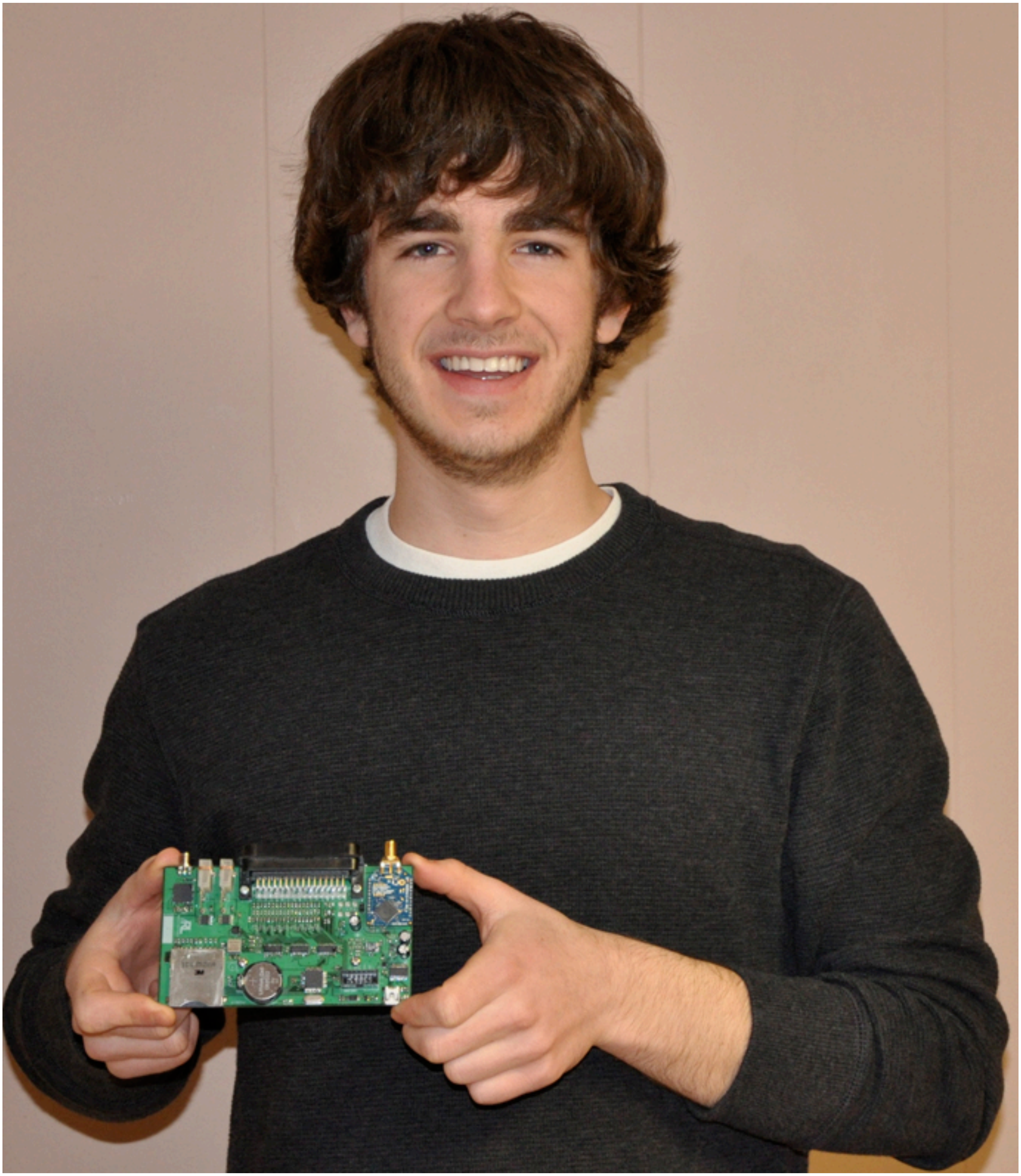
```
PUB writeLong(EEPROMaddress, value) '' 19 Stack Longs

''
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┐
'' │ Writes a long to the EEPROM.
                               │
'' │
                               │
'' │ Returns true on success and false on failure. Writes over page boundaries will be
truncated.                             │
'' │
                               │
'' │ EEPROMaddress - Starting byte address of the data to access.
                               │
'' │ Value         - Value to write to the EEPROM.
                               │
''
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┘

  return writePage(EEPROMaddress, @value, 4)
PUB readPage(EEPROMaddress, RAMaddress, byteCount) '' 14 Stack Longs

''
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┐
'' │ Reads bytes from the EEPROM.
                               │
'' │
                               │
'' │ Returns true on success and false on failure. Reads over device boundaries will be
truncated.                            │
'' │
                               │
'' │ EEPROMaddress - Starting byte address of the data to access.
                               │
'' │ RAMaddress    - Starting byte address of the data to write to.
                               │
'' │ ByteCount     - Number of bytes to read.
                               │
''
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┘

  byteCount := ((byteCount <# (constant(|<((>|(EEPROM_Size #> 1)) - 1)) - (EEPROMaddress &
constant((|<((>|(EEPROM_Size #> 1)) - 1)) - 1)))) #> 0)

  result := eepromPoll(EEPROMaddress)

  if(result)

    stopDataTransfer
    startDataTransfer

    result and= transmitPacket(constant((EEPROM_Address << 4) | 1) | ((EEPROMaddress >> 15) &
