4.0 Receiver Software

4.1 Program Structure

The software has two major sections, the main routine and an interrupt service routine. This description will go from the general to more specific in terms of objects, the state transitions for each object, and timing diagrams:

Figure 4-1 illustrates the program object structure. The 3 major objects are the GPS RCVR, the Channel, and the Satellite. The satellite interacts with the channel through the RF signal. There is a one-to-one correspondence of each channel with one satellite. There can be as many as 32 satellites (not counting other satellites using the same PRN family) but the GP2021 has only 12 channels. With a 24 satellite constellation the maximum number of satellites in view of a receiver on the ground is 12. With a larger constellation some will not be tracked if more than 12 are in view. The channels interact with the GPS RCVR through the GPS ISR (Interrupt Service Routine).

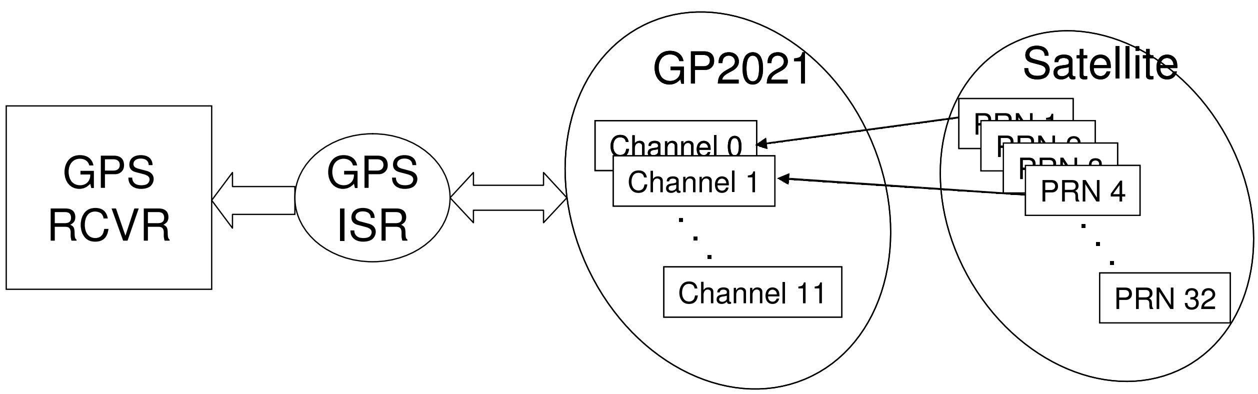

Figure 4-1
OpenSourceGPS Object Structure

More detail of the interactions between GPS RCVR and GPS ISR are shown in figure 4-2. The GPS ISR communicates directly with flags that tell the main program when to compute a navigation fix, when satellite visibility should be checked and when a new frame of data is ready. The navigation data and transmission time is provided in common memory.

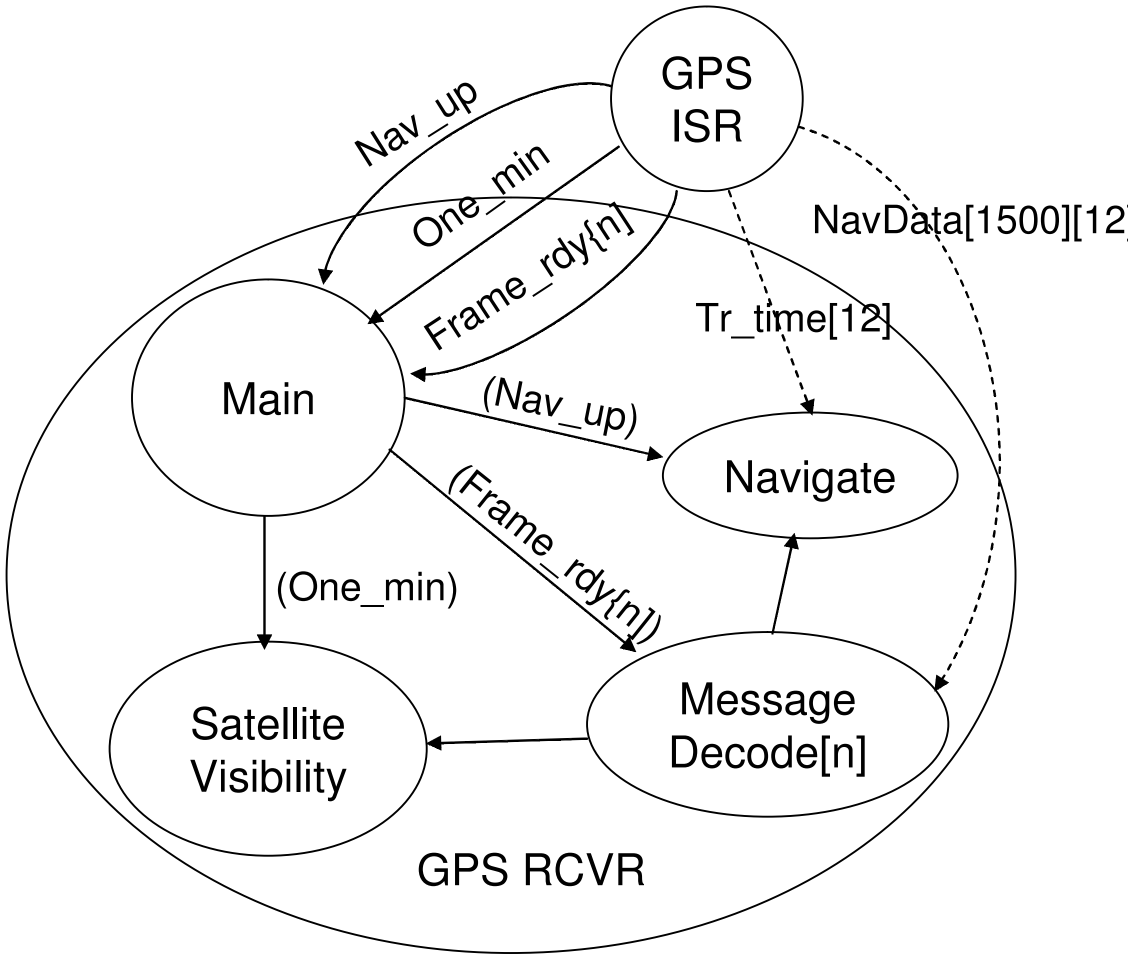

Figure 4-2
GPS Receiver Software Collaboration Diagram

The receiver state diagram is shown in figure 4-3. The most complex part is determining how to start. From then on it is either navigating or re-starting. The cold start is a bootstrap operation. If the receiver does not know enough information to determine which satellites are in view it must search every PRN code to see if it is available. In addition, since the position or receiver clock is not known the range of Doppler to be searched is also very wide. A warm start can be performed if an almanac is available and a rough idea of the receiver's location is available. The Doppler search can be narrowed. The last starting algorithm is the hot start. This is when the data needed for a warm start is available along with valid clock and ephemeris data. As soon as the start of a subframe is found the range measurements from the satellite can be used.

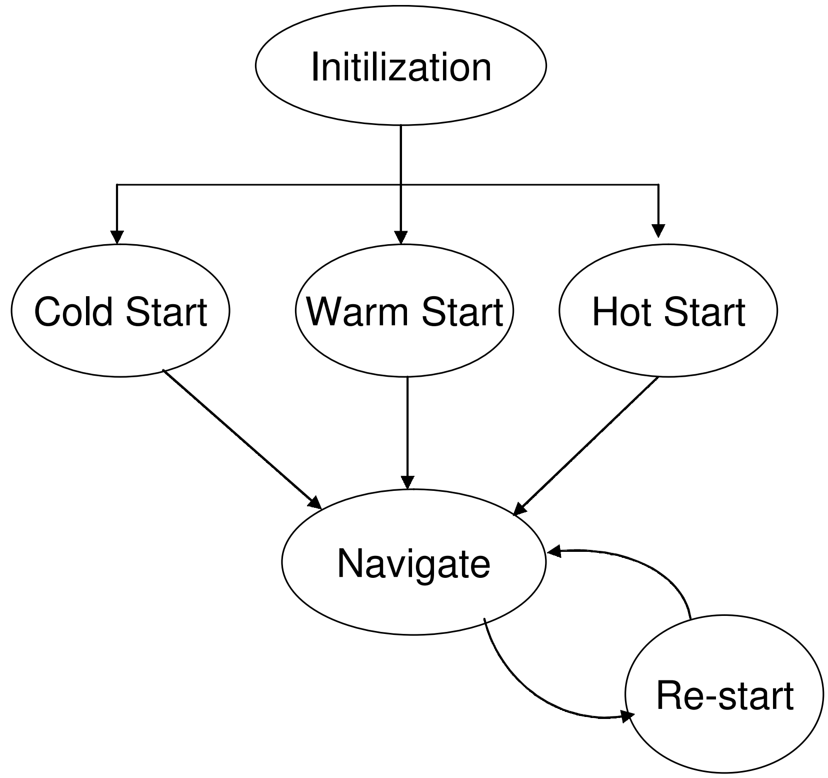

Figure 4-3
GPS RCVR State Diagram

The channel state diagram is the most complex. The prompt and dither magnitudes are the root sum squared value of the I and Q components of the two correlators which are separated by ½ chip. The state numbers are the ones seen on page 1 of the display. When fully tracking the satellite the channel is in state 4.

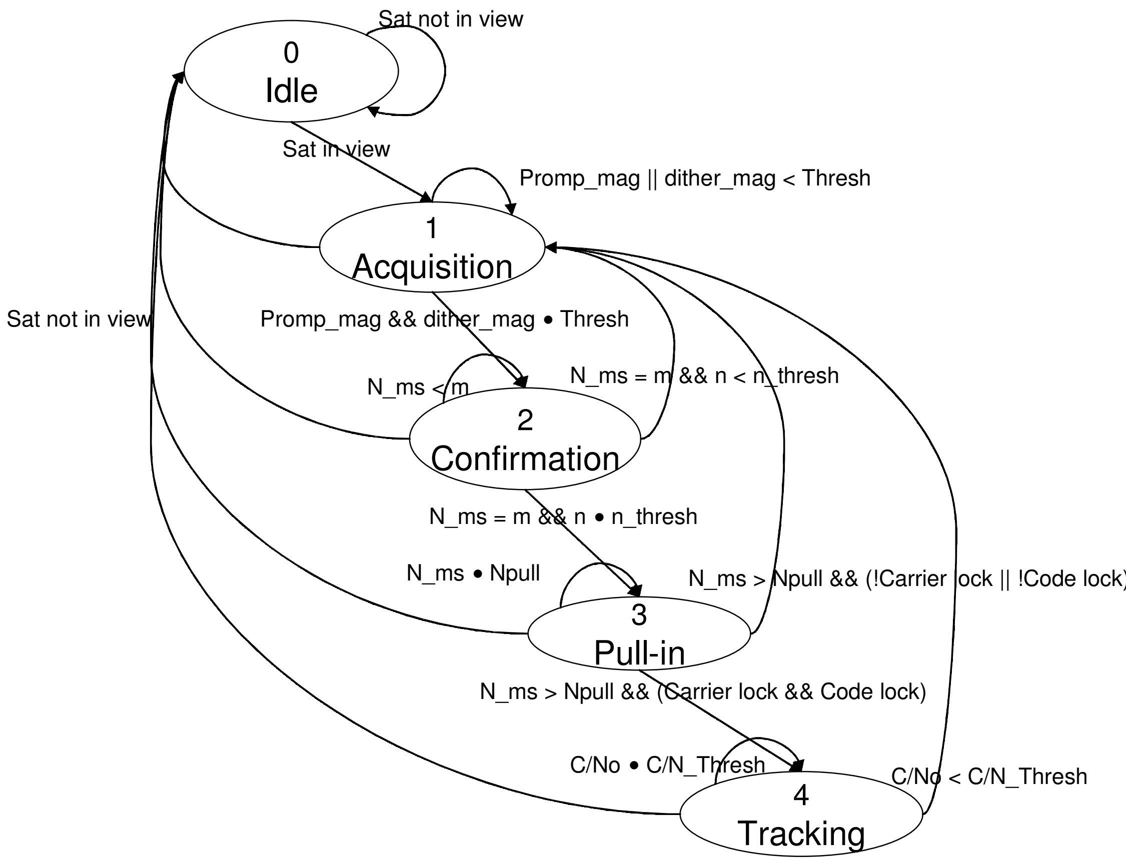

Figure 4-4
Channel State Diagram

The state of the satellite is shown in figure 4-5. We start out not knowing the almanac or ephemeris. Whatever data becomes available has the possibility of showing the satellite as healthy or unhealthy. Ultimately the satellite is declared usable only if the current ephemeris data says it is healthy.

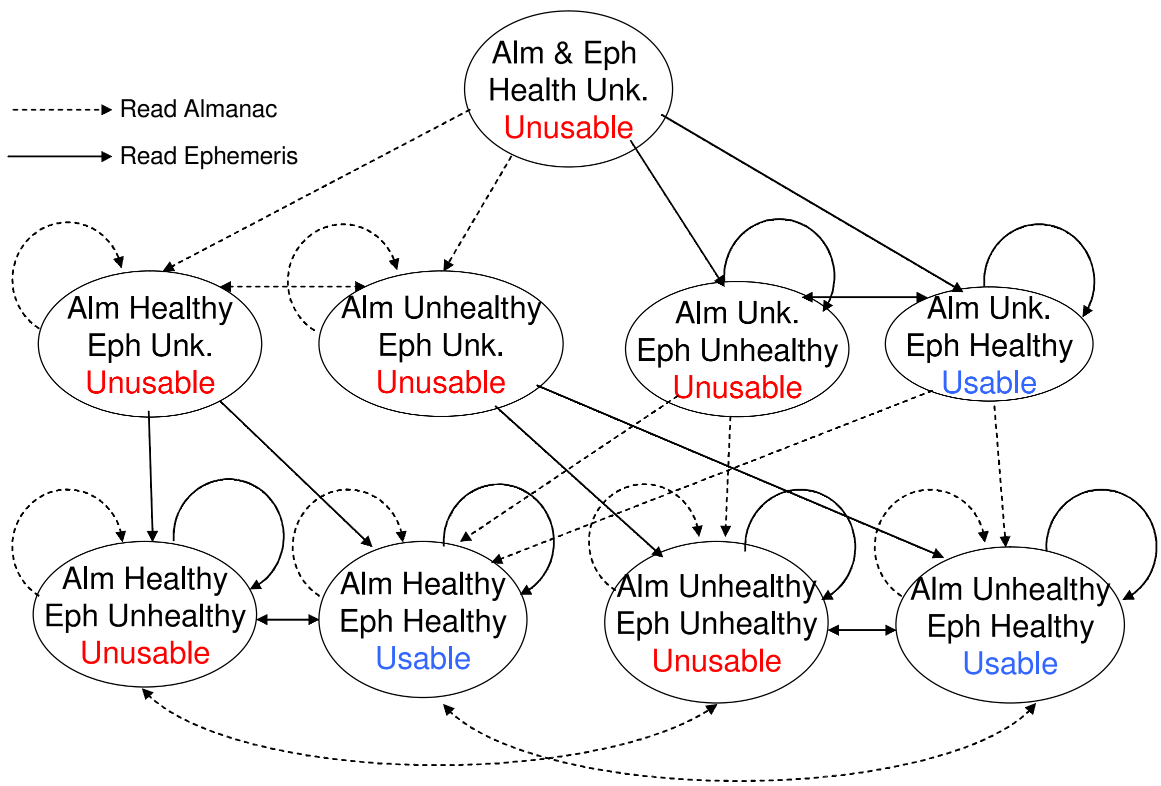

Figure 4-5
Satellite State Diagram

In figure 4-6 the major startup actions are depicted. Since GPS is such an accurate source of time the computer clock and clock interrupt routine are taken over. The computer clock is set to interrupt every few hundred micro-seconds and the IRQ 0 of the real time clock is re-directed to the GPS ISR. After the GP2021, the interrupt controller and PC clock are configured the program alternates between the GPS RCVR and GPS ISR code.

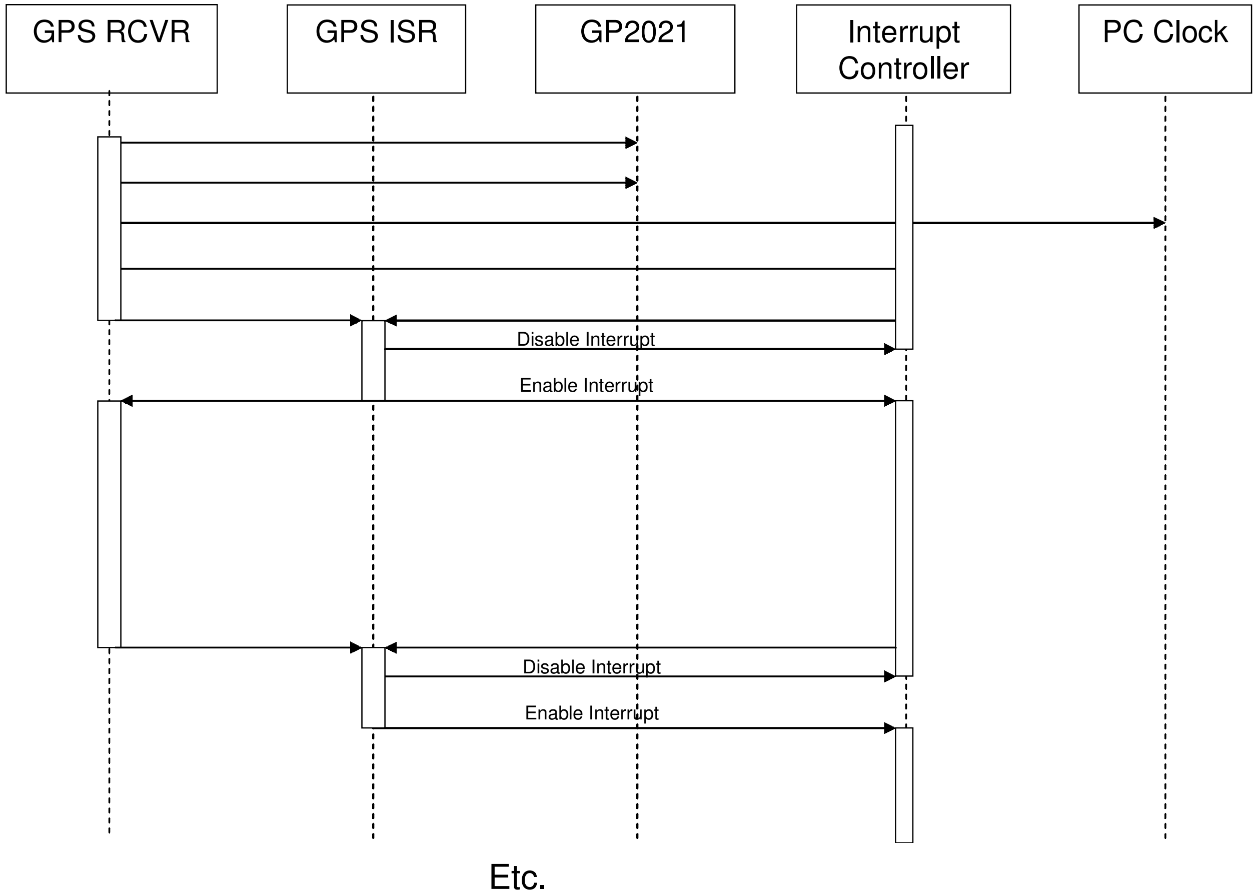

Figure 4-6
Receiver Startup Sequence Diagram

Figure 4-7 provides some more detail on the GPS interrupt service routine. The PC clock is set to wake up the interrupt controller after a few hundred micro-seconds. The first thing the ISR must do is to latch the channel correlation data. After reading the correlator and TIC status it reads the correlators and gets the measurement data. At this point it loops through all of the channels with data to be processed and either slews the correlator (during acquisition), or sets the carrier and code DCOs to track the signal. In order to avoid writing an ISR to be re-entrant the PC clock is reset after all of the channels are processed. This avoids more than 1 interrupt request being active at any time.

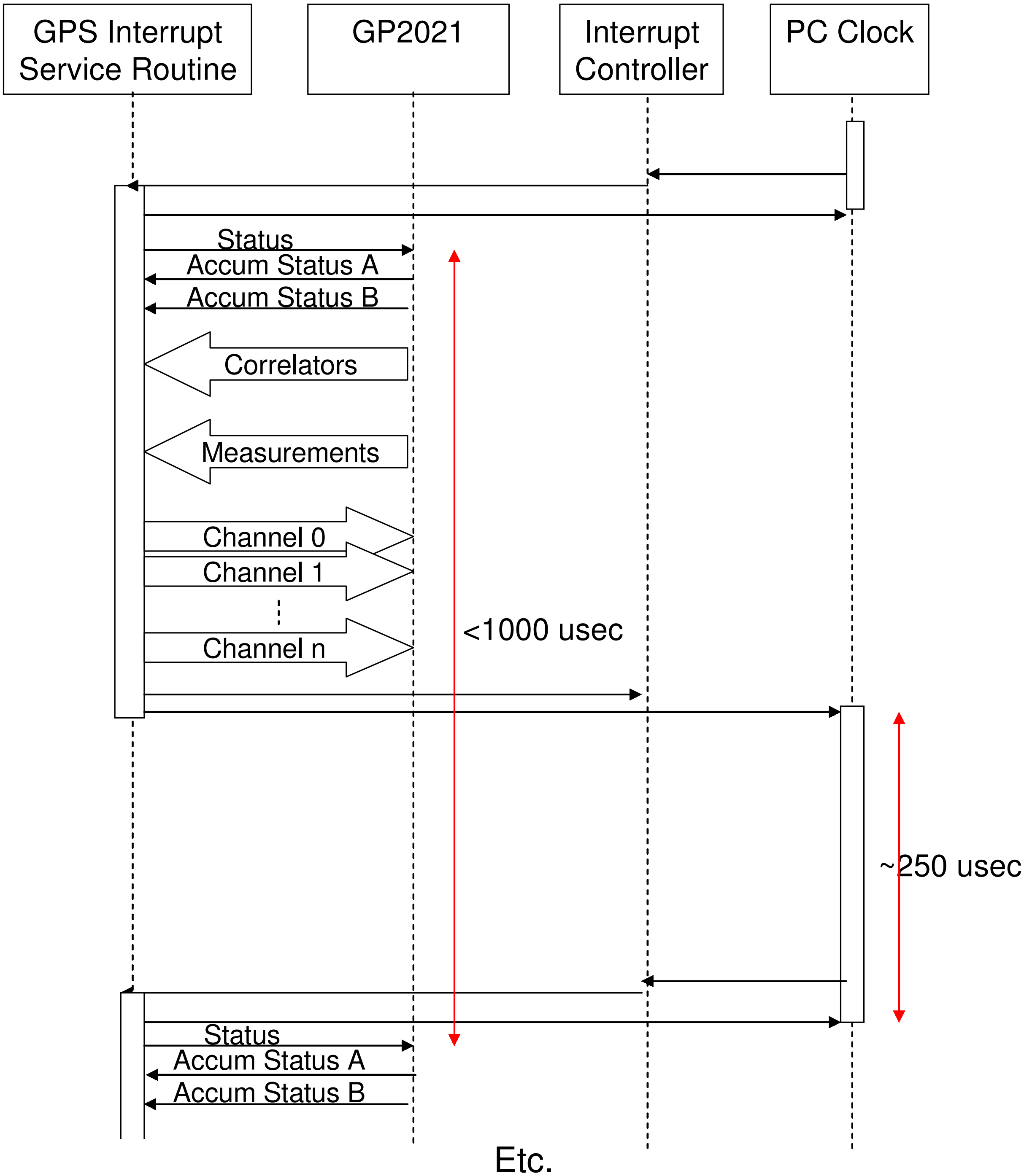

Figure 4-7
GPS Interrupt Service Routine Sequence Diagram

4.1.1 Constants.h

Only a few constants such as pi, the speed of light etc. are defined.

```
double const pi=3.1415926535898E0,r_to_d=57.29577951308232;
double const c=2.99792458e8,omegae=7.2921151467E-5;
//    WGS-84 speed of light m/sec and earth rotation rate rad/sec

double const lambda=0.1902936728; // L1 wavelength in meters

long const SCALED_PI_ON_2 = 25736L;  // used for fixed point Atan function
long const SCALED_PI =      51472L;  // used for fixed point Atan function
```

4.1.2 Globals.h

Globals.h is set up with 2 sections. The first is the definition of the globals with their corresponding initialization and is only used by MAIN which is defined only in gpsrcvr.cpp. The second section is the same set of globals but without initialization which is included when needed by other parts of the program.

```
#ifdef MAIN

channel chan[12];
int a_missed,n_chan,chmax=11,display_page=0;

unsigned test[16]=
{ 0x0001,0x0002,0x0004,0x0008,0x0010,0x0020,0x0040,0x0080,
  0x0100,0x0200,0x0400,0x0800,0x1000,0x2000,0x4000,0x8000};

unsigned int tr_ch[13];

int prn_code[37]={0,0x3f6,0x3ec,0x3d8,0x3b0,0x04b,0x096,0x2cb,0x196,0x32c,
                0x3ba,0x374,0x1d0,0x3a0,0x340,0x280,0x100,0x113,0x226,
                0x04c,0x098,0x130,0x260,0x267,0x338,0x270,0x0e0,0x1c0,
                0x380,0x22b,0x056,0x0ac,0x158,0x058,0x18b,0x316,0x058};

int out_debug,out_pos,out_vel,out_time,out_kalman;

hms cur_lat,cur_long;

#define IRQLEVEL        0       // IRQ Line

char Version[40]; // NMEA
//  These are arrays for debugging
//  they can be written into while running and dumped to a
//  file at the end of the run
//long qdither[6][1500];
//long qprompt[6][1500];
//long idither[6][1500];
//long iprompt[6][1500];
//int qdither0[30000];
//int qprompt0[30000];
```

```
//int idither0[30000];
//int iprompt0[30000];

  long store_code,store_carrier;
// definitions with default values which can be overridden in
// file rcvr_par.dat
int  nav_tic,search_max_f=30,search_min_f=0,cold_prn=1,ICP_CTL=0;
long rms=312,acq_thresh=650,code_corr,time_on=0;
long pull_code_k=111,pull_code_d=7,pull_carr_k=-12,pull_carr_d=28;
long trk_code_k=55, trk_code_d=3, trk_carr_k=-9, trk_carr_d=21;
long cc_scale=1540;
float nav_up=1.0;
double speed,heading;
int  pull_in_time=1000,phase_test=500;
long  d_freq=4698,d_tow,trk_div=19643;
int   confirm_m=10,n_of_m_thresh=8,key,tic_count=0,hms_count=0;
int   nav_count,min_flag,nav_flag,sec_flag,n_track;
unsigned int interr_int=512;
float clock_offset=-0.6;
ecef rec_pos_ecef;
long   i_TIC_dt;
double m_time[3],delta_m_time,m_error,TIC_dt;
long   TIC_cntr,old_TIC_cntr,TIC_ref=571427L,TIC_sum;

int
bit_pat[12]={0x2,0x4,0x8,0x10,0x20,0x40,0x80,0x100,0x200,0x400,0x800,0x1000};
int last_hi_carr[12],last_hi_code[12],ch_status;

char last_address;

int ms_count;
int astat,mstat;

unsigned int Com0Baud;  // NMEA
unsigned int Com1Baud;
unsigned int GPGGA;
unsigned int GPGSV;
unsigned int GPGSA;
unsigned int GPVTG;
unsigned int GPRMC;
unsigned int GPZDA;

char tzstr[40];// = "TZ=PST8PDT";

time_t thetime;

FILE *stream,*debug,*in,*out,*kalm;

almanac gps_alm[33];

int   SVh[33],ASV[33];
float b0,b1,b2,b3,al0,al1,al2,al3; // broadcast ionospheric delay model
float a0,a1,tot,WNt,dtls,WNlsf,DN,dtlsf; //broadcast UTC data

ephemeris gps_eph[33];

pvt rpvt;
```

```
state receiver;

float gdop,pdop,hdop,vdop,tdop,alm_toa;
unsigned long  clock_tow;
llh  rec_pos_llh;
llh current_loc,rp_llh;
eceft track_sat[13];
ecef rec_pos_xyz;
int alm_gps_week,gps_week,almanac_valid,almanac_flag,handle;
unsigned long sf[6][11];
int p_error[6],status;

enum {off,acquisition,confirm,pull_in,track};
//     0       1          2       3      4
enum {cold_start,warm_start,hot_start,tracking,navigating};
//        0          1          2        3          4

unsigned long test_l[33]={0x00000000L,              // single bit set numbers
  0x00000001L,0x00000002L,0x00000004L,0x00000008L,  // for testing bit positions
  0x00000010L,0x00000020L,0x00000040L,0x00000080L,
  0x00000100L,0x00000200L,0x00000400L,0x00000800L,
  0x00001000L,0x00002000L,0x00004000L,0x00008000L,
  0x00010000L,0x00020000L,0x00040000L,0x00080000L,
  0x00100000L,0x00200000L,0x00400000L,0x00800000L,
  0x01000000L,0x02000000L,0x04000000L,0x08000000L,
  0x10000000L,0x20000000L,0x40000000L,0x80000000L};

float mask_angle;
char  header[45],trailer;
double meas_dop[13];

ecef d_sat[13];

long carrier_ref=0x1f7b1b9L,code_ref=0x016ea4a8L;
double  dt[13],cbias;


int m_tropo, m_iono, align_t; // flags for using tropo and iono models


satvis  xyz[33];

#else

extern channel chan[12];
extern int a_missed, n_chan, chmax, display_page;

extern unsigned test[16];
extern unsigned int tr_ch[13];
extern int prn_code[37];
extern int out_debug,out_pos,out_vel,out_time,out_kalman;

extern hms cur_lat,cur_long;

#define IRQLEVEL        0       // IRQ Line
```

```
extern char Version[40]; // NMEA
//  These are arrays for debugging
//  they can be written into while running and dumped to a
//  file at the end of the run
//  long far i_prompta[4500],far q_prompta[4500];
//  long far i_dithera[4500],far q_dithera[4500];
//  long far car_freq[4500],far chip_freq[4500];
//  int far data_bit[4500];

//extern long qdither[6][1500];
//extern long qprompt[6][1500];
//extern long idither[6][1500];
//extern long iprompt[6][1500];
//extern int qdither0[30000];
//extern int qprompt0[30000];
//extern int idither0[30000];
//extern int iprompt0[30000];


extern  long store_code,store_carrier;
// definitions with default values which can be overridden in
// file rcvr_par.dat
extern int  nav_tic,search_max_f,search_min_f,cold_prn,ICP_CTL;
extern long rms,acq_thresh,code_corr,time_on;
extern long pull_code_k,pull_code_d,pull_carr_k,pull_carr_d;
extern long trk_code_k, trk_code_d, trk_carr_k, trk_carr_d;
extern long cc_scale;
extern float nav_up;
extern double speed,heading;
extern int  pull_in_time,phase_test;
extern long  d_freq,d_tow,trk_div;
extern int   confirm_m,n_of_m_thresh,key,tic_count,hms_count;
extern int   nav_count,min_flag,nav_flag,sec_flag,n_track;
extern unsigned int interr_int;
extern float clock_offset;
extern ecef rec_pos_ecef;
extern long   i_TIC_dt;
extern double m_time[3],delta_m_time,m_error,TIC_dt;
extern long   TIC_cntr,old_TIC_cntr,TIC_ref,TIC_sum;

extern int bit_pat[12];
extern int last_hi_carr[12],last_hi_code[12],ch_status;

extern char last_address;

extern int ms_count;
extern int astat,mstat;

extern unsigned int Com0Baud;  // NMEA
extern unsigned int Com1Baud;
extern unsigned int GPGGA;
extern unsigned int GPGSV;
extern unsigned int GPGSA;
extern unsigned int GPVTG;
extern unsigned int GPRMC;
extern unsigned int GPZDA;
```

```
extern char tzstr[40];// = "TZ=PST8PDT";

extern time_t thetime;

extern FILE *stream,*debug,*in,*out,*kalm;

extern almanac gps_alm[33];

extern int   SVh[33],ASV[33];
extern float b0,b1,b2,b3,al0,al1,al2,al3; // broadcast ionospheric delay model
extern float a0,a1,tot,WNt,dtls,WNlsf,DN,dtlsf; //broadcast UTC data

extern ephemeris gps_eph[33];

extern pvt rpvt;

extern state receiver;

extern float gdop,pdop,hdop,vdop,tdop,alm_toa;
extern unsigned long  clock_tow;
extern llh  rec_pos_llh;
extern llh current_loc,rp_llh;
extern eceft track_sat[13];
extern ecef rec_pos_xyz;
extern int alm_gps_week,gps_week,almanac_valid,almanac_flag,handle;
extern unsigned long sf[6][11];
extern int p_error[6],status;

enum {off,acquisition,confirm,pull_in,track};
//     0      1         2       3      4
enum {cold_start,warm_start,hot_start,tracking,navigating};
//        0          1          2        3          4

extern unsigned long test_l[33];

extern float mask_angle;
extern char  header[45],trailer;
extern double meas_dop[13];

extern ecef d_sat[13];

extern long carrier_ref,code_ref;
extern double  dt[13],cbias;


extern int m_tropo, m_iono, align_t; // flags for using tropo and iono models


extern satvis  xyz[33];

#endif
```

4.2 Data Structures

The following section describes the data and data structures used in the software and are defined in structs.h .

4.2.1 Channel data structures:

```
struct channel
{
  int  state;
  long code_freq,      // commanded code DCO setting
       carrier_freq,   // commanded carrier DCO setting
       doppler,        // estimated Doppler in Hz
       carrier_corr;   // carrier DCO correction for clock error
  char message[1500],  // a frame of navigation data
       tow_sync,       // flag to indicate time sync
       prn,            // PRN assigned to this channel
       bit,            // number of data bits into message
       frame_ready;    // flag to indicate frame of data is ready to
                       // be processed
  int  offset,       // Bit number of start of nav message in the
                      // circular bit register
       codes,
       n_freq,     //  frequency delta in search pattern
       del_freq;   //  delta frequency used in search
  int  t_count,
       ms_count,   // number of ms into state 3
       ms_set,     //
       i_confirm;
  int  ms_epoch,
       n_frame,
       ch_time,
       i_count;
  int  con_thresh, // threshold
       n_thresh,   // number of times above threshold in confirm
       sfid,
       missed,
       page5;
  int  i_dith,     // binary number from correlator I dither
       q_dith,     // binary number from correlator Q dither
       i_prompt,   // binary number from correlator I prompt
       q_prompt;   // binary number from correlator Q prompt
  long sum,
       avg,
       old_theta,
       old_q_sum,
       th_rms;
  long dfreq,
       dfreq1,
       dcarr1,
```

```
       dcarr,
       cycle_sum;
  long old_carr_dco_phase;
  long q_dith_20,  // Q dither coherently integrated for 20 ms
       q_prom_20,  // Q prompt coherently integrated for 20 ms
       i_dith_20,  // I dither coherently integrated for 20 ms
       i_prom_20;  // I prompt coherently integrated for 20 ms
  long prompt_mag, // RSS of I and Q of prompt correlator
         dith_mag; // RSS of I and Q of dither correlator
  long tr_bit_time,// GPS time when signal left satellite
       meas_bit_time,// number of bits into the GPS week at
                      // measurement time
       TOW,          // Time of Week from nav message
       TLM;          // Telemetry word from nav message
  long carrier_counter;
  long  d_carr_phase;
  double int_carr_phase;
  unsigned long fifo0,  // fifo used for message synchronization
                fifo1;
  unsigned int carr_dco_phase,
               carr_cycle_l,
               carr_cycle_h;
  unsigned int epoch,
               code_phase,
               code_dco_phase;
  float CNo;   // Estimated signal carrier to noise ratio dB-Hz
};
```

4.2.2 Satellite Data Structures:

```
struct almanac            // Approximate orbital parameters
{
  float w,      // orbit angular velocity
        ety,    // eccentricity
        inc,    // inclination
        rra,    // rate of right ascension
        sqa,    // square root of semi-major axis
        lan,    // longitude of ascending node
        aop,    // argument of perigee
        ma,     // mean anomaly
        toa,    // time of almanac
        af0,    // satellite clock offset
        af1;    // satellite clock rate
  char text_message[23];
  int  health,  // is satellite usable?
       week,    //
       sat_file;
};
```

```
struct ephemeris        // Precise orbital parameters
{
 int iode,
     iodc,
     ura,
     valid,
     health,
     week;
 double dn,
        tgd,
        toe,
        toc,
        omegadot,
        idot,
        cuc,
        cus,
        crc,
        crs,
        cic,
        cis;
        ma,
        e,
        sqra,
        w0,
        inc0,
        w,
        wm,
        ety,
        af0,
        af1,
        af2;
};
```

4.2.3 Receiver Data Structures:

```
float b0,b1,b2,b3,     // Klobuchar Ionospheric
      al0,al1,al2,al3; // delay model parameters

float a0,a1,tot,WNt,dtls,WNlsf,DN,dtlsf; // GPS -> UTC conversion data

struct ecef
{
  double x,y,z;
};

struct eceft
{
  double x,y,z,tb;
  float az,el;
```

```
};

struct llh         // Latitude, Longitude, Height above Ellipsoid
{
  double lat,lon,hae;
};

struct pvt         // Position Velocity Time
{
   double x,y,z,dt;
   double xv,yv,zv,df;
};

float gdop,pdop,hdop,vdop,tdop,alm_toa;
unsigned long  clock_tow;
int alm_gps_week,
    gps_week,
    almanac_valid,
    almanac_flag,
    handle;

int m_tropo, m_iono;  // flags for using tropo and iono models
int align_t;          // flag for turning on the algorithm to align
                      //the receiver measurements to GPS time
struct velocity
{
  double  east,north,up;
  double  clock_err;
  ecef    x,y,z;
};

struct dms            // angle in degrees, minutes, seconds
{
 int deg,min;
 float sec;
};

dms cur_lat,
    cur_long;

struct state
{
  velocity  vel;
  eceft     pos;
  llh       loc; // location in lat, long, hae
};



//  These are arrays for debugging
//  they can be written into while running and dumped to a
```

```
//  file at the end of the run
  long far i_prompta[4500],
       far q_prompta[4500];
  long far i_dithera[4500],
       far q_dithera[4500];
  long far car_freq[4500],
       far chip_freq[4500];
  int far data_bit[4500];
  long store_code,
       store_carrier;

// definitions with default values which can be
// overridden in file rcvr_par.dat
int  nav_tic,
     search_max_f=30,
     search_min_f=0,
     cold_prn=1;
long rms=312,        // the standard deviation of noise in correlation
                     // counts
     acq_thresh=650,
     code_corr,
     time_on=0;
long pull_code_k=111,
     pull_code_d=7,
     pull_carr_k=-12,
     pull_carr_d=28;
long trk_code_k=55,
     trk_code_d=3,
     trk_carr_k=-9,
     trk_carr_d=21;
long cc_scale=1540; // number of carrier cycles in a CA chip
float nav_up=1.0;   // navigation update interval (sec)
double speed,       // horizontal speed in meters/sec
       heading;     // horizontal heading deg from north->east
int  pull_in_time=1000, // number of ms for pull in
     phase_test=500;    // number of ms at end of pull in to
                        // measure phase error
long  d_freq=4698,
      d_tow,
      trk_div=19643;
int   confirm_m=10,
      n_of_m_thresh=8,
      key,
      tic_count=0,
      hms_count=0;
int   nav_count,
      min_flag,
      nav_flag,
      sec_flag,
      n_track;
unsigned int interr_int=512;  // approx number of micro sec to delay next
                              // interrupt
```

```
float clock_offset=-0.6; // extimate of clock freq error in
                         // ppm (parts per million
ecef rec_pos_ecef;       // receiver position in ecef
long   i_TIC_dt;
double m_time[3],
delta_m_time,
m_error,
TIC_dt;
long   TIC_cntr,
       old_TIC_cntr,
       TIC_ref=571427L,  // nominal TIC setting for 100 ms tic
                         // intervals
       TIC_sum;

int debug_counter;
```

Matrix classes:

```
typedef class CMatrix Matrix;
typedef class CMatrix ColumnVector;
typedef class CMatrix DiagonalMatrix;
typedef class CMatrix SymmetricMatrix;



struct matrep
{
     double     **Mat;
     int   r;              // number of rows in the matrix
     int   c;              // number of columns in the matrix
};
```

4.3 Receiver Control Algorithms

4.3.1 Introduction

The main routine of course manages the entire program. Only truly time critical functions are part of the interrupt routine. Other functions such as decoding the navigation message and computing the position and velocity fixes are called from the main loop. The overall timing is referenced from the 100 ms “tic”.