$E))
```

```
    repeat byteCount
      byte[RAMaddress++] := receivePacket(--byteCount)

  stopDataTransfer

PUB writePage(EEPROMaddress, RAMaddress, byteCount) '' 14 Stack Longs

''
┌──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┐
'' │ Writes bytes to the EEPROM.
                               │
'' │
                               │
'' │ Returns true on success and false on failure. Writes over page boundaries will be
truncated.                             │
'' │
                               │
'' │ EEPROMaddress - Starting byte address of the data to access.
                               │
'' │ RAMaddress    - Starting byte address of the data to read from.
                               │
'' │ ByteCount     - Number of bytes to write.
                               │
''
└──────────────────────────────────────────────────────────────────────────────────────────────
────────────────────────────┘

  byteCount := ((byteCount <# (constant(|<(>|(EEPROM_Page_Size #> 1)) - 1)) - (EEPROMaddress &
 constant((|<(>|(EEPROM_Page_Size #> 1)) - 1)))) #> 0)

  result := eepromPoll(EEPROMaddress)

  if(result)

    repeat byteCount
      result and= transmitPacket(byte[RAMaddress++])

  stopDataTransfer

PRI eepromPoll(EEPROMaddress) ' 8 Stack Longs

  startDataTransfer

  result := cnt

  repeat until(transmitPacket(constant(EEPROM_Address << 4) | ((EEPROMaddress >> 15) & $E)))

    stopDataTransfer
    startDataTransfer

    if((cnt - result) > (clkfreq / constant(1000 / ((EEPROM_Max_Write_Time <# 1000) #> 1))))
      stopDataTransfer
      return false

  result := transmitPacket(EEPROMaddress >> 8)
  result and= transmitPacket(EEPROMaddress)

PRI transmitPacket(value) ' 4 Stack Longs
```