4.3.2 Main

```
void main()
{
  char  ch;
  self_test();
  io_config(0x301);
  test_control(0);
  system_setup(0);
  reset_cntl(0x0);
//  delay(100);
  reset_cntl(0x1fff);
  ch_status=1;
  read_rcvr_par();
  rec_pos_xyz.x=0.0;
  rec_pos_xyz.y=0.0;
  rec_pos_xyz.z=0.0;

  if (out_kalman==1) kalm   =fopen("gpskalm.log","w+");
  if (out_pos==1 || out_vel==1 ||out_time==1) stream=fopen("gpsrcvr.log","w+");
  if (out_debug==1) debug=fopen("debug.log","w+");
  read_initial_data();
  current_loc=receiver_loc();
  rec_pos_llh.lon=current_loc.lon;
  rec_pos_llh.lat=current_loc.lat;
  rec_pos_llh.hae=current_loc.hae;
  nav_tic=nav_up*10;
  old_TIC_cntr=TIC_cntr=TIC_ref;
//  program_TIC(TIC_cntr);
  code_corr=clock_offset*24.0;
  for (ch=0;ch<=11;ch++) chan[ch].state=off;
  time(&thetime);  // set up thetime so it can be taken over by this program
#ifdef VCPP
  _setbkcolor(1);
  _displaycursor( _GCURSOROFF);
  _clearscreen( _GCLEARSCREEN);  // PGB MS
#endif
#ifdef BCPP
  clrscr();
#endif

  if ( status != cold_start ) chan_allocate();
  else if (status==cold_start ) cold_allocate();
  m_time[1]=clock_tow;
```

```
  read_ephemeris();
  int err;
  open_com( 0, Com0Baud, 0, 1, 8, &err ); // NMEA

  Interrupt_Install();
//
  do
  {
    if (kbhit()) key = getch();
    else         key = '\0';
    for (ch=0;ch<=11;ch++)
    {
      if (chan[ch].frame_ready==1 )
      {
           navmess(chan[ch].prn,ch);   // decode the navigation message
           chan[ch].frame_ready=0;     // for this channel
      }
    }
    if (sec_flag==1)
    {
        SendNMEA();
        almanac_flag=0;
        thetime++;
#ifdef BCPP
        stime(&thetime);
#endif
        clock_tow=(++clock_tow)%604800L;
        time_on++;
        sec_flag=0;
        for (ch=0;ch<=11;ch++)
        {
            if (chan[ch].state==track)
            {
      //  Estimate  C/No
              if (chan[ch].avg>0 )
              {
                   chan[ch].CNo=10.*log10(chan[ch].avg/1395.*
                                             chan[ch].avg/1395.*25.*1.7777);
              }
              else chan[ch].CNo=0.0;
              if (chan[ch].CNo<25.0)
              {
      // calculate carrier clock and doppler correction
                  chan[ch].carrier_corr=(-xyz[chan[ch].prn].doppler-
                                      clock_offset*1575.42)/42.57475e-3;
      // calculate code clock and doppler correction
                  code_corr=clock_offset*24.+xyz[chan[ch].prn].doppler/65.5;
                  chan[ch].code_freq=code_ref+code_corr;
                  ch_code(ch,chan[ch].code_freq);   // 1.023 MHz chipping rate
                  chan[ch].state=acquisition;
                  chan[ch].t_count=0;
                  chan[ch].n_frame=0;
                  chan[ch].codes=0;
                  chan[ch].n_freq=search_min_f;
                  chan[ch].tow_sync=0;
                  chan[ch].del_freq=1;
                  chan[ch].carrier_freq=carrier_ref+chan[ch].carrier_corr+
```

```
                                      d_freq*chan[ch].n_freq;  // set carrier
                      ch_carrier(ch,chan[ch].carrier_freq);       // select carrier
                  }
                }
            }
        }
//    nav fix once every X seconds
    if (nav_flag==1)
    {
      nav_fix();
      nav_flag=0;
    }
//    channel allocation once every minute
    if (min_flag==1)
    {
       if ( status != cold_start ) chan_allocate();
       else if (status==cold_start ) cold_allocate();
       min_flag=0;
#ifdef BCPP
       clrscr();
#endif
#ifdef VCPP
       _clearscreen( _GCLEARSCREEN); // PGB MS
#endif
    }
    display();
    if (key =='p' || key=='P')
    {
       display_page++;
       display_page=display_page % 4;
#ifdef BCPP
       clrscr();
#endif
#ifdef VCPP
       _clearscreen( _GCLEARSCREEN); // PGB MS
#endif
    }
  } while (key != 'x' && key != 'X');/*Stay in loop until 'X' key is pressed.*/
//
// Remove our interrupt and restore the old one
//
  Interrupt_Remove();
  close_com(); // NMEA
// Update the Almanac Data file
  if (almanac_valid==1)  write_almanac();
//  Update the Ephemeris Data file
  write_ephemeris();
//  Update the ionospheric model and UTC parameters
  write_ion_utc();
//    Update the curloc file for the next run
  if ( status==navigating )
  {
     out=fopen("curloc.dat","w+");
     fprintf(out,"latitude  %f\n",rec_pos_llh.lat*r_to_d);
     fprintf(out,"longitude %f\n",rec_pos_llh.lon*r_to_d);
     fprintf(out,"hae       %f\n",rec_pos_llh.hae);
     fclose(out);
```

```
  }

  fcloseall();
}
```

4.3.2.1 Display

```
/*******************************************************************************
FUNCTION display()
RETURNS  None.

PARAMETERS None.

PURPOSE
      This function displays the current status of the receiver on the
      computer screen.  It is called when there is nothing else to do

WRITTEN BY
      Clifford Kelley

*******************************************************************************/
void display(void)
{
  char ch;
#ifdef VCPP
  _settextposition(1,1);
#endif
#ifdef BCPP
  gotoxy(1,1);
#endif
  printf("                     OpenSource GPS Software Version 1.13\n");
  printf("%s",ctime(&thetime));
  printf("TOW  %6ld\n",clock_tow);
  printf("meas time %f  error %f  delta %f\n",m_time[1],m_error,delta_m_time);
  cur_lat.deg=rec_pos_llh.lat*r_to_d;
  cur_lat.min=(rec_pos_llh.lat*r_to_d-cur_lat.deg)*60;
  cur_lat.sec=(rec_pos_llh.lat*r_to_d-cur_lat.deg-cur_lat.min/60.)*3600.;
  cur_long.deg=rec_pos_llh.lon*r_to_d;
  cur_long.min=(rec_pos_llh.lon*r_to_d-cur_long.deg)*60;
  cur_long.sec=(rec_pos_llh.lon*r_to_d-cur_long.deg-cur_long.min/60.)*3600.;
  printf("   latitude    longitude          HAE      clock error (ppm)\n");
  printf("  %4d:%2d:%5.2f  %4d:%2d:%5.2f  %10.2f  %f\n",
  cur_lat.deg,abs(cur_lat.min),fabs(cur_lat.sec),cur_long.deg,abs(cur_long.min),
  fabs(cur_long.sec),rec_pos_llh.hae,clock_offset);
  printf(" Speed     Heading      TIC_dt\n");
  printf(" %lf   %lf   %lf\n",speed,heading*r_to_d,TIC_dt);
  printf("   \n");
  printf("tracking %2d status %1d almanac valid %1d gps week %4d\n",
  n_track,status,almanac_valid,gps_week%1024);
  if (display_page==0)
  {
       printf(
" ch prn state n_freq az el doppler t_count n_frame sfid ura page missed
CNo\n");
       for (ch=0;ch<=11;ch++)
```

```
        {
             printf(
" %2d %2d  %2d  %3d   %4.0f  %3.0f   %6.0f   %4d  %4d  %2d  %3d  %3d%5d
%4.1f\n",
             ch,chan[ch].prn,chan[ch].state,chan[ch].n_freq,
             xyz[chan[ch].prn].azimuth*57.3,xyz[chan[ch].prn].elevation*57.3,

       xyz[chan[ch].prn].doppler,chan[ch].t_count,chan[ch].n_frame,chan[ch].sfid,

       gps_eph[chan[ch].prn].ura,chan[ch].page5,chan[ch].missed,chan[ch].CNo);
        }
  printf(" GDOP=%6.3f  HDOP=%6.3f  VDOP=%6.3f
TDOP=%6.3f\n",gdop,hdop,vdop,tdop);  }
  else if (display_page==1)
  {
       printf(  " ch prn state TLM      TOW  Health  Valid  TOW_sync offset\n");
       for (ch=0;ch<=11;ch++)
       {
            printf(" %2d %2d  %2d  %6ld   %6ld   %2d     %2d     %2d   %4d\n",
            ch, chan[ch].prn, chan[ch].state, chan[ch].TLM, chan[ch].TOW,

       gps_eph[chan[ch].prn].health,gps_eph[chan[ch].prn].valid,chan[ch].tow_sync
,
            chan[ch].offset);
       }
  }
  else if (display_page==2)
  {
       printf(" ch prn state n_freq az  el        tropo        iono\n");
       for (ch=0;ch<=11;ch++)
       {
            printf(" %2d %2d  %2d  %3d   %4.0f  %3.0f   %10.4lf   %10.4lf\n",
            ch, chan[ch].prn, chan[ch].state, chan[ch].n_freq,
            xyz[chan[ch].prn].azimuth*57.3, xyz[chan[ch].prn].elevation*57.3,
            chan[ch].Tropo*c, chan[ch].Iono*c);
       }
  }
  else if (display_page==3)
  {
       printf(" ch prn state      Pseudorange     delta Pseudorange\n");
       for (ch=0;ch<=11;ch++)
       {
            printf(" %2d %2d  %2d  %20.10lf   %15.10lf\n",
            ch, chan[ch].prn, chan[ch].state, chan[ch].Pr, chan[ch].dPr);
       }
  }
  else if (display_page==4)// can be used for debugging purposes
  {
  }

}
```

4.3.2.2 Channel Allocation

The GP2021 has 12 channels and so is capable of tracking up to 12 satellites. The 24 satellite constellation can put from 5 to 12 satellites into view of a receiver on the surface of the earth. Thus for every satellite in view a channel is available to track it. This makes the software much easier than that required if a smaller number of channels is available. There are two allocation algorithms needed. The first, Cold Allocation is used when there is not enough information to to compute what satellites are in view, the second is used when there is.

4.3.2.2.1 Cold Allocation

Cold Allocation is a boot strap algorithm for channel allocation. It tries every PRN code in groups of 12 and searches a very wide Doppler region since we are not sure of the clock error or our velocity or position.

```
/*******************************************************************************
FUNCTION cold_allocate()
RETURNS  None.

PARAMETERS None.

PURPOSE  To allocate the PRNs to channels for a cold start, start by searching
        for PRN 1 through 12 and cycling through all PRN numbers skipping
channels
        that are tracking

WRITTEN BY
      Clifford Kelley

*******************************************************************************/
void  cold_allocate()
{
   satvis dummy;
   char ch,i,alloc;
   search_max_f=50;          // widen the search for a cold start
   dummy=satfind(0);
   almanac_valid=1;
   reset_cntl(0x1fff);
   for (i=1;i<=32;i++)
   {
      if (gps_alm[i].inc>0.0 && gps_alm[i].week!=gps_week%1024) almanac_valid=0;
   }
   if (al0==0.0 && b0==0.0)almanac_valid=0;
   for (ch=0;ch<=chmax;ch++) // if no satellite is being tracked
                             // turn the channel off
   {
      if ( chan[ch].CNo<30.0 )// if C/No is too low turn the channel off
      {
        chan[ch].state=off;
        chan[ch].prn=0;
      }
   }
   for (i=0;i<=chmax;i++)
   {
      alloc=0;
      for (ch=0;ch<=chmax;ch++)
```

```
        {
           if (chan[ch].prn==cold_prn)
           {
              alloc=1;// satellite is already allocated a channel
              break;
           }
        }
        if (alloc==0) // if not allocated find an empty channel
        {
          for (ch=0;ch<=chmax;ch++)
          {
            if (chan[ch].state==off)
            {
               chan[ch].carrier_corr=-clock_offset*1575.42/42.57475e-3;
               chan[ch].carrier_freq=carrier_ref+chan[ch].carrier_corr;// compute
                                                                       // carrier
               ch_carrier(ch,chan[ch].carrier_freq);             // select carrier
               chan[ch].code_freq=code_ref;
               ch_code(ch,chan[ch].code_freq);       // 1.023 MHz chipping rate
               ch_cntl(ch,prn_code[cold_prn]|0xa000);// 0xa000 for late select
                                                     // satellite
               chan[ch].prn=cold_prn;
               ch_on(ch);
               chan[ch].state=acquisition;
               chan[ch].codes=0;
               chan[ch].n_freq=search_min_f;
               chan[ch].del_freq=1;
               cold_prn=cold_prn%32+1;
               break;
            }
          }
        }
    }
}
```

4.3.2.2.2 Warm/Hot Allocation

The Warm/Hot Allocation routine uses the current time, location, and satellite almanac to predict what satellites are in view and what doppler will be observed.

```
/******************************************************************************
FUNCTION chan_allocate()
RETURNS  None.

PARAMETERS None.

PURPOSE
      This function allocates the channels with PRN numbers

WRITTEN BY
      Clifford Kelley

******************************************************************************/
```

```
void  chan_allocate()
{
   char ch,prnn,alloc;
   int i;
   almanac_valid=1;
   for (prnn=1;prnn<=32;prnn++)
   {
      xyz[prnn]=satfind(prnn);
      if (gps_alm[prnn].inc>0.0 && gps_alm[prnn].week!=gps_week%1024)
      {
         almanac_valid=0;
      }
   }
   if (al0==0.0 && b0==0.0)almanac_valid=0;
   for (ch=0;ch<=11;ch++) // if the sat has dropped below mask angle
                          // turn the channel off
   {
      if (xyz[chan[ch].prn].elevation < mask_angle ||
                    gps_alm[chan[ch].prn].ety == 0.0)
      {
         chan[ch].state=off;
         chan[ch].tow_sync=0;
         chan[ch].prn=chan[ch].offset=0;
         chan[ch].CNo=0.0;
         chan[ch].n_freq=chan[ch].t_count=chan[ch].n_frame=chan[ch].sfid=0;
         chan[ch].Pr=chan[ch].dPr=chan[ch].Tropo=chan[ch].Iono=0.0;
         chan[ch].TOW=chan[ch].TLM=0;
         for (i=0;i<1500;i++) chan[ch].message[i]=0;
      }
   }
   for (prnn=1;prnn<=32;prnn++)
   {
      if (xyz[prnn].elevation > mask_angle && gps_alm[prnn].health==0 &&
                   gps_alm[prnn].ety != 0.00)
      {
         alloc=0;
         for (ch=0;ch<=11;ch++)
         {
             if (chan[ch].prn==prnn)
             {
                   alloc=1;// satellite already allocated a channel
                   break;
             }
         }
         if (alloc==0) // if not allocated find an empty channel
         {
            for (ch=0;ch<=11;ch++)
            {
               reset_cntl(0x1fff);
               if (chan[ch].state==off)
               {
      // calculate carrier clock and doppler correction
                   chan[ch].carrier_corr=(-xyz[prnn].doppler-
                                      clock_offset*1575.42)/42.57475e-3;
      // calculate code clock and doppler correction
                   code_corr=clock_offset*24.+xyz[prnn].doppler/65.5;
                   chan[ch].code_freq=code_ref+code_corr;
```

```
                    ch_code(ch,chan[ch].code_freq);    // 1.023 MHz chipping rate
                    ch_cntl(ch,prn_code[prnn]|0xa000); // select satellite
                    chan[ch].prn=prnn;
                    chan[ch].state=acquisition;
                    chan[ch].codes=0;
                    chan[ch].n_freq=search_min_f;
                    chan[ch].del_freq=1;
                    chan[ch].carrier_freq=carrier_ref+chan[ch].carrier_corr+
                                  d_freq*chan[ch].n_freq;   // set carrier
                    ch_carrier(ch,chan[ch].carrier_freq);       // select carrier
                    break;
                }
              }
            }
          }
        }
      }
```

4.4 GPS Interrupt Service Routine (GPS ISR)

4.4.1 Main

The main interrupt routine checks for correlation and measurement data, retrieves these data and then branches to the routine for the state each channel is in.

```
/******************************************************************************
FUNCTION GPS_Interrupt()

RETURNS  None.

PARAMETERS None.

PURPOSE
      This function replaces the current IRQ0 Interrupt service routine with
      our GPS function which will perform the acquisition - tracking functions

WRITTEN BY
      Clifford Kelley

******************************************************************************/
#ifdef VCPP

void __interrupt _far GPS_Interrupt(void) // MS

#endif


#ifdef BCPP

void interrupt GPS_Interrupt(...)

#endif
{
//      int astat,mstat;
        unsigned int add;
        char ch;
        to_gps(0x80,0);              // tell 2021 to latch the correllators
        a_missed=from_gps(0x83);     // did we miss any corellation data
        astat=from_gps(0x82);        // get info on what channels have data ready
        for (ch=0;ch<=chmax;ch++)
        {
             if (astat & test[ch])
             {
         add=0x84+(ch<<2);
         chan[ch].i_dith=from_gps(add);    // inphase dither
         add++;
         chan[ch].q_dith=from_gps(add);    // quadrature dither
         add++;
         chan[ch].i_prompt=from_gps(add);  // inphase prompt
         add++;
         chan[ch].q_prompt=from_gps(add);  // quadrature prompt
         if (a_missed & test[ch])
         {
```

```
            chan[ch].missed++;
            ch_accum_reset(ch);
            }
        }
       }
       for (ch=0;ch<=chmax;ch++)
       {
            if (astat & test[ch])
            {
      switch(chan[ch].state)
      {
        case acquisition:
             ch_acq(ch);
             break;
        case confirm    :
             ch_confirm(ch);
             break;
        case pull_in    :
             ch_pull_in(ch);
             break;
        case track      :
             ch_track(ch);
             break;
      }
            }
       }
       mstat=a_missed & 0x2000;             // has a tic occured?
       if (mstat)
       {
            tic_count=(++tic_count)%10;
            if (tic_count==0) sec_flag=1;      // one second has passed
            hms_count=(++hms_count)%600;
            if (hms_count==0) min_flag=1;      // one minute has passed
            nav_count=(++nav_count)%nav_tic;
            if (nav_count==0) nav_flag=1;
            TIC_sum+=TIC_cntr+1;
            add=1;
            for (ch=0;ch<=chmax;ch++)
        {
             add++;
             chan[ch].carr_cycle_l=from_gps(add);    // get carrier data for
       add++;                                  // computing
             chan[ch].carr_dco_phase=from_gps(add);  // delta-pseudorange
       add+=3;
             chan[ch].carr_cycle_h=from_gps(add);
             add+=3;

chan[ch].cycle_sum=chan[ch].carr_cycle_l+65536L*chan[ch].carr_cycle_h;
            }

         if (nav_count==0)         // time to set up measurement data
         {
              add=1;
              for (ch=0;ch<=chmax;ch++)
           {
                 chan[ch].code_phase=from_gps(add); // get code data for
computing
```

```
                add+=3;                          // pseudorange
                chan[ch].epoch=from_gps(add);
                add++;
                chan[ch].code_dco_phase=from_gps(add);
                add+=4;
                chan[ch].meas_bit_time=chan[ch].tr_bit_time;
                chan[ch].doppler=chan[ch].carrier_freq;
                chan[ch].carrier_counter=chan[ch].cycle_sum;
                chan[ch].d_carr_phase=chan[ch].carr_dco_phase -
chan[ch].old_carr_dco_phase;
                chan[ch].old_carr_dco_phase=chan[ch].carr_dco_phase;
              }
              i_TIC_dt= TIC_sum+old_TIC_cntr-TIC_cntr;
              TIC_sum=0;
            }
            for (ch=0;ch<=chmax;ch++)
          {
          chan[ch].old_carr_dco_phase=chan[ch].carr_dco_phase;
       }
       }
// reset the interrupt
#ifdef VCPP
       _outp( 0x20,0x20 ); // MS
#endif
#ifdef BCPP
       outportb(0x20,0x20);
#endif
}
```

4.4.2 Search State

The process of tracking a satellite begins by searching for the signal. This process starts by doing trial correlations and checking the correlation results to determine if the signal is present. As illustrated in figure 4-8 the search is conducted in code and frequency space. The GP2021 has the ability to add N ½ chip intervals to the correlation time. This is called slewing the code. Starting at frequency where the signal is expected the rss of the inphase and quadrature values are checked for the promt and dither correlators every millisecond. Every millisecond the code is slewed 1 chip and thus every ½ chip in code space is searched.

The values are compared to a threshold. If both the prompt and dither magnitudes are above this threshold the channel transitions to the confirmation state.

As seen in figure 4-9 the sample distribution has Rayleigh distribution when a signal is not present and Ricean when a signal is present.

$$p_n(z) = \frac{z}{\sigma_n^2} e^{-\left(\frac{z^2}{2\sigma_n^2}\right)}$$

$$p_s(z) = \frac{z}{\sigma_n^2} e^{-\left(\frac{z^2+A^2}{2\sigma_n^2}\right)} I_0\left(\frac{zA}{\sigma_n^2}\right)$$

$$P_d = \int_{V_t}^{\infty} \frac{z}{\sigma_n^2} e^{-\left(\frac{z^2+A^2}{2\sigma_n^2}\right)} I_0\left(\frac{zA}{\sigma_n^2}\right) dz$$

$$P_{fa} = \int_{0}^{V_t} \frac{z}{\sigma_n^2} e^{-\left(\frac{z^2}{2\sigma_n^2}\right)} dz$$

$$P_{fa} = e^{-\left(\frac{V_t^2}{2\sigma_n^2}\right)}$$

$$V_t = \sigma_n \sqrt{-2\ln(P_{fa})}$$

This example, is for a 1 ms integration interval, the 1 sigma value of noise is 312 and the signal to noise ratio shown is 3. The process of searching is a classic example of the trade-off between the false alarm rate and the probability of detection

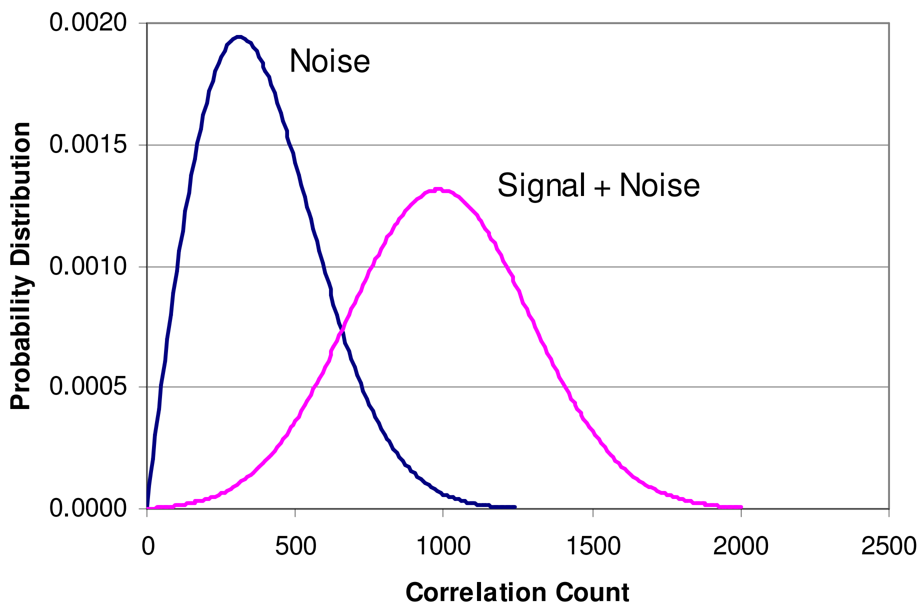

Figure 4-8
Acquisition Distributions

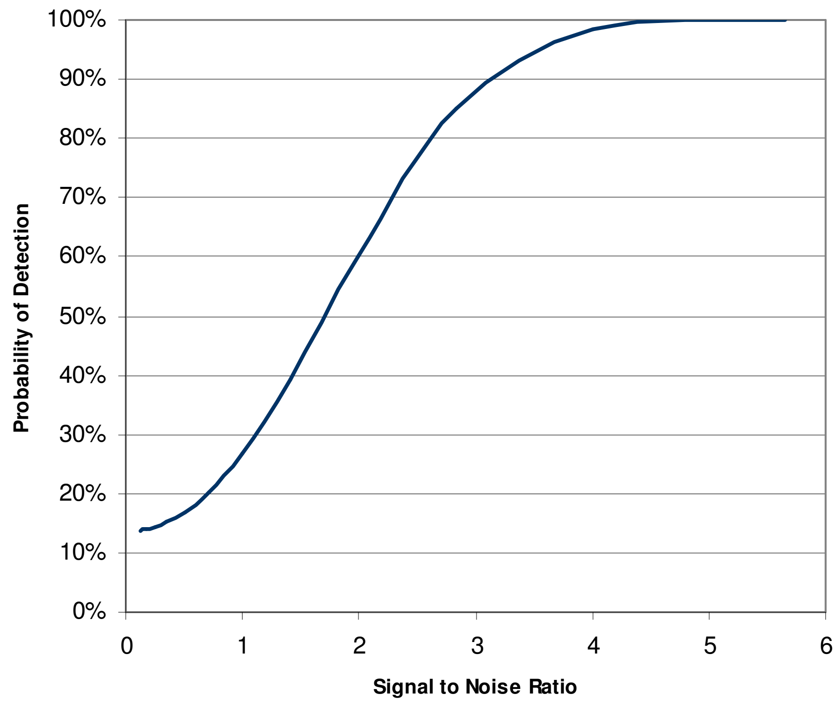

Figure 4-9
Probability of Detection vs Signal to Noise Ratio for Threshold of 630

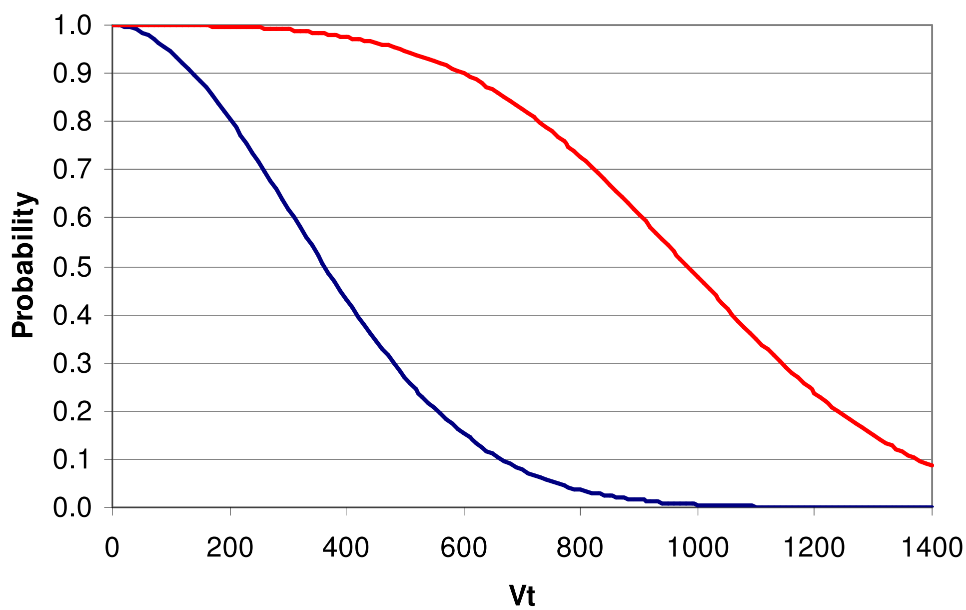

Figure 4-10
Probability of Detection/False alarm vs Vt

In this algorithm both the prompt and dither correlation values must be above the threshold. This keeps the probability of detection high while greatly reducing the false alarm rate.

MORE STUFF NEEDED

```
/******************************************************************************
FUNCTION ch_acq(char ch)
RETURNS  None.

PARAMETERS
                  ch  char

PURPOSE  to perform initial acquisition by searching code and frequency space
                  looking for a high correlation

WRITTEN BY
      Clifford Kelley

******************************************************************************/

void ch_acq(char ch)
{
  long prompt_mag,dith_mag;
  if (abs(chan[ch].n_freq)<=search_max_f)//  search frequencies
  {
       prompt_mag=rss(chan[ch].i_prompt,chan[ch].q_prompt);
       dith_mag  =rss(chan[ch].i_dith,chan[ch].q_dith);
       if ((dith_mag > acq_thresh) && (prompt_mag > acq_thresh))
       {
             ch_code_slew(ch,2044);          // slew back 1 chip so we can
             chan[ch].state=confirm;         // confirm the signal
             chan[ch].i_confirm=0;
             chan[ch].n_thresh=0;
       }
       else
       {
            ch_code_slew(ch,2);
            chan[ch].codes+=2;
       }
       if (chan[ch].codes==2044)
       {
            chan[ch].n_freq+=chan[ch].del_freq;
            chan[ch].del_freq=-(chan[ch].del_freq+sign(chan[ch].del_freq));
            chan[ch].carrier_freq=carrier_ref+chan[ch].carrier_corr+

      d_freq*chan[ch].n_freq;   // set carrier
            ch_carrier(ch,chan[ch].carrier_freq);             // select carrier
            chan[ch].codes=0;
       }
  }
  else
  {
             chan[ch].n_freq=search_min_f;                    // keep looping
             chan[ch].del_freq=1;
```

```
            chan[ch].carrier_freq=carrier_ref+chan[ch].carrier_corr+

      d_freq*chan[ch].n_freq;    // set carrier
            ch_carrier(ch,chan[ch].carrier_freq);            // select carrier
            chan[ch].codes=0;
  }
}
```

4.4.3 Confirmation State

In order to keep the probability of detection high the acquisition state produces a high false alarm rate. The purpose of the confirmation state is to maintain this high probability of detection while decreasing the false alarm rate. In this state we keep the code and carrier settings constant and collect M correlation time samples. If either the prompt or delay results are above the threshold it is counted and if this count is greater than or equal to N then the channel transitions to the pull-in state. If not we transition back to the acquisition state.

The probability of detection at this point is:

MORE STUFF NEEDED

While the probability of false alarm at this point is:

MORE STUFF NEEDED

```
void ch_confirm(char ch)
{
  long prompt_mag, dith_mag;
  prompt_mag = rss(chan[ch].i_prompt,chan[ch].q_prompt);
  dith_mag   = rss(chan[ch].i_dith,chan[ch].q_dith);
  if ((prompt_mag > acq_thresh) || (dith_mag > acq_thresh))
                chan[ch].n_thresh++;
  if (chan[ch].i_confirm==confirm_m)  // have we got m samples?
  {
       if (chan[ch].n_thresh >= n_of_m_thresh)  // are enough > threshold?
       {
            chan[ch].state=pull_in;         // go to pull-in
            chan[ch].ch_time=0;             // initialize variables for pull-in
            chan[ch].sum=0;
            chan[ch].th_rms=0;
            chan[ch].ms_set=0;
       }
       else chan[ch].state=acquisition;
  }
  chan[ch].i_confirm++;    // keep track of the number of samples
```

}

4.4.4 Pull-in

At the point of confirmation we have confirmed to a high degree of confidence that a signal has been found. But, because the correlation peak is so wide in frequency we may be very far off in frequency. The pull-in state attempts to track the signal by using a combination of frequency and phase tracking to lock on the carrier. In addition a code tracking loop is implemented to track the signal in code.
In order to avoid transients the loops are not immediately closed. An x ms delay is built into the code loop and y ms delay is built into the carrier loop. Both loops are updated every millisecond. Since we don't want the tracking state to start in the middle of a data bit we stay in pull-in for at least the required number of ms and then is extended until the end of a data bit. For the last m ms of pull-in the average correlation value, phase error and data bit sync is checked to decide whether or not to go into the tracking state. In addition pull-in is the final step in detection. The pull-in range is approximately 2/3T or 667 Hz for a 1 ms integration interval.

4.4.4.1 Code Tracking

The code tracking loop is a classic DLL updated every millisecond.

MORE STUFF NEEDED

4.4.4.2 Carrier Tracking

The carrier tracking loop has both a phase locked loop (PLL) and a frequency locked loop (FLL). A phase locked loop tracks with less noise but is limited in the phase error it can track. The FLL works with large phase errors but it has more noise In either case since both correlators are about ¼ of a chip from the peak they both have equally valid phase data. Because of this the correlation results from both inphase and both quadrature correlators are added. Since it is likely that the carrier frequency is far away from the receiver carrier DCO setting the loop starts out weighted toward a FLL and it's weighting is decreased as a function of time until the PLL dominates. Figure 4-11 and 4-12 show the FLL weighting and some sample data collected during a pull-in.

MORE STUFF NEEDED

Figure 4-11
PLL Weighting

Figure 4-12
Carrier Phase during Pull-In

4.4.4.3 Bit synchronization

The data message is transmitted at a rate of 20 ms per bit or 50 Hz. Since the correllators dump data at 1 ms intervals each bit lasts for 20 samples. The start of a data bit can be detected by noting when the phase of the signal has changed by 180 degrees. As seen in figure 4-13 each channel in the GP 2021 has a counter that counts the number correlator dumps and runs from 0 to 19 and rolls over. The algorithm used in this routine as shown in figure 4-14 sets this counter to 0 when the carrier phase changes from +-90 degrees with a tolerance of ~10 degrees to a carrier phase of -+90 degrees also with a tolerance of ~10 degrees. As shown in the state diagram a transition to state 4 or tracking is not allowed unless the ms counter is set.

MORE STUFF NEEDED

Figure 4-13
Bit Synchronization

```
void ch_pull_in(char ch)
{
   long ddf,ddcar,theta_e,wdot_gain;
   long q_sum,i_sum,theta,theta_dot;
   long prompt_mag,dith_mag;
   prompt_mag=rss(chan[ch].i_prompt,chan[ch].q_prompt);
   dith_mag  =rss(chan[ch].i_dith,chan[ch].q_dith);
   chan[ch].sum+=prompt_mag+dith_mag;
//
// code tracking loop
//
   if ( prompt_mag != 0 || dith_mag != 0)
   {
      chan[ch].dfreq=((prompt_mag - dith_mag)<<14)
                                         /(prompt_mag+dith_mag)*pull_code_k;
      ddf  =(chan[ch].dfreq-chan[ch].dfreq1)*pull_code_d;
      if ( chan[ch].ch_time > 2 )            // don't close the code loop
      {                                      // for 2 ms to avoid transients
            chan[ch].code_freq =((chan[ch].dfreq+ddf)>>14)+chan[ch].code_freq;
            ch_code(ch,chan[ch].code_freq);
      }
   }
   chan[ch].dfreq1=chan[ch].dfreq;
//
// carrier tracking loop
//
   q_sum=chan[ch].q_dith+chan[ch].q_prompt;
   i_sum=chan[ch].i_dith+chan[ch].i_prompt;
   if (i_sum !=0 || q_sum !=0) theta=fix_atan2(q_sum,-i_sum);
   else theta=chan[ch].old_theta;
   theta_dot=theta-chan[ch].old_theta;
   chan[ch].ms_count++;
//
// check to see if we are at the edge of a data bit
//
   if ( sign(q_sum)==-sign(chan[ch].old_q_sum) && (a_missed & test[ch])==0
```

```
          && labs(labs(theta)-25736)<4096
          && labs(labs(chan[ch].old_theta)-25736)<4096)
{
    chan[ch].ms_count=0;       // if yes set the counter to 1
    ch_epoch_load(ch,0x1);     // load it to ms counter on GP 2021
    chan[ch].ms_set=1;         // set flag to say we have bit sync
}
chan[ch].old_theta = theta;
if      ( theta> 0 ) theta_e = theta-25736;
else if ( theta<= 0) theta_e = theta+25736;
if (chan[ch].ch_time>pull_in_time-phase_test) chan[ch].th_rms+=
                                                 (theta_e*theta_e)>>14;
if ( labs(theta_dot) < 32768L )
{
    if (q_sum != 0 || i_sum !=0)
    {
       wdot_gain=chan[ch].ch_time/499;
       wdot_gain*=wdot_gain;
       wdot_gain*=wdot_gain;
       chan[ch].dcarr=pull_carr_k*(theta_dot*5/(1+wdot_gain)+theta_e);
       ddcar=(chan[ch].dcarr-chan[ch].dcarr1)*pull_carr_d;
       if ( chan[ch].ch_time > 5 )        // don't close the loop for
       {                                  // 5 ms to avoid transients
             chan[ch].carrier_freq = ((chan[ch].dcarr+ddcar)>>14)+
                                     chan[ch].carrier_freq;
             ch_carrier(ch,chan[ch].carrier_freq);
       }
   }
}
chan[ch].dcarr1=chan[ch].dcarr;
chan[ch].old_q_sum=q_sum;
chan[ch].ch_time++;
chan[ch].ms_count=chan[ch].ms_count%20;
if (chan[ch].ch_time>=pull_in_time && chan[ch].ms_count==19)   // done with
                                                              // pull-in
{                                                       // wait until end of
    chan[ch].avg=chan[ch].sum/pull_in_time/2;          // a data bit
    chan[ch].th_rms=fix_sqrt(chan[ch].th_rms/phase_test);
    if ( chan[ch].avg>14*rms/10 && chan[ch].th_rms<12000 && chan[ch].ms_set)
    {                                      // go to track
            chan[ch].avg=chan[ch].avg*20;     // if code and carrier track
            chan[ch].state=track;             // and the ms counter is set
            chan[ch].t_count=0;
            chan[ch].sum=0;
            chan[ch].q_dith_20=0;
            chan[ch].q_prom_20=0;
            chan[ch].i_dith_20=0;
            chan[ch].i_prom_20=0;
            if (ch==0)
            {
                 debug_counter=0;
                 store_code=chan[ch].code_freq;
                 store_carrier=chan[ch].carrier_freq;
            }
            chan[ch].dfreq=0;
    }
    else                                   // else go back to acquisition
```

```
    {
            chan[ch].state=acquisition;
            chan[ch].codes=0;
            chan[ch].code_freq=code_ref+code_corr;
            ch_code(ch,chan[ch].code_freq);        // 1.023 MHz chipping rate
    }
  }
}
```

4.4.5 Tracking State

The tracking state is very similar to pull-in state except for two areas. The carrier loop commands are computed at 1000 Hz and the code loop commands are computed at 50 Hz. Also, whenever a code loop command is computed the data bit is determined and sent to the pream routine which finds the preamble in the navigation message and synchronizes up to the TOW message and checks the subframe number.