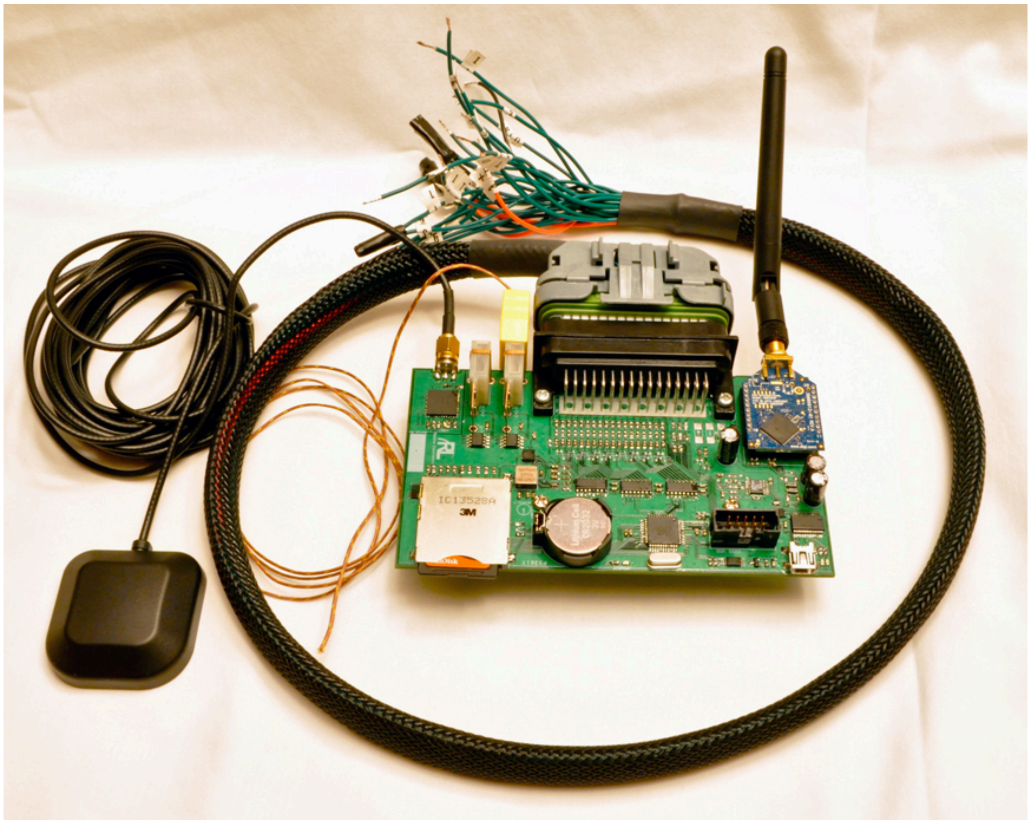
```
  value := ((!value) >< 8)

  repeat 8

    dira[constant(((Data_Pin <# 31) #> 0))] := value

    dira[constant(((Clock_Pin <# 31) #> 0))] := false

    dira[constant(((Clock_Pin <# 31) #> 0))] := true

    value >>= 1

  dira[constant(((Data_Pin <# 31) #> 0))] := false

  dira[constant(((Clock_Pin <# 31) #> 0))] := false

  result := not(ina[constant(((Data_Pin <# 31) #> 0))])

  dira[constant(((Clock_Pin <# 31) #> 0))] := true

  dira[constant(((Data_Pin <# 31) #> 0))] := true

PRI receivePacket(aknowledge) ' 4 Stack Longs

  dira[constant(((Data_Pin <# 31) #> 0))] := false

  repeat 8

    result <<= 1

    dira[constant(((Clock_Pin <# 31) #> 0))] := false

    result |= ina[constant(((Data_Pin <# 31) #> 0))]

    dira[constant(((Clock_Pin <# 31) #> 0))] := true

  dira[constant(((Data_Pin <# 31) #> 0))] := not(not(aknowledge))

  dira[constant(((Clock_Pin <# 31) #> 0))] := false
  dira[constant(((Clock_Pin <# 31) #> 0))] := true

  dira[constant(((Data_Pin <# 31) #> 0))] := true

PRI startDataTransfer ' 3 Stack Longs

  dira[constant(((Data_Pin <# 31) #> 0))] := true
  dira[constant(((Clock_Pin <# 31) #> 0))] := true

PRI stopDataTransfer ' 3 Stack Longs

  dira[constant(((Clock_Pin <# 31) #> 0))] := false
  dira[constant(((Data_Pin <# 31) #> 0))] := false

{{

┌───────────────────────────────────────────────────────────────────────────────────────────
─────────────────────────────┐
│                                                  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.                        │
└──────────────────────────────────────────────────────────────────────────────────────────────
───────────────────────────────┘
}}
```