MORE STUFF NEEDED

4.4.5.1 Code Tracking

As can be seen in figure 4-15 the tracking state code tracking loop is very similar to the pull-in state tracking loop. The addition of carrier aiding allows the bandwidth to be decreased from xx Hz in the pull-in state to xx Hz. Since the code tracking loop will lose lock when carrier lock is lost this does not appear to penalize the performance.

MORE STUFF NEEDED

Figure 4-15
Tracking State Code Tracking Loop

Since this is the loop used when measurements are made an analysis of the noise tracking is presented.

As given in reference 1

$$\sigma_{tDLL} = \lambda_c \sqrt{\frac{2F_1 d^2 B_n}{c/n_0}\left[2(1-d)+\frac{4F_2 d}{Tc/n_0}\right]}$$

Where:

σ_{tDLL}	=	1-sigma thermal noise code tracking jitter
F1	=	DLL discriminator correlator factor (dimensionless)
	=	1 for time shared tau-dithered early/late correlator
	=	½ for dedicated early and late correlators
d	=	correlator spacing between early, prompt, and late (chips)
Bn	=	code loop noise bandwidth (Hz)
c/n0	=	carrier to noise power expressed as a ratio
	=	$10^{C/No/10}$ for C/No in dB-Hz
T	=	predetection integration time (sec)
F2	=	DLL discriminator type factor (dimensionless)
	=	1 for early/late type discriminator
	=	½ for dot product type discriminator
λc	=	the code length of 293.05 m/chip for C/A-code

The GP2021 chipset along with OpenSourceGPS uses an F1 of 1/2, d = 1/2 , and F2 = 1. The other parameters are …

Figure 4-16 shows the expected performance of OpenSourceGPS code tracking jitter vs C/No

Figure 4-16
Code Tracking jitter vs C/No

4.4.5.2 Carrier Tracking

Since this is the loop used when measurements are made an analysis of the noise tracking is presented. As given in reference xx:

$$\sigma_{tPLL} = \frac{\lambda_L}{2\pi}\sqrt{\frac{B_n}{c/n_0}\left[1+\frac{1}{2T\,c/n_0}\right]}$$

Where:

σ_{tPLL} = 1-sigma thermal noise code tracking jitter
Bn = carrier loop noise bandwidth (Hz)
c/n_0 = carrier to noise power expressed as a ratio
= $10^{C/No/10}$ for C/No in dB-Hz
T = predetection integration time (sec)
λ_L = the carrier frequency wave length of 0.1903 m/cycle for L1

Figure 4-17 shows the expected performance of OpenSourceGPS carrier tracking jitter vs C/No

MORE STUFF NEEDED

Figure 4-17
Carrier Tracking jitter vs C/No

4.4.5.3 Message Synchronization

Message synchronization starts by detecting the message preamble. To do this it uses a 62 bit split and offset fifo data structure. Figure 4-18 illustrates how this was set up. Two 32 bit registers are linked at the 30th bit of first register. This allows 2 message words to be checked for parity and have enough data to analyze the TLM and HOW words. The routine uses a set of nested comparisons to reduce the computation time. The first 8 bits of the TLM word is the preamble and the first 17 bits of the HOW word is a truncated TOW Count. Bits 20-22 of the HOW word are the subframe ID.

MORE STUFF NEEDED

Figure 4-18
Preamble Detection

In order to save time the preamble is checked in several stages.

The algorithm is a follows:

Check for preamble or inverse of preamble bits
- Check parity of word 0
  - Check parity of word 1
    - Check subframe number (1-5)
      - Check TOW against receiver time

When all of these checks pass we know that the current data bit is bit 59 of the nav message (counting from 0).

The next thing we set is the transmission bit time into the week. Since the message time is the start of the next subframe we set the bit time to 240 bits less than the TOW.

The navigation message itself is continuously recorded into a circular register. When the subframe number is found we can determine the offset bit number in this register. This value tells the navigation message decoding routine where the message starts.

```
void pream(char ch,char bit)
{
  static unsigned long pream=0x22c00000L;
  unsigned long  parity0,parity1;
  static unsigned long pb1=0xbb1f3480L,pb2=0x5d8f9a40L,pb3=0xaec7cd00L;
  static unsigned long pb4=0x5763e680L,pb5=0x6bb1f340L,pb6=0x8b7a89c0L;
  unsigned long TOWs,HOW,TLM,TLMs;
  int sfid_s;

  if (chan[ch].fifo1&0x20000000L)
  {
        chan[ch].fifo0=(chan[ch].fifo0<<1)+ 1;
  }
  else
  {
        chan[ch].fifo0=chan[ch].fifo0<<1;
  }

  chan[ch].fifo1=(chan[ch].fifo1<<1)+bit;

  if (chan[ch].fifo0&0x40000000L)
  {
        TLM = chan[ch].fifo0^0x3fffffc0L;
  }
  else
  {
        TLM = chan[ch].fifo0;
  }

  if (chan[ch].fifo1&0x40000000L)
  {
        HOW = chan[ch].fifo1^0x3fffffc0L;
  }
  else
  {
        HOW = chan[ch].fifo1;
  }

  if (((pream^TLM)&0x3fc00000L)==0)   // preamble pattern found?
  {

        parity0=(xors(TLM & pb1)<<5)+(xors(TLM & pb2)<<4)+
                    (xors(TLM & pb3)<<3)+(xors(TLM & pb4)<<2)+
                    (xors(TLM & pb5)<<1)+(xors(TLM & pb6));

        if (parity0==(TLM & 0x3f))    // is parity of TLM ok?
        {
              parity1=(xors(HOW & pb1)<<5)+(xors(HOW & pb2)<<4)+
                                 (xors(HOW & pb3)<<3)+(xors(HOW & pb4)<<2)+
                                 (xors(HOW & pb5)<<1)+(xors(HOW & pb6));

              if (parity1==(HOW & 0x3f))  // is parity of HOW ok?
```

```
            {
                 sfid_s=int((HOW & 0x700)>>8); // compute the subframe id
                                               // number
                 TLMs=(TLM>>2) & 0x3fff;
                 if (sfid_s==1)      // synchronize on subframe 1 if TOW matches
                 {                   // clock within 300 seconds
                   TOWs =(HOW & 0x3fffffffL)>>13;
                   d_tow=clock_tow-TOWs*6+5; // +5 since TOW is 6 sec ahead
                   if ( labs(d_tow)<300)
                   {
                       chan[ch].offset=chan[ch].t_count-59;
                       ch_epoch_load(ch,(0x1f&ch_epoch_chk(ch))|0xa00);  ///cwk
                       if (chan[ch].offset<0.0) chan[ch].offset+=1500;
                       chan[ch].tr_bit_time=TOWs*300-240;
                       chan[ch].TOW=TOWs*6;
                       chan[ch].tow_sync=1;
                       thetime=thetime-d_tow;
                       clock_tow=TOWs*6-5;
                       chan[ch].sfid=sfid_s;
                       chan[ch].TLM=TLMs;
                   }
                 }
// allow resync on other subframes if TOW matches clock to 3 seconds
// this should improve the re-acquisition time
                 else if (sfid_s>1 && sfid_s<6)
                 {
                   TOWs =(HOW & 0x3fffffffL)>>13;
                   d_tow=clock_tow-TOWs*6+5;  // +5 since TOW is 6 sec ahead
                   if ( labs(d_tow)<3)
                   {
                       chan[ch].offset=chan[ch].t_count-59-(sfid_s-1)*300;
                       ch_epoch_load(ch,(0x1f&ch_epoch_chk(ch))|0xa00);
                       if (chan[ch].offset<0.0) chan[ch].offset+=1500;
                       chan[ch].tr_bit_time=TOWs*300-240;
                       chan[ch].tow_sync=1;
                       chan[ch].TOW=TOWs*6;
                       chan[ch].sfid=sfid_s;
                       chan[ch].TLM=TLMs;
                   }
              }
            }
        }
  }
//  a 1500 bit frame of data is ready to be read
  if ((chan[ch].t_count-chan[ch].offset)%1500==0) chan[ch].frame_ready=1;
}
```

4.4.5.4 Ch_track listing

```
void ch_track(char ch)
{
     long ddf,ddcar,q_sum,i_sum;

     chan[ch].ms_count=(++chan[ch].ms_count)%20;   //  Software ms count
     chan[ch].q_dith_20+=chan[ch].q_dith;          //  20 ms summed q_dith
```

```
     chan[ch].q_prom_20+=chan[ch].q_prompt;        //  20 ms summed q_prompt
     chan[ch].i_dith_20+=chan[ch].i_dith;          //  20 ms summed i_dith
     chan[ch].i_prom_20+=chan[ch].i_prompt;        //  20 ms summed i_prompt

//
//  now the carrier phase locked loop
//
     q_sum=chan[ch].q_dith+chan[ch].q_prompt;
     i_sum=chan[ch].i_dith+chan[ch].i_prompt;
     if ( q_sum != 0 || i_sum != 0)
    {

chan[ch].dcarr=(i_sum<<14)*trk_carr_k*sign(q_sum)/rss(q_sum,i_sum);//check
       ddcar=(chan[ch].dcarr-chan[ch].dcarr1)*trk_carr_d;          //here
       chan[ch].carrier_freq=((chan[ch].dcarr+ddcar)>>14)+chan[ch].carrier_freq;
       ch_carrier(ch,chan[ch].carrier_freq);
     }
     chan[ch].old_q_sum=q_sum;
     chan[ch].dcarr1=chan[ch].dcarr;
//
     if (chan[ch].ms_count==19)  // process code loop and data bit
    {
       chan[ch].tr_bit_time++;
       chan[ch].prompt_mag=rss(chan[ch].i_prom_20,chan[ch].q_prom_20);
       chan[ch].dith_mag  =rss(chan[ch].i_dith_20,chan[ch].q_dith_20);
       chan[ch].sum+=chan[ch].prompt_mag+chan[ch].dith_mag;
// code tracking loop
       if ( chan[ch].prompt_mag != 0 || chan[ch].dith_mag != 0)
       {

// with carrier aiding
            ddf=((chan[ch].prompt_mag-chan[ch].dith_mag)*2048)/
            (chan[ch].prompt_mag+chan[ch].dith_mag)*trk_code_k;
            chan[ch].dfreq+=ddf;
            chan[ch].code_freq =(chan[ch].dfreq/trk_code_d+ddf)/256+
            (carrier_ref-chan[ch].carrier_freq)/cc_scale +code_ref;   // here
was >>12
            ch_code(ch,chan[ch].code_freq);
       }
       chan[ch].dfreq1=chan[ch].dfreq;

       chan[ch].bit=bsign(chan[ch].q_prom_20+chan[ch].q_dith_20);//bsign is data
bit
       pream(ch,chan[ch].bit);   // see if we can find the preamble
       chan[ch].message[chan[ch].t_count]=chan[ch].bit;
       if ( ch==0 && debug_counter<4500)
       {
            i_prompta[debug_counter]=ddf;
            q_prompta[debug_counter]=chan[ch].dfreq;
            i_dithera[debug_counter]=chan[ch].prompt_mag;
            q_dithera[debug_counter]=chan[ch].dith_mag;
            car_freq[debug_counter]=chan[ch].carrier_freq;
            chip_freq[debug_counter]=chan[ch].code_freq;
            data_bit[debug_counter]=chan[ch].bit;
            debug_counter++;
       }
       chan[ch].t_count++;
```

```
        if (chan[ch].t_count%5==0)
        {
              chan[ch].avg=chan[ch].sum/10;
              chan[ch].sum=0;
        }
        chan[ch].q_dith_20=0;
        chan[ch].q_prom_20=0;
        chan[ch].i_dith_20=0;
        chan[ch].i_prom_20=0;
   }
   if (chan[ch].t_count==1500)
   {
        chan[ch].n_frame++;
        chan[ch].t_count=0;
   }
}
```

4.4.6 Miscellaneous Functions

4.4.6.1 RSS

The RSS or Root Sum Square is used to determine the amplitude of the signal which has been separated into its inphase and quadrature components. This is needed during the pull-in state when the phase is unknown, in the tracking state it is no longer needed since the phase is driven to produce a maximum value of the inphase component while the quadrature component is used as the error signal. A number of methods have been documented. The method used in OpenSourceGPS is the xx method

$$RSS \approx \max(a,b) + \min(a,b)/2$$

```
/*******************************************************************************
FUNCTION rss(long a, long b)
RETURNS  long integer

PARAMETERS
      a  long integer
            b  long integer

PURPOSE
      This function finds the fixed point magnitude of a 2 dimensional vector

WRITTEN BY
      Clifford Kelley

*******************************************************************************/

//inline long rss(long a,long b )
long rss(long a,long b)
{
   long result,c,d;
   c=labs(a);
   d=labs(b);
```

```
    if (c==0 && d==0) result=0;
    else
    {
        if (c>d) result=(d>>1)+c;
        else     result=(c>>1)+d;
    }
    return (result);
}
```

As illustrated in Figure 4-19 this method produces a maximum error of

MORE STUFF NEEDED

Figure 4-19
RSS Algorithm Error

4.4.6.2 fix Sqrt

The Sqrt algorithm uses Newtons method for a 32 bit long integer. It iteratively solves this equation.

$$dx = \frac{R - x * x}{2}$$

```
/*******************************************************************************
FUNCTION fix_sqrt(long x)
RETURNS  long integer

PARAMETERS
      x long integer
```

```
PURPOSE
      This function finds the fixed point square root of a long integer

WRITTEN BY
      Clifford Kelley

******************************************************************************/

long fix_sqrt(long x)
{
  long xt,scr;
  int i;
  i=0;
  xt=x;
  do
  {
     xt=xt>>1;
     i++;
  } while (xt>0);
  i=(i>>1)+1;
  xt=x>>i;
  do
  {
     scr=xt*xt;
     scr=x-scr;
     scr=scr>>1;
     scr=scr/xt;
     xt=scr+xt;
  } while (scr!=0);
  xt=xt<<7;
  return(xt);
}
```

4.4.6.3 fix atan2

The pull-in state uses a two dimensional arc tangent function that computes the phase of the signal by using the inphase and quadrature components. The basic function is a quadrant dependent version of this equation:

$$\theta = x - \frac{2x^3}{9}$$

While this is not a taylor power series expansion it has a smaller error than the equivalent truncated taylor power series. As illustrated in Figure 4-20 this approximation produces an error of less than x% at its worst.

MORE STUFF NEEDED

Figure 4-20
Fix Atan2 Error Function

```
/*******************************************************************************
FUNCTION fix_atan2(long y,long x)
RETURNS  long integer

PARAMETERS
            x  long   in-phase fixed point value
            y  long   quadrature fixed point value

PURPOSE
      This function computes the fixed point arctangent represented by
      x and y in the parameter list
      1 radian = 16384
      based on the power series  f-f^3*2/9

WRITTEN BY
      Clifford Kelley
      Fixed for y==x  added special code for x==0 suggested by Joel Barnes, UNSW
*******************************************************************************/
long fix_atan2(long y,long x)
{
  long result,n,n3;
  if ((x==0) && (y==0))
        return(0); // invalid case

  if (x>0 &&  x>=labs(y))
  {
             n=(y<<14)/x;
             n3=((((n*n)>>14)*n)>>13)/9;
             result=n-n3;
  }
```

```
  else if (x<=0 && -x>=labs(y))
  {
            n=(y<<14)/x;
            n3=((((n*n)>>14)*n)>>13)/9;
            if      ( y>0) result=n-n3+SCALED_PI;
            else if (y<=0) result=n-n3-SCALED_PI;
  }
  else if (y>0 &&  y>labs(x))
  {
            n=(x<<14)/y;
            n3=((((n*n)>>14)*n)>>13)/9;
            result=SCALED_PI_ON_2-n+n3;
  }
  else if (y<0 && -y>labs(x))
  {
            n=(x<<14)/y;
            n3=((((n*n)>>14)*n)>>13)/9;
            result=-n+n3-SCALED_PI_ON_2;
  }
  return(result);
}
```

4.7 Position/Velocity/Time Computations

4.7.1 Satellite Location Algorithms

4.7.1.1 Almanac

As mentioned before the almanac is low accuracy orbit description used to calculate what satellites are expected to be visible for channel/PRN allocation.

```
/*****************************************************************************
FUNCTION satpos_almanac(float time, char n)
RETURNS  None.

PARAMETERS
                  time   float  time of week
                  n      char   satellite prn

PURPOSE

        THIS SUBROUTINE CALCULATES THE SATELLITE POSITION
        BASED ON ALMANAC DATA

     R    - RADIUS OF SATELLITE AT TIME T
     SLAT - SATELLITE LATITUDE
     SLONG- SATELLITE LONGITUDE
     T    - TIME FROM START OF WEEKLY EPOCH
     ETY  - ORBITAL ECCENTRICITY
     TOA  - TIME OF APPLICABILITY FROM START OF WEEKLY EPOCH
     INC  - ORBITAL INCLINATION
     RRA  - RATE OF RIGHT ASCENSION
     SQA  - SQUARE ROOT OF SEMIMAJOR AXIS
     LAN  - LONGITUDE OF NODE AT WEEKLY EPOCH
     AOP  - ARGUMENT OF PERIGEE
     MA   - MEAN ANOMALY AT TOA

WRITTEN BY
      Clifford Kelley

*****************************************************************************/

ecef satpos_almanac( float time, char n)
{
      double ei,ea,diff,r,ta,la,aol,xp,yp,d_toa;
      ecef result;
/*
      MA IS THE ANGLE FROM PERIGEE AT TOA
*/
      d_toa=time-gps_alm[n].toa;
      if (d_toa>302400.0) d_toa=d_toa-604800.0;
      else if (d_toa<-302400.0)d_toa=d_toa+604800.0;
      ei=gps_alm[n].ma+d_toa*gps_alm[n].w;
      ea=ei;
      do
      {
         diff=(ei-(ea-gps_alm[n].ety*sin(ea)))/(1.-gps_alm[n].ety*cos(ea));
```

```
            ea=ea+diff;
        } while (fabs(diff) > 1.0e-6);
/*
        EA IS THE ECCENTRIC ANOMALY
*/
        if (gps_alm[n].ety != 0.0 )
        ta=atan2(sqrt(1.-pow(gps_alm[n].ety,2))*sin(ea),cos(ea)-gps_alm[n].ety);
        else
        ta=ea;
/*
        TA IS THE TRUE ANOMALY (ANGLE FROM PERIGEE)
*/
        r=pow(gps_alm[n].sqa,2)*(1.-pow(gps_alm[n].ety,2)*cos(ea));
/*
        R IS THE RADIUS OF SATELLITE ORBIT AT TIME T
*/
        aol=ta+gps_alm[n].aop;
/*
        AOL IS THE ARGUMENT OF LATITUDE

              LA IS THE LONGITUDE OF THE ASCENDING NODE
*/
        la=gps_alm[n].lan+(gps_alm[n].rra-omegae)*d_toa-gps_alm[n].toa*omegae;
        xp=r*cos(aol);
        yp=r*sin(aol);
        result.x=xp*cos(la)-yp*cos(gps_alm[n].inc)*sin(la);
        result.y=xp*sin(la)+yp*cos(gps_alm[n].inc)*cos(la);
        result.z=yp*sin(gps_alm[n].inc);
        return(result);
}
```

4.7.1.2 Ephemeris

The ephemeris routine computes the precise position of the satellite in ecef and the satellites atomic clock offset from GPS time. The methodology is similar to the almanac routine with the addition of more second order corrections.

```
/********************************************************************************
FUNCTION satpos_ephemeris(double t, char n)
RETURNS  None.

PARAMETERS
                  t  double   time of week
                  n  char     satellite prn

PURPOSE

     THIS SUBROUTINE CALCULATES THE SATELLITE POSITION
     BASED ON BROADCAST EPHEMERIS DATA

     R    - RADIUS OF SATELLITE AT TIME T
     Crc  - RADIUS COSINE CORRECTION TERM
     Crs  - RADIUS SINE   CORRECTION TERM
     SLAT - SATELLITE LATITUDE
```

```
     SLONG- SATELLITE LONGITUDE
     TOE  - TIME OF EPHEMERIS FROM START OF WEEKLY EPOCH
     ETY  - ORBITAL INITIAL ECCENTRICITY
     TOA  - TIME OF APPLICABILITY FROM START OF WEEKLY EPOCH
     INC  - ORBITAL INCLINATION
     IDOT - RATE OF INCLINATION ANGLE
     CUC  - ARGUMENT OF LATITUDE COSINE CORRECTION TERM
     CUS  - ARGUMENT OF LATITUDE SINE   CORRECTION TERM
     CIC  - INCLINATION COSINE CORRECTION TERM
     CIS  - INCLINATION SINE   CORRECTION TERM
     RRA  - RATE OF RIGHT ASCENSION
     SQA  - SQUARE ROOT OF SEMIMAJOR AXIS
     LAN  - LONGITUDE OF NODE AT WEEKLY EPOCH
     AOP  - ARGUMENT OF PERIGEE
     MA   - MEAN ANOMALY AT TOA
     DN   - MEAN MOTION DIFFERENCE

WRITTEN BY
      Clifford Kelley

*******************************************************************************/

eceft satpos_ephemeris(double t,char n)
{
      double ei,ea,diff,ta,aol,delr,delal,delinc,r,inc;
      double la,xp,yp,bclk,tc,d_toc,d_toe;
      double xls,yls,zls,range1,tdot,satang,xaz,yaz;
      double az;
      ecef north,east,up;
      eceft result;
//
//     MA IS THE ANGLE FROM PERIGEE AT TOA
//
      d_toc=t-gps_eph[n].toc;
      if (d_toc>302400.0) d_toc=d_toc-604800.0;
      else if (d_toc<-302400.0)d_toc=d_toc+604800.0;
      bclk=gps_eph[n].af0+gps_eph[n].af1*d_toc+gps_eph[n].af2*d_toc*d_toc
        -gps_eph[n].tgd;
      tc=t-bclk;
      d_toe=tc-gps_eph[n].toe;
      if (d_toe>302400.0) d_toe=d_toe-604800.0;
            else if (d_toe<-302400.0)d_toe=d_toe+604800.0;
            ei=gps_eph[n].ma+d_toe*(gps_eph[n].wm+gps_eph[n].dn);
      ea=ei;
      do
      {
       diff=(ei-(ea-gps_eph[n].ety*sin(ea)))/(1.0E0-gps_eph[n].ety*cos(ea));
       ea=ea+diff;
            } while (fabs(diff) > 1.0e-12 );
      bclk=bclk-4.442807633E-10*gps_eph[n].ety*gps_eph[n].sqra*sin(ea);
            result.tb=bclk;
//
//     ea is the eccentric anomaly
//
      ta=atan2(sqrt(1.00-pow(gps_eph[n].ety,2))*sin(ea),cos(ea)-gps_eph[n].ety);
//
//     TA IS THE TRUE ANOMALY (ANGLE FROM PERIGEE)
```

```
//
      aol=ta+gps_eph[n].w;
//
//     AOL IS THE ARGUMENT OF LATITUDE OF THE SATELLITE
//
//     calculate the second harmonic perturbations of the orbit
//
      delr  =gps_eph[n].crc*cos(2.0*aol)+gps_eph[n].crs*sin(2.0*aol);
      delal =gps_eph[n].cuc*cos(2.0*aol)+gps_eph[n].cus*sin(2.0*aol);
      delinc=gps_eph[n].cic*cos(2.0*aol)+gps_eph[n].cis*sin(2.0*aol);
//
//     R IS THE RADIUS OF SATELLITE ORBIT AT TIME T
//
      r=pow(gps_eph[n].sqra,2)*(1.00-gps_eph[n].ety*cos(ea))+delr;
      aol=aol+delal;
      inc=gps_eph[n].inc0+delinc+gps_eph[n].idot*d_toe;
//
//     LA IS THE CORRECTED LONGITUDE OF THE ASCENDING NODE
//
      la=gps_eph[n].w0+(gps_eph[n].omegadot-omegae)*d_toe-
                    omegae*gps_eph[n].toe;
      xp=r*cos(aol);
      yp=r*sin(aol);
      result.x=xp*cos(la)-yp*cos(inc)*sin(la);
      result.y=xp*sin(la)+yp*cos(inc)*cos(la);
      result.z=yp*sin(inc);
      result.az=0.0;
      result.el=0.0;
      if (rec_pos_xyz.x != 0.0 || rec_pos_xyz.y != 0.0 || rec_pos_xyz.z != 0.0)
      {
/*
      CALCULATE THE POSITION OF THE RECEIVER
*/
      north.x=-cos(rec_pos_llh.lon)*sin(rec_pos_llh.lat);
      north.y=-sin(rec_pos_llh.lon)*sin(rec_pos_llh.lat);
      north.z= cos(rec_pos_llh.lat);
      east.x=-sin(rec_pos_llh.lon);
      east.y= cos(rec_pos_llh.lon);
      east.z=0.0;
      up.x=cos(rec_pos_llh.lon)*cos(rec_pos_llh.lat);
      up.y=sin(rec_pos_llh.lon)*cos(rec_pos_llh.lat);
      up.z=sin(rec_pos_llh.lat);
/*
     DETERMINE IF A CLEAR LINE OF SIGHT EXISTS
*/
      xls =result.x-rec_pos_xyz.x;
      yls =result.y-rec_pos_xyz.y;
      zls =result.z-rec_pos_xyz.z;
      range1=sqrt(xls*xls+yls*yls+zls*zls);
      tdot=(up.x*xls+up.y*yls+up.z*zls)/range1;

      if ( tdot >= 1.00 ) satang=pi/2.0;
      else if ( tdot <= -1.00 ) satang=-pi/2.0;
      else satang=asin(tdot);

      xaz=east.x*xls+east.y*yls;
      yaz=north.x*xls+north.y*yls+north.z*zls;
```

```
        if (xaz !=0.0 || yaz !=0.0) az=atan2(xaz,yaz);
        else az=0.0;
        result.el=satang;
        result.az=az;
    }
    return(result);
}
```

4.7.2 Range corrections

4.7.2.1 Troposphere

The lower part of the atmosphere (the Troposphere) has an index of refraction slightly more that 1.0 which is largely independent of the frequency. This causes the radio signal to slow down and thus makes the range measurement larger than it actually is. For the most part this effect is predictable and a number of models have been used to compensate for it.

The model used in OpenSourceGPS is:

$$Dt_{tropo} = \frac{2.47e^{-0.000133*HAE}}{\sin(el)+0.0121}$$

This is based on Black and xx in reference xx.

The magnitude and trend of this correction is apparent in figure 4-21.

MORE STUFF NEEDED

Figure 4-21
Troposphere Correction

4.7.2.2 Ionosphere

In addition the charged particles in the upper part of the atmosphere (the Ionosphere) also slow down the signal. In this case the delay is a function of frequency and if two frequency measurements are available this delay can be measured with high precision. Since L2 is not available to this receiver we use the ionopheric correction data provided in the navigation message. This data is provided in page 51 of the almanac.

```
double tropo_iono(char ch,float az,float el,double gps_time)
{
  double d_Trop,alt_factor;
  double d_Ion,psi,phi,lambdai,phim,per,x,F,amp,t;

//  Try a simple troposphere model
  if (current_loc.hae>200000.0) alt_factor=0.0;
  else if (current_loc.hae<0.0) alt_factor=1.0;
  else alt_factor=exp(-current_loc.hae*1.33e-4);
  if (m_tropo==1) d_Trop=2.47/(sin(el)+.0121)*alt_factor/c;
  else d_Trop=0.0;
  tropo[ch]=d_Trop;
  if (d_Trop<0.0) printf("el=%lf, hae=%lf",el,current_loc.hae);

//  Use an ionosphere model
  if (m_iono==1)
  {
       psi=0.0137/(el/pi+0.11)-.022;
       phi=current_loc.lat/pi+psi*cos(az);
       if      (phi > 0.416) phi= 0.416;
       else if (phi <-0.416 )phi=-0.416;
       lambdai=current_loc.lon/pi+psi*sin(az)/cos(phi*pi);
       t=43200.0*lambdai+gps_time-int((43200.0*lambdai+gps_time)/86400.)*86400.;
       if (t<0.0) t=t+86400.0;
       phim=phi+0.064*cos((lambdai-1.617)*pi);
//
//  If available from the nav message use its Ionosphere model
//
       if (b0 != 0.0 && al0 != 0.0)
       {
            per=b0+b1*phim+b2*phim*phim+b3*phim*phim*phim;
            amp=al0+al1*phim+al2*phim*phim+al3*phim*phim*phim;
       }
//
// else try this set of default iono model parameters
//
       else
       {
            per=b0+b1*phim+b2*phim*phim+b3*phim*phim*phim;
            amp=al0+al1*phim+al2*phim*phim+al3*phim*phim*phim;
       }
       if ( per <72000.0 ) per=72000.0;
       x=2.*pi*(t-50400.)/per;
       F=1.0+16.0*pow(0.53-el/pi,3);
       if ( amp < 0.0 ) amp=0.0;
       if (fabs(x) < 1.5707)  d_Ion=F*(5.0e-9+amp*(1.0-x*x/2.+x*x*x*x/24.0));
       else                   d_Ion=F*5.0e-9;
```

```
  }
  else d_Ion=0.0;
  iono[ch]=d_Ion;
  return(d_Trop+d_Ion);
}
```

4.7.2.3 Satellite clock

The satellite clocks are allowed to run freely with an occasional reset when the clock drifts far enough off that the navigation message cannot correct for it.
The navigation message gives the offset AF0 and linear drift AF1 of the satellite atomic clock with respect to GPS time. The navigation message includes a time acceleration term but it's purpose was to take into account the relativistic effects. Since the equation is easy to implement the receiver is expected to compute it and AF2 is set to zero.

MORE STUFF NEEDED

4.7.3 Other effects

4.7.3.1 Sagnac effect

The Sagnac effect is the consequence of computing positions in a rotating coordinate system as if they were occurring in an inertial coordinate system. In a GPS receiver this effect occurs when computing the position of the satellites and receiver in ECEF. Since the radio signal was transmitted a number of milliseconds before being received the satellite position must be rotated back to match the coordinate systems at the time when the signals are received.

MORE STUFF NEEDED

4.7.3.2 Relativistic effects

While a number of relativistic effects produce measurable errors the one explicitly taken into account of in OpenSourceGPS is the clock error produced by the gravitational potential as the satellite moves through it orbit.

MORE STUFF NEEDED

4.7.4 Computing Pseudorange

When a TIC occurs the state of the PRN code registers is latched and read whenever a navigation fix is scheduled.

The transmission time from the satellite is determined by adding the number of data bits into the week to the PRN code counter and the measured phase

Before the pseudorange is used to compute position it is subjected to a number of integrity checks. Early in the development of OpenSourceGPS a number of problems were encountered where the code register data were out of bounds. This was later isolated to data bus loading problems. While these checks are still in the code the primary integrity problem is the PRN code ambiguity. This condition occurs about 1 in every 10,000 pseudorange measurements. Fortunately though it is easy to detect. As described in the GP2021 manual page xx this is due to the fact that the C/A code only has 1023 code states which is not an integer power of 2. As illustrated in figure 4-22 when the PRN code generator resets from 2045 -> 0 there is a slight delay and instead of a measurement of 2046 is read. When this occurs the measurement from the channel is ignored.

MORE STUFF NEEDED

4.7.5 Computing Delta-Pseudorange

Once the position is computed the velocity is computed using either the carrier trackin loop DCO setting (CTL) or the integrated carrier phase or ICP.

MORE STUFF NEEDED

```
/*******************************************************************************
FUNCTION nav_fix()
RETURNS  None.

PARAMETERS None.

PURPOSE
      This function determines the pseudorange and doppler to each
      satellite and calls pos_vel_time to determine the position and
      velocity of the receiver

WRITTEN BY
      Clifford Kelley

*******************************************************************************/

void  nav_fix(void)
{
   char   ch,n,bit;
   double tr_time[13],tr_avg,ipart,clock_error;
   static double t_cor[13];
```

```
   unsigned int    i,ms,chip,phase;
   int meas_bit_time_rem;
   long meas_bit_time_offset;
   ecef   rp_ecef;
   eceft  dm_gps_sat[13],dp_gps_sat[13];

   tr_avg=0.0;
   n=1;
   for (ch=0;ch<=11;ch++)
   {
       meas_bit_time_offset=0;
       ms=chan[ch].epoch & 0x1f;
       chip=chan[ch].code_phase;
       phase=chan[ch].code_dco_phase;
       bit=chan[ch].epoch>>8;
       chan[ch].int_carr_phase=chan[ch].carrier_counter+
                                         float(chan[ch].d_carr_phase)/1024.;
       if (out_debug)
       {
                fprintf(debug," ch= %d,%d,%ld,%d,%d,%d,%d,%d,%d,%d,%d,%f,",

ch,chan[ch].prn,chan[ch].meas_bit_time,bit,ms,chip,phase,chan[ch].state,
          gps_eph[chan[ch].prn].valid,gps_eph[chan[ch].prn].health,
          chan[ch].tow_sync,chan[ch].CNo);
          if (ICP_CTL==0) fprintf(debug,"%ld\n",chan[ch].doppler);
          else            fprintf(debug,"%lf\n",chan[ch].int_carr_phase);
       }
//  Use only satellites being tracked with valid ephemeris data
//  (valid subframe 1,2,3), good health,high enough raw snr, and
//  valid phase data

       meas_bit_time_rem = chan[ch].meas_bit_time%50;
       if ( meas_bit_time_rem == bit+1 ) meas_bit_time_offset = -1;
       if ( meas_bit_time_rem == bit-1 ) meas_bit_time_offset = +1;
       if ( meas_bit_time_rem == 0 && bit == 49 ) meas_bit_time_offset = -1;
       if ((chan[ch].meas_bit_time+meas_bit_time_offset)%50==bit &&
           chan[ch].state==track && chan[ch].CNo>33 &&
                    gps_eph[chan[ch].prn].valid==1 &&
                    gps_eph[chan[ch].prn].health==0 &&
                    chan[ch].tow_sync==1 && phase<1024 && chip<2046)
       {
          tr_time[n]= (chan[ch].meas_bit_time + meas_bit_time_offset)*.02 +
                                     ms/1000.0+chip/2.046e6+phase/2.046e6/1024.;
             tr_ch[n]=ch;
             tr_avg+=tr_time[n];
             n++;
       }
}
n_track=n-1;
TIC_dt=i_TIC_dt*175.0e-9; //each clock count is 175 ns
// if (ICP_CTL==1)TIC_dt=TIC_dt/float(nav_tic); //use basic TIC interval for ICP
if (out_debug) fprintf(debug,"n_track= %d\n",n_track);
for (i=1;i<=n_track;i++)
{
       track_sat[i]=satpos_ephemeris(tr_time[i],chan[tr_ch[i]].prn);
      // process Carrier Tracking Loop or Integrated Carrier Phase
       if(ICP_CTL==0)// satellite velocity
```

```
        {
           dm_gps_sat[i]=satpos_ephemeris(tr_time[i]-
TIC_dt/2.0,chan[tr_ch[i]].prn); //for CTL

dp_gps_sat[i]=satpos_ephemeris(tr_time[i]+TIC_dt/2.0,chan[tr_ch[i]].prn);
                 d_sat[i].x=(dp_gps_sat[i].x-dm_gps_sat[i].x)/TIC_dt-
track_sat[i].y*omegae;
                 d_sat[i].y=(dp_gps_sat[i].y-
dm_gps_sat[i].y)/TIC_dt+track_sat[i].x*omegae;
                 d_sat[i].z=(dp_gps_sat[i].z-dm_gps_sat[i].z)/TIC_dt;
           meas_dop[i]=(chan[tr_ch[i]].doppler-33010105L)*42.574746268e-3;
}
        else
        {
                 dm_gps_sat[i]=satpos_ephemeris(tr_time[i]-
TIC_dt/float(nav_tic),chan[tr_ch[i]].prn); //for ICP
                 dp_gps_sat[i]=track_sat[i];
                 d_sat[i].x=(dp_gps_sat[i].x-
dm_gps_sat[i].x)/TIC_dt*float(nav_tic)-track_sat[i].y*omegae;
                 d_sat[i].y=(dp_gps_sat[i].y-
dm_gps_sat[i].y)/TIC_dt*float(nav_tic)+track_sat[i].x*omegae;
           d_sat[i].z=(dp_gps_sat[i].z-dm_gps_sat[i].z)/TIC_dt*float(nav_tic);
           meas_dop[i]= chan[tr_ch[i]].int_carr_phase/TIC_dt*float(nav_tic)-
1.405396826e6;
        }
              t_cor[i]=track_sat[i].tb-

tropo_iono(tr_ch[i],track_sat[i].az,track_sat[i].el,tr_time[i]);
              dt[i]=m_time[1]-(tr_time[i]-t_cor[i]);
       }
       if (n_track>=4)
       {
        rpvt=pos_vel_time(n_track);
        cbias=rpvt.dt;
        clock_error=rpvt.df;
        m_time[1]=m_time[1]-cbias;
        rp_ecef.x=rpvt.x;
        rp_ecef.y=rpvt.y;
        rp_ecef.z=rpvt.z;
        rp_llh=ecef_to_llh(rp_ecef);   // last section to open up
        if (rp_llh.hae>-2000.0 && rp_llh.hae< 18000 ) // a quick reasonableness
check
        {
//
//  Translate velocity into North, East, Up coordinates
//
            velocity();
            if
(sqrt(pow(receiver.vel.north,2.0)+pow(receiver.vel.east,2.0)+pow(receiver.vel.up
,2.0))<514.0)
            {
                  if (fabs(clock_error)<5.0) clock_offset=clock_error;
                  if (almanac_valid==1) status=navigating;
                  if (align_t==1)
                  {
                    old_TIC_cntr=TIC_cntr;
                    delta_m_time= modf(m_time[1],&ipart);
```

```
                    if (nav_up<1.0)
                    {
                         delta_m_time=modf(delta_m_time/nav_up,&ipart);
                         if (delta_m_time>0.5) m_error=(delta_m_time-
1.0)*nav_up;
                         else m_error=delta_m_time*nav_up;
                    }
                    else
                    {
                         if (delta_m_time>0.5) m_error=(delta_m_time-
1.0)/nav_up;
                         else m_error=delta_m_time/nav_up;
                    }
                    TIC_cntr=(TIC_ref-m_error*TIC_ref/10)*(1.0-
clock_offset*1.0e-6);
                    program_TIC(TIC_cntr);
                  }
                  rec_pos_llh.lon=rp_llh.lon;
                  rec_pos_llh.lat=rp_llh.lat;
                  rec_pos_llh.hae=rp_llh.hae;
                  current_loc.lon=rp_llh.lon;
                  current_loc.lat=rp_llh.lat;
                  current_loc.hae=rp_llh.hae;
                  rec_pos_xyz.x=rp_ecef.x;
                  rec_pos_xyz.y=rp_ecef.y;
                  rec_pos_xyz.z=rp_ecef.z;
//
//  Calculate DOPS
//
                  dops(n_track);             // see if this is the problem
                  if (out_pos==1) fprintf(stream,"time = %20.10lf, lat=
%lf,long= %lf,hae= %lf ",

      m_time[1],rec_pos_llh.lat*r_to_d,rec_pos_llh.lon*r_to_d,rec_pos_llh.hae);
         if (out_vel==1) fprintf(stream,"vn= %lf, ve= %lf, vu= %lf ",
         receiver.vel.north,receiver.vel.east,receiver.vel.up);
         if (out_time==1) fprintf(stream," clock= %lf ",clock_offset);
         if (out_pos || out_vel || out_time) fprintf(stream,"hdop= %f,vdop=
%f,tdop= %f\n",hdop,vdop,tdop);
//
//    Since we have a valid position/velocity narrow the doppler search window
//    to +-5 doppler bins
//
                  search_max_f=5;
                  m_time[0]=m_time[1];
            }
        }
       }
       else
       {
                  m_time[1]=m_time[1]+TIC_dt*(1.0+clock_offset/1.e6); // less
than 4 sats
                  rp_ecef.x=0.0;
                  rp_ecef.y=0.0;
                  rp_ecef.z=0.0;
                  rpvt.xv=0.0;
                  rpvt.yv=0.0;
```

```
                 rpvt.zv=0.0;
       }
       if (out_kalman==1)   // Kalman filter output
       {
                 fprintf(kalm,
                 "time %20.10lf, rpx %15.10lf, rpy %15.10lf, rpz %15.10lf, ",
                   m_time[1],rp_ecef.x,rp_ecef.y,rp_ecef.z);
                 fprintf(kalm,"rvx %15.10lf, rvy %15.10lf, rvz %15.10lf, Nsats
%d\n",
                   rpvt.xv,rpvt.yv,rpvt.zv,n_track);
       }
       for (i=1;i<=n_track;i++)
       {
                 chan[tr_ch[i]].Pr=(m_time[1]-(tr_time[i]-t_cor[i]))*c;
                 chan[tr_ch[i]].dPr=meas_dop[i]*lambda;
                 if (out_kalman==1)   // Kalman filter output
                 {
                       fprintf(kalm,"  PRN %2d,  px %20.10lf,  py %20.10lf, pz
%20.10lf,  ",

      chan[tr_ch[i]].prn,track_sat[i].x,track_sat[i].y,track_sat[i].z);
                       fprintf(kalm," vx %16.10lf, vy %16.10lf, vz %16.10lf,
",
                       d_sat[i].x,d_sat[i].y,d_sat[i].z);
                       fprintf(kalm," Pr %20.10lf,  dPr %16.10lf\n",
                       chan[tr_ch[i]].Pr,chan[tr_ch[i]].dPr);
                 }
       }

}

/*******************************************************************************
FUNCTION velocity(void)
RETURNS  None.

PARAMETERS None.

PURPOSE  To convert velocity from ecef to local level (WGS-84) axes

WRITTEN BY
      Clifford Kelley

*******************************************************************************/
void velocity(void)
{
    receiver.north.x=-cos(rec_pos_llh.lon)*sin(rec_pos_llh.lat);
    receiver.north.y=-sin(rec_pos_llh.lon)*sin(rec_pos_llh.lat);
    receiver.north.z= cos(rec_pos_llh.lat);
    receiver.east.x=-sin(rec_pos_llh.lon);
    receiver.east.y= cos(rec_pos_llh.lon);
//  receiver.east.z=0.0;
    receiver.up.x=cos(rec_pos_llh.lon)*cos(rec_pos_llh.lat);
    receiver.up.y=sin(rec_pos_llh.lon)*cos(rec_pos_llh.lat);
    receiver.up.z=sin(rec_pos_llh.lat);

       receiver.vel.north = rpvt.xv*receiver.north.x + rpvt.yv*receiver.north.y
+
```

```
                        rpvt.zv*receiver.north.z;
       receiver.vel.east  = rpvt.xv*receiver.east.x + rpvt.yv*receiver.east.y;
       receiver.vel.up    = rpvt.xv*receiver.up.x + rpvt.yv*receiver.up.y +
                        rpvt.zv*receiver.up.z;


speed=sqrt(receiver.vel.north*receiver.vel.north+receiver.vel.east*receiver.vel.
east);
       if (speed==0.0) heading=0.0;
       else heading=atan2(receiver.vel.east,receiver.vel.north);

}
```