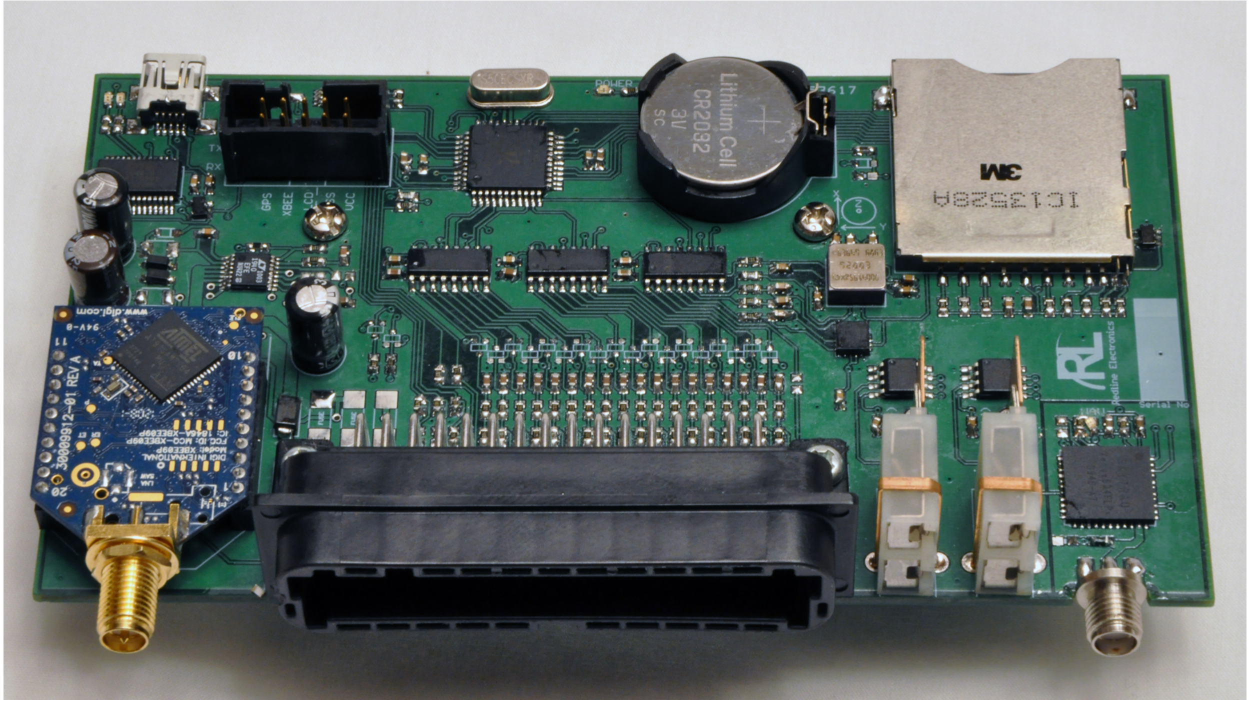
6. Pictures



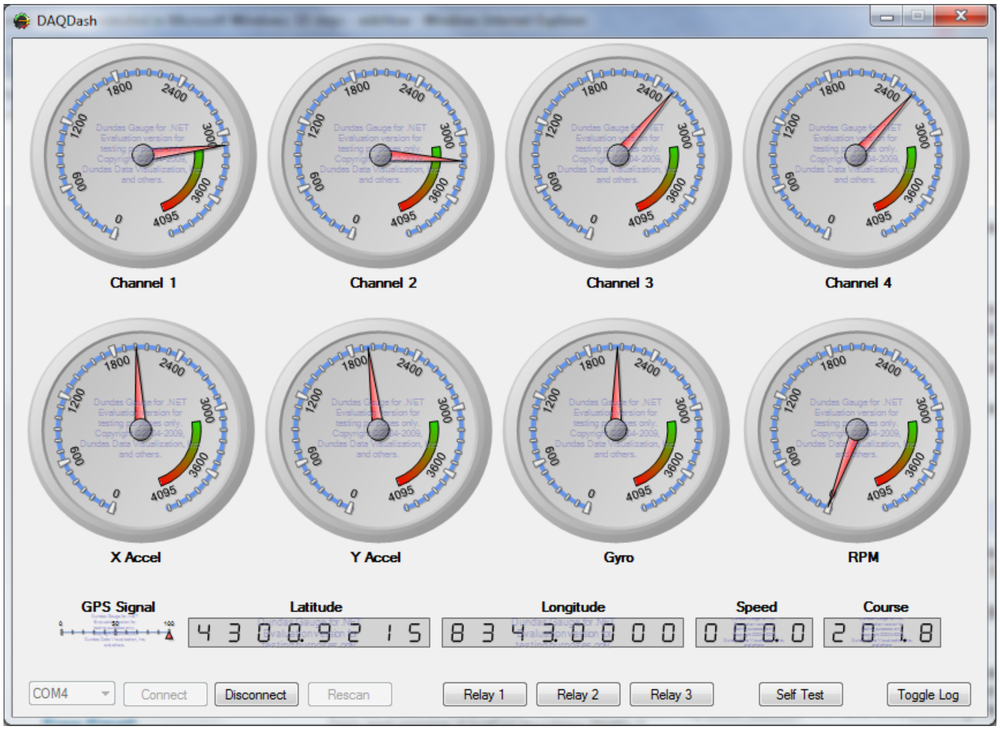
Ryan David, holding DAQPac



DAQPac with wiring harness, GPS and transceiver antennas, and thermocouple



Detail view of DAQPac



Latitude 4300.9215

Longitude 8343.0000

Speed 000.0

Course 201.8

One of several potential implementations for interfacing with DAQPac