4.7.6 Computing Position

The method used in the software is LS or Least Squares method which minimizes the square of the difference from the measured range from each satellite and a position fix.

$$x = \left(H^T H\right)^{-1} H^T z$$

$$x = \left(H^T W^{-1} H\right)^{-1} H^T W^{-1} z$$

The position is solved iterativly starting with the center of the earth if no position is available or the last known solution. This can generally be accomplished in fewer than 4 iterations.

MORE STUFF NEEDED

4.7.7 Computing Velocity

Once the position is computed the velocity is computed using either the carrier trackin loop DCO setting (CTL) or the integrated carrier phase or ICP. Since the satellite-receiver geometry has been resolved the velocity can be solved directly.

MORE STUFF NEEDED

```
/*************************************************************************
FUNCTION pos_vel_time(int nsl)
RETURNS  None.

PARAMETERS
                  nsl   int

PURPOSE

      This routine processes the all-in-view pseudorange to arrive
      at a receiver position
```

```
INPUTS:
    pseudo_range[nsl] Vector of measured range from satellites to the receiver
       sat_location[nsl][3] Array of satellite locations in ECEF when the signal
                                          was sent
    nsl      number of satellites used

OUTPUTS:
    RP[3]    VECTOR OF RECEIVER POSITION IN ECEF (X,Y,Z)
    CBIAS    RECEIVER CLOCK BIAS FROM GPS TIME

VARIABLES USED:
    C        SPEED OF LIGHT IN VACUUM IN M/S
    S[6][5]  MATRIX USED FOR SOLVING FOR RECEIVER POSITION CORRECTION
    B[5]     RESULTING RECEIVER CLOCK BIAS & POSITION CORRECTIONS
    X,Y,Z    TEMPORARY RECEIVER POSITION
    T        TEMPORARY RECEIVER CLOCK BIAS
    R[5]     RANGE FROM RECEIVER TO SATELLITES

IF THE POSITION CANNOT BE DETERMINED THE RESULT OF RP
WILL BE (0,0,0) THE CENTER OF THE EARTH

WRITTEN BY
      Clifford Kelley

*************************************************************************/

pvt  pos_vel_time( int nsl)
{
  double dd[5][5],r,ms[5][13],pm[5][13],bm[13],br[5],correct_mag,x,y,z,t;
  double a1,b1,c1,d1,e1,f1,g1,h1,i1,j1,k1,l1,m1,n1,o1,p1,denom,alpha;
  int i,j,k,nits;
  pvt result;
//
//  USE ITERATIVE APPROACH TO SOLVING FOR THE POSITION OF
//  THE RECEIVER
//
  nits=0;
  t=0.0;
  x=rec_pos_xyz.x;
  y=rec_pos_xyz.y;
  z=rec_pos_xyz.z;
  do
  {
    for (i=1;i<=nsl;i++)
       {
//
//   Compute range in ECI at the time of arrival at the receiver
//
            alpha=(t-dt[i])*omegae;
            r=sqrt(pow(track_sat[i].x*cos(alpha)-track_sat[i].y*sin(alpha)-x,2)+

pow(track_sat[i].y*cos(alpha)+track_sat[i].x*sin(alpha)-y,2)+
                         pow(track_sat[i].z-z,2));
            bm[i]=r-(dt[i]-t)*c;
            ms[1][i]=(track_sat[i].x*cos(alpha)-track_sat[i].y*sin(alpha)-x)/r;
            ms[2][i]=(track_sat[i].y*cos(alpha)+track_sat[i].x*sin(alpha)-y)/r;
            ms[3][i]=(track_sat[i].z-z)/r;
```

```
            ms[4][i]=1.0;
        }
        a1=0.;b1=0.;c1=0.;d1=0.;
        e1=0.;f1=0.;g1=0.;h1=0.;
        i1=0.;j1=0.;k1=0.;l1=0.;
        m1=0.;n1=0.;o1=0.;p1=0.;
        for (k=1;k<=nsl;k++)
        {
              a1+=ms[1][k]*ms[1][k];
              b1+=ms[1][k]*ms[2][k];
              c1+=ms[1][k]*ms[3][k];
              d1+=ms[1][k]*ms[4][k];
//      e1+=ms[2][k]*ms[1][k];   for completeness, the matrix is symmetric
              f1+=ms[2][k]*ms[2][k];
              g1+=ms[2][k]*ms[3][k];
              h1+=ms[2][k]*ms[4][k];
//      i1+=ms[3][k]*ms[1][k];
//      j1+=ms[3][k]*ms[2][k];
              k1+=ms[3][k]*ms[3][k];
              l1+=ms[3][k]*ms[4][k];
//      m1+=ms[1][k];
//      n1+=ms[2][k];
//      o1+=ms[3][k];
              p1+=ms[4][k];
         }
         o1=l1;m1=d1;n1=h1;e1=b1;i1=c1;j1=g1;
/*
         SOLVE FOR THE MATRIX INVERSE
*/
             denom=(k1*p1-l1*o1)*(a1*f1-b1*e1) + (l1*n1-j1*p1)*(a1*g1-c1*e1) +
                   (j1*o1-k1*n1)*(a1*h1-d1*e1) + (l1*m1-i1*p1)*(c1*f1-b1*g1) +
                   (i1*o1-k1*m1)*(d1*f1-b1*h1) + (i1*n1-j1*m1)*(c1*h1-d1*g1);
             dd[1][1]=f1*(k1*p1-l1*o1)+g1*(l1*n1-j1*p1)+h1*(j1*o1-k1*n1);
             dd[1][2]=e1*(l1*o1-k1*p1)+g1*(i1*p1-l1*m1)+h1*(k1*m1-i1*o1);
             dd[1][3]=e1*(j1*p1-n1*l1)-i1*(f1*p1-n1*h1)+m1*(f1*l1-j1*h1);
             dd[1][4]=e1*(n1*k1-j1*o1)+i1*(f1*o1-n1*g1)+m1*(j1*g1-f1*k1);
//           dd[2][1]=b1*(l1*o1-k1*p1)+j1*(c1*p1-d1*o1)+n1*(d1*k1-c1*l1);
             dd[2][1]=dd[1][2];
             dd[2][2]=a1*(k1*p1-l1*o1)+c1*(l1*m1-i1*p1)+d1*(i1*o1-k1*m1);
             dd[2][3]=a1*(l1*n1-j1*p1)+i1*(b1*p1-n1*d1)+m1*(j1*d1-b1*l1);
             dd[2][4]=a1*(j1*o1-n1*k1)-i1*(b1*o1-n1*c1)+m1*(b1*k1-c1*j1);
//           dd[3][1]=b1*(g1*p1-h1*o1)-f1*(c1*p1-o1*d1)+n1*(c1*h1-d1*g1);
             dd[3][1]=dd[1][3];
//           dd[3][2]=a1*(o1*h1-g1*p1)+e1*(c1*p1-o1*d1)+m1*(d1*g1-c1*h1);
             dd[3][2]=dd[2][3];
             dd[3][3]=a1*(f1*p1-h1*n1)+b1*(h1*m1-e1*p1)+d1*(e1*n1-f1*m1);
             dd[3][4]=a1*(n1*g1-f1*o1)+e1*(b1*o1-c1*n1)+m1*(c1*f1-b1*g1);
//           dd[4][1]=b1*(h1*k1-g1*l1)+f1*(c1*l1-d1*k1)+j1*(d1*g1-c1*h1);
             dd[4][1]=dd[1][4];
//           dd[4][2]=a1*(g1*l1-h1*k1)-e1*(c1*l1-d1*k1)+i1*(c1*h1-d1*g1);
             dd[4][2]=dd[2][4];
//           dd[4][3]=a1*(j1*h1-f1*l1)+e1*(b1*l1-d1*j1)+i1*(d1*f1-b1*h1);
             dd[4][3]=dd[3][4];
             dd[4][4]=a1*(f1*k1-g1*j1)+b1*(g1*i1-e1*k1)+c1*(e1*j1-f1*i1);

             if ( denom<=0.0 )
             {
```

```
              result.x=1.0;       // something went wrong
              result.y=1.0;       // set solution to near center of earth
              result.z=1.0;
              result.dt=0.0;
            }
            else
            {
              for (i=1;i<=4;i++)
              {
                   for (j=1;j<=4;j++) dd[i][j]=dd[i][j]/denom;
              }
              for (i=1;i<=nsl;i++)
              {
                   for (j=1;j<=4;j++)
                   {
                        pm[j][i]=0.0;
                        for (k=1;k<=4;k++)pm[j][i]+=dd[j][k]*ms[k][i];
                   }
             }
             for (i=1;i<=4;i++)
             {
                  br[i]=0.0;
                  for (j=1;j<=nsl;j++)br[i]+=bm[j]*pm[i][j];
             }
             nits++;
             x=x+br[1];
             y=y+br[2];
             z=z+br[3];
             t=t-br[4]/c;
             correct_mag=sqrt(br[1]*br[1]+br[2]*br[2]+br[3]*br[3]);
             }
  } while ( correct_mag > 0.01 && correct_mag < 1.e8 && nits < 10);
  result.x=x;
  result.y=y;
  result.z=z;
  result.dt=t;
//
//  Now for Velocity
//

  for (i=1;i<=nsl;i++)
  {
       alpha=(dt[i]-t)*omegae;
       r=sqrt(pow(track_sat[i].x*cos(alpha)-track_sat[i].y*sin(alpha)-x,2)+
                    pow(track_sat[i].y*cos(alpha)+track_sat[i].x*sin(alpha)-
y,2)+
                    pow(track_sat[i].z-z,2));
       bm[i]=((track_sat[i].x*cos(alpha)-track_sat[i].y*sin(alpha)-
x)*d_sat[i].x+
                    (track_sat[i].y*cos(alpha)+track_sat[i].x*sin(alpha)-
y)*d_sat[i].y+
                    (track_sat[i].z-z)*d_sat[i].z)/r-meas_dop[i]*lambda;
  }
  for (i=1;i<=4;i++)
  {
        br[i]=0.0;
        for (j=1;j<=nsl;j++)br[i]+=bm[j]*pm[i][j];
```

```
  }
  result.xv=br[1]+y*omegae;     //  get rid of earth
  result.yv=br[2]-x*omegae;     //  rotation rate
  result.zv=br[3];
  result.df=br[4]/c*1000000.0; // frequency error in parts per million (ppm)
  return(result);
}
```

4.8 NMEA Serial Output

MORE STUFF NEEDED

4.9 Miscellaneous Algorithms

4.9.1 ECEF and Latitude/Longitude/Height Conversions

```
/******************************************************************************
FUNCTION ecef_to_llh(ecef pos)
RETURNS  position in llh structure

PARAMETERS
                  pos   ecef

PURPOSE    Convert a position in in cartesian ecef coordinates to
                    Geodetic WGS 84 coordinates

Based on equations found in Hoffman-Wellinhoff

WRITTEN BY
      Clifford Kelley

******************************************************************************/

llh ecef_to_llh(ecef pos)
{
  double p,n,thet,esq,epsq;
  llh result;
  p=sqrt(pos.x*pos.x+pos.y*pos.y);
  thet=atan(pos.z*a/(p*b));
  esq =1.0-b*b/(a*a);
  epsq=a*a/(b*b)-1.0;
  result.lat=atan((pos.z+epsq*b*pow(sin(thet),3))/(p-esq*a*pow(cos(thet),3)));
  result.lon=atan2(pos.y,pos.x);
  n=a*a/sqrt(a*a*cos(result.lat)*cos(result.lat) +
       b*b*sin(result.lat)*sin(result.lat));
  result.hae=p/cos(result.lat)-n;
  return(result);
}

/******************************************************************************
FUNCTION llh_to_ecef(llh pos)
RETURNS  position in ecef structure

PARAMETERS
                  pos   llh

PURPOSE    Convert a position in Geodetic WGS 84 coordinates to cartesian
                    ecef coordinates

Based on equations found in Hoffman-Wellinhoff

WRITTEN BY
      Clifford Kelley

******************************************************************************/

ecef llh_to_ecef(llh pos)
```

```
{
  double n;
  ecef result;
  n=a*a/sqrt(a*a*cos(pos.lat)*cos(pos.lat)+b*b*sin(pos.lat)*sin(pos.lat));
  result.x=(n+pos.hae)*cos(pos.lat)*cos(pos.lon);
  result.y=(n+pos.hae)*cos(pos.lat)*sin(pos.lon);
  result.z=(b*b/(a*a)*n+pos.hae)*sin(pos.lat);
  return(result);
}
```

Satfind

Satfind uses the receiver position, time, and almanac to predict satellites in view, their azimuth, elevation and Doppler shift of the signal

```
/*******************************************************************************
FUNCTION satfind()
RETURNS  None.

PARAMETERS None.

PURPOSE

      THIS FUNCTION DETERMINES THE SATELLITES TO SEARCH FOR
      WHEN ALMANAC DATA IS AVAILABLE

WRITTEN BY
      Clifford Kelley

*******************************************************************************/

satvis satfind(char i)
{
      float tdot,az;
   float satang,alm_time,almanac_date;
   double range1,range2,xls,yls,zls,xaz,yaz;
   long  jd_yr;
   ecef gpspos1,gpspos2,north,east,up;
   satvis result;
   int  jd_m;
   struct tm *gmt;
   double time_s;
/*
      INITIALIZE ALL THE CONSTANTS
*/
//    gotoxy(1,24);
//    printf("->satfind");
      putenv(tzstr);
      tzset();
//   thetime=time(NULL);
      gmt=gmtime(&thetime);
// set up the correct time
      if (gmt->tm_mon <= 1)
      {
```

```
        jd_yr =365.25*(gmt->tm_year-1.+1900.);
        jd_m  =30.6001*(gmt->tm_mon+14.);
      }
      else
      {
        jd_yr=365.25*(gmt->tm_year+1900.);
        jd_m =30.6001*(gmt->tm_mon+2.);
      }
      time_s=gmt->tm_min/1440.+gmt->tm_sec/86400.+1720981.5+gmt->tm_hour/24.
       +jd_yr+jd_m+gmt->tm_mday;
      gps_week=int((time_s-2444244.5)/7.);
      almanac_date=gps_alm[i].week*7.0+2444244.5;
   if (gps_week-gps_alm[i].week>512) almanac_date+=1024*7.0;
   alm_time=(time_s-almanac_date)*86400.;
   clock_tow=(time_s-gps_week*7.-2444244.5)*86400.;
/*
      CALCULATE THE POSITION OF THE SATELLITES
*/
   if (gps_alm[i].inc > 0.0 && i>0)
      {
      gpspos1=satpos_almanac(alm_time,i);
      gpspos2=satpos_almanac(alm_time+1.0,i);
/*
      CALCULATE THE POSITION OF THE RECEIVER
*/
      rec_pos_xyz=llh_to_ecef(current_loc);
      north.x=-cos(current_loc.lon)*sin(current_loc.lat);
      north.y=-sin(current_loc.lon)*sin(current_loc.lat);
   north.z= cos(current_loc.lat);
   east.x=-sin(current_loc.lon);
   east.y= cos(current_loc.lon);
//   east.z=0.0;
   up.x=cos(current_loc.lon)*cos(current_loc.lat);
   up.y=sin(current_loc.lon)*cos(current_loc.lat);
   up.z=sin(current_loc.lat);
/*
     DETERMINE IF A CLEAR LINE OF SIGHT EXISTS
*/
      xls =gpspos1.x-rec_pos_xyz.x;
      yls =gpspos1.y-rec_pos_xyz.y;
      zls =gpspos1.z-rec_pos_xyz.z;
      range1=sqrt(xls*xls+yls*yls+zls*zls);
      tdot=(up.x*xls+up.y*yls+up.z*zls)/range1;
      xls =xls/range1;
      yls =yls/range1;
      zls =zls/range1;
      range2=sqrt(pow(gpspos2.x-rec_pos_xyz.x-rpvt.xv,2)+
                           pow(gpspos2.y-rec_pos_xyz.y-rpvt.yv,2)+
                           pow(gpspos2.z-rec_pos_xyz.z-rpvt.zv,2));

      if ( tdot >= 1.00 ) satang=pi/2.0;
      else if ( tdot <= -1.00 ) satang=-pi/2.0;
      else  satang=asin(tdot);

      xaz=east.x*xls+east.y*yls;
      yaz=north.x*xls+north.y*yls+north.z*zls;
      if (xaz !=0.0 || yaz !=0.0) az=atan2(xaz,yaz);
```

```
        else az=0.0;
        result.x=gpspos1.x;
        result.y=gpspos1.y;
        result.z=gpspos1.z;
        result.elevation=satang;
        result.azimuth  =az;
        result.doppler  =(range1-range2)*5.2514;
    }
//      gotoxy(1,24);
//      printf("satfind->");
        return(result);
}
```

4.9.2 Dilution of Precision (DOP)

Early in the development of GPS it was clear that the two major effects on positioning error were the accuracy of measuring the range to the satellite and the geometry between the receiver and the satellitges. It was convenient to compute these two performance parameters separately. As long as the ranging error from one satellite to the next are uncorrelated and approximately the same this is a good approximation. Early receivers which had a limited number of channels used DOP as a means to decide which satellites to track.

$$\operatorname{cov}(dx) = E\left(dx, dx^T\right)$$

$$\operatorname{cov}(dx) = \left(H^T H\right)^{-1} \sigma_r$$

Where H is the matrix of line of sight vectors from the receiver to the satellite in the locally level earth axes.

$$H = \begin{bmatrix} R1_x & R1_y & R1_z & 1 \\ R2_x & R2_y & R2_z & 1 \\ R3_x & R3_y & R3_z & 1 \\ . & . & . & . \\ RN_x & RN_y & RN_z & 1 \end{bmatrix}$$

The result is a 4x4 matrix which is divided up into sections.

$$\begin{bmatrix} NDOP & . & . & . \\ . & EDOP & . & . \\ . & . & VDOP & . \\ . & . & . & TDOP \end{bmatrix}$$

EDOP = East Dilution of Precision
NDOP = North Dilution of Precision
VDOP = Vertical Dilution of Precision

These are typically combined in a number of ways to produce

$$HDOP = \sqrt{NDOP^2 + EDOP^2}$$ Horizontal Dilution of Precision

$$PDOP = \sqrt{NDOP^2 + EDOP^2 + VDOP^2}$$ Position Dilution of Precision

$$GDOP = \sqrt{NDOP^2 + EDOP^2 + VDOP^2 + TDOP^2}$$ Geometric Dilution of Precision

```
/*******************************************************************************
FUNCTION dops(int nsl)

RETURNS  None.

PARAMETERS
                  nsl  int

PURPOSE

      This routine computes the dops

INPUTS:
       sat_location[nsl][3] Array of satellite locations in ECEF when the signal
                                                 was sent
    nsl      number of satellites used
    receiver position

OUTPUTS:
    hdop = horizontal dilution of precision (rss of ndop & edop)
    vdop = vertical dilution of precision
    tdop = time dilution of precision
    pdop = position dilution of precision (rss of vdop & hdop)
    gdop = geometric dilution of precision (rss of pdop & tdop)

WRITTEN BY
      Clifford Kelley

*******************************************************************************/

void  dops( int nsl)
{
  double r,xls,yls,zls;
//  double det;
  int i;
  Matrix H(nsl,4),G(4,4);

  receiver.north.x=-cos(rec_pos_llh.lon)*sin(rec_pos_llh.lat);
  receiver.north.y=-sin(rec_pos_llh.lon)*sin(rec_pos_llh.lat);
  receiver.north.z= cos(rec_pos_llh.lat);
  receiver.east.x=-sin(rec_pos_llh.lon);
  receiver.east.y= cos(rec_pos_llh.lon);
//receiver.east.z=0.0;
  receiver.up.x=cos(rec_pos_llh.lon)*cos(rec_pos_llh.lat);
  receiver.up.y=sin(rec_pos_llh.lon)*cos(rec_pos_llh.lat);
```

```
  receiver.up.z=sin(rec_pos_llh.lat);
  for (i=1;i<=nsl;i++)
  {
//
//   Compute line of sight vectors
//
      xls=track_sat[i].x-rec_pos_xyz.x;
      yls=track_sat[i].y-rec_pos_xyz.y;
      zls=track_sat[i].z-rec_pos_xyz.z;

      r=sqrt(xls*xls+yls*yls+zls*zls);

      H(i,1)=(xls*receiver.north.x+yls*receiver.north.y+zls*receiver.north.z)/r;
      H(i,2)=(xls*receiver.east.x+yls*receiver.east.y)/r;
      H(i,3)=(xls*receiver.up.x+yls*receiver.up.y+zls*receiver.up.z)/r;
      H(i,4)=1.0;
  }

//  G=(H.transpose()*H).inverse();  //for Alberto's library
  G=(H.t()*H).i();                 // for Newmat library

  hdop=sqrt(G(1,1)+G(2,2));
  vdop=sqrt(G(3,3));
  tdop=sqrt(G(4,4));
  pdop=sqrt(G(1,1)+G(2,2)+G(3,3));
  gdop=sqrt(G(1,1)+G(2,2)+G(3,3)+G(4,4));
//  gotoxy(1,24);
//  printf("->dops");
}
```

4.9.3 Self Test

Before the program runs a quick self test is conducted to be confident that the digital interface from the computer to the GP2021 is working properly. The method used is to send two bit patterns to register 0xF2 (data retention register) and the data bus test register and read them back to confirm the data bus is working.

```
void self_test(void)
{
  unsigned int indataaax,indata55x,indataaay,indata55y,error;
  error=0;
  data_retent_w(0x5500);
  data_bus_test_w(0xaa55);
  indata55x=data_retent_r();
  indataaax=data_bus_test_r();
  data_retent_w(0xaa00);
  data_bus_test_w(0x55aa);
  indataaay=data_retent_r();
  indata55y=data_bus_test_r();
  if ((indata55x != 0x5500) || (indataaax != 0xaa55)
      || (indataaay != 0xaa00) || (indata55y != 0x55aa))
  {
       error=1;
```

```
       printf("data line error\n");
       printf("indata55x=%x , indataaax=%x\n",indata55x,indataaax);
       printf("indataaay=%x , indata55y=%x\n",indataaay,indata55y);
  }
  if (error==1) exit(0);
}
```

4.9.4 Navigation Message Decoding

In order to determine when a new navigation message is available every time a new frame of data is detected it is decoded. If the IODC and IODE are already available the message is discarded. If a new IODC or IODE is detected the message is decoded.
The decoding of the navigation message is directly taken from the ICD-GPS-200. Since the parity scheme used by GPS cannot do error correction or detect more than one bit in error a number of integrity checks are made to assure that the navigation message is reasonable.

MORE STUFF NEEDED

```
/*******************************************************************************
FUNCTION navmess()
RETURNS  None.

PARAMETERS None.

PURPOSE
      This function assembles and decodes the 1500 bit nav message
      into almanac and ephemeris messages

WRITTEN BY
      Clifford Kelley
3-2-2002  Made corrections suggested by Georg Beyerle GFZ
*******************************************************************************/

void navmess(char prn,char ch)
{
  int i,j,k;
  unsigned long isqra,ie,iomega0;
  long iaf0,iomegadot;
  char itgd,iaf2;
//  int icapl2;
  int iweek,iura,ihealth,iodc,iaf1;
  unsigned int itoe,itoc;
//  int fif;
  int iode,icrs,idn,icuc,icus,icic,iomegad;
  int icis,icrc,idoe,idot;
  unsigned int iae,iatoa;
  static i4page,i5page;
  int i4data,i5data,isv,iaomegad;
  long iaaf0,iaaf1,iadeli,iaomega0,im0,inc0,iw;
  unsigned long iasqr;
  long iaw,iam0,scale,ia0,ia1;
```

```
  char ial0,ial1,ial2,ial3,ibt0,ibt1,ibt2,ibt3;
  int  itot,iWNt,idtls,iWNlsf,iDN,idtlsf;//WNa
  int sfr,word,i4p,i5p;
  double r_sqra,r_inc0,r_ety;
  float d_toe;
//
//    assemble the 1500 data bits into subframes and words
//
//    gotoxy(1,24);
//    printf("->navmess prn %d  ch %d",prn,ch);
    d_toe=clock_tow-gps_eph[prn].toe;
    if (d_toe>302400.0) d_toe=d_toe-604800.0;
    else if (d_toe<-302400.0)d_toe=d_toe+604800.0;
       j=0;
       for (sfr=1;sfr<=5;sfr++)
       {
         for (word=1;word<=10;word++)
         {
              scale=536870912L;
              sf[sfr][word]=0;
              for (i=0;i<=29;i++)
              {
                    if (chan[ch].message[(j+chan[ch].offset)%1500]==1)
                    sf[sfr][word]+=scale;
                    scale=scale>>1;
                    j++;
              }
         }
       }
  parity_check(); // check the parity of the 1500 bit message
//
//   EPHEMERIS DECODE subframes 1, 2, 3
//
//  subframe 1
//
//   check parity of first 3 subframes, since it is a circular register
//   we may have over written the first few bits so allow for word 1 of
//   subframe 1 to have a problem
//

  if ((p_error[1]==0 || p_error[1]==0x200) && p_error[2]==0 && p_error[3]==0)
  {
            iodc=int(((sf[1][3] & 0x3) <<8 ) | ((sf[1][8] & 0xFF0000L) >>16));
            iode=int(sf[2][3]  >> 16);
            idoe=int(sf[3][10] >> 16);
//      fprintf(kalm,"prn=%d iodc=%d, iode=%d,idoe=%d\n",prn,iodc,iode,idoe);
//      fprintf(kalm," eph.iode=%d,
eph.iodc=%d\n",gps_eph[prn].iode,gps_eph[prn].iodc);
// if both copies of iode agree and we have a new iode or new iodc then process
the ephemeris
            if (iode==idoe && ((iode!=gps_eph[prn].iode) ||
(iodc!=gps_eph[prn].iodc)))
            {

                  iweek= int(sf[1][3] >> 14);
//                icapl2=( sf[1][3] & 0x3000 ) >> 12;
                  iura=int(( sf[1][3] & 0xF00 ) >> 8);
```

```
                  ihealth=int(( sf[1][3] & 0xFC ) >> 2);
                  itgd=int(sf[1][7] & 0xFF);
                  itoc=int(sf[1][8] & 0xFFFF);
                  iaf2=int(sf[1][9] >> 16);
                  iaf1=int(sf[1][9] & 0xFFFF);
                  iaf0=sf[1][10] >> 2;
                  if (bit_test_l(iaf0,22)) iaf0=iaf0 | 0xFFC00000L;
//
//   subframe 2
//
                  icrs=int(sf[2][3] & 0xFFFF);
                  idn=int(sf[2][4] >> 8);
                  im0=((sf[2][4] & 0xFF) << 24) | sf[2][5];
                  icuc=int(sf[2][6] >>8);
                  ie=((sf[2][6] & 0xFF) << 24) | sf[2][7];
                  icus=int(sf[2][8] >> 8);
                  isqra=(((sf[2][8] & 0xFF) << 24) | sf[2][9]);
                  itoe=int(sf[2][10] >> 8);
//                fif=int((sf[2][10] & 0x80) >> 7);
//
// subframe 3
//
                  icic=int(sf[3][3] >> 8);
                  icis=int(sf[3][5] >> 8);
                  inc0=((sf[3][5] & 0xFF) << 24) | sf[3][6];
                  iomega0=((sf[3][3] & 0xFF) << 24) | sf[3][4];
                  icrc=int(sf[3][7] >> 8);
                  iw=((sf[3][7] & 0xFF) << 24) | sf[3][8];
                  iomegadot=sf[3][9];
                  if (bit_test_l(iomegadot,24)) iomegadot=iomegadot |
0xFF000000L;
                  idot=int((sf[3][10] & 0xFFFC) >> 2);
                  if (bit_test_l(idot,14)) idot=idot | 0xC000;

                  r_sqra=isqra*c_2m19;
                  r_inc0=inc0*c_2m31*pi;
                  r_ety=ie*c_2m33;
//          fprintf(kalm,"sqra=%lf, inc=%lf,ety=%lf\n",r_sqra,r_inc0,r_ety);
//
// Does this ephemeris make sense?
//
                  if ((r_inc0<1.05 && r_inc0>0.873) && (r_sqra>5100.0 &&
r_sqra<5200.0) &&
                        (r_ety <.05 && r_ety>0.0))
                  {
                        gps_eph[prn].valid=1;

                        gps_eph[prn].iode=iode;
                        gps_eph[prn].iodc=iodc;
                        gps_eph[prn].week=iweek;
                        gps_eph[prn].ura=iura;
                        gps_eph[prn].health=ihealth;
                        gps_eph[prn].tgd=itgd*c_2m31;
                        gps_eph[prn].toc=itoc*16.0;
                        gps_eph[prn].af2=iaf2*c_2m55;
                        gps_eph[prn].af1=iaf1*c_2m43;
                        gps_eph[prn].af0=iaf0*c_2m31;
```

```
                        gps_eph[prn].crs=icrs*c_2m5;
                        gps_eph[prn].dn=idn*c_2m43*pi;
                        gps_eph[prn].ma=im0*c_2m31*pi;
                        gps_eph[prn].cuc=icuc*c_2m29;
                        gps_eph[prn].ety=r_ety;
                        gps_eph[prn].cus=icus*c_2m29;
                        gps_eph[prn].sqra=r_sqra;
                        gps_eph[prn].wm=19964981.84/pow(r_sqra,3);
                        gps_eph[prn].toe=itoe*c_2p4;
                        gps_eph[prn].cic=icic*c_2m29;
                        gps_eph[prn].cis=icis*c_2m29;
                        gps_eph[prn].inc0=r_inc0;
                        gps_eph[prn].w0=iomega0*c_2m31*pi;
                        gps_eph[prn].crc=icrc*c_2m5;
                        gps_eph[prn].w=iw*c_2m31*pi;
                        gps_eph[prn].omegadot=iomegadot*c_2m43*pi;
                        gps_eph[prn].idot=idot*c_2m43*pi;

                        if (out_debug)  write_Debug_ephemeris(prn);
                  }
                else if (gps_eph[prn].valid==1 && d_toe>7200.0)
gps_eph[prn].valid=0;

            }
  }
//
//    ALMANAC DECODE  subframes 4 and 5
//
//    SUBFRAME 4
//
// check parity of subframes 4 and five and don't bother if we have the almanac
//
  if (p_error[4]==0 && p_error[5]==0 && almanac_valid==0 && almanac_flag==0)
  {
             almanac_flag=1;
             i4data= int(sf[4][3] >> 22);
             i4p= int((sf[4][3] & 0x3F0000L) >> 16);
             if (i4p != i4page && i4data==1)  // i4p all we need is a page to be
             {                                // read from 1 satellite
             i4page=i4p;
             if (i4page > 24 && i4page < 33)
             {
                  isv=i4page ;
                  gps_alm[isv].week=gps_week%1024;
                  iae=int(sf[4][3] & 0x00FFFFL);
                  gps_alm[isv].ety=iae*c_2m21;
                  iatoa=int(sf[4][4] >> 16);
                  gps_alm[isv].toa=iatoa*c_2p12;
                  iadeli=sf[4][4] & 0x00FFFFL;
                  if (bit_test_l(iadeli,16)) iadeli=iadeli | 0xFFFF0000L;
                  gps_alm[isv].inc=(iadeli*c_2m19+0.3)*pi;
                  iomegad=int(sf[4][5] >> 8);
                  gps_alm[isv].rra=iomegad*c_2m38*pi;
                  gps_alm[isv].health=int(sf[4][5] & 0x0000FF);
                  iasqr=sf[4][6];
                  gps_alm[isv].sqa=iasqr*c_2m11;
```

```
            if (gps_alm[isv].sqa>0.0) gps_alm[isv].w=19964981.84/
                                                pow(gps_alm[isv].sqa,3);
            iaomega0=sf[4][7];
            if (bit_test_l(iaomega0,24)) iaomega0=iaomega0 | 0xFF000000L;
            gps_alm[isv].lan=iaomega0*c_2m23*pi;
            iaw=sf[4][8];
            if (bit_test_l(iaw,24)) iaw=iaw | 0xFF000000L;
            gps_alm[isv].aop=iaw*c_2m23*pi;
            iam0=sf[4][9];
            if (bit_test_l(iam0,24)) iam0=iam0 | 0xFF000000L;
            gps_alm[isv].ma=iam0*c_2m23*pi;
            iaaf0=(sf[4][10] >> 13) | ((sf[4][10] & 0x1C)>>2);
            if (bit_test_l(iaaf0,11)) iaaf0=iaaf0 | 0xFFFFF800L;
            gps_alm[isv].af0=iaaf0*c_2m20;
            iaaf1=(sf[4][10] | 0xFFE0) >> 5;
            if (bit_test_l(iaaf1,11)) iaaf1=iaaf1 | 0xFFFFF800L;
            gps_alm[isv].af1=iaaf1*c_2m38;
      }
      else if ( i4page == 55 )
      {
            gps_alm[prn].text_message[0]=char((sf[4][3] & 0x00FF00) >> 8);
            gps_alm[prn].text_message[1]=char( sf[4][3] & 0x0000FF);
            for ( k=1;k<=7;k++)
            {
              gps_alm[prn].text_message[3*k-1]= char(sf[4][k+3] >> 16);
              gps_alm[prn].text_message[3*k  ]= char((sf[4][k+3] &
0x00FF00) >> 8);
              gps_alm[prn].text_message[3*k+1]= char(sf[4][k+3] &
0x0000FF);
            }
      }
      else if ( i4page == 56 ) // Broadcast Ionosphere Model & UTC
Parameters
      {
            ial0=int((sf[4][3] & 0x00FF00) >> 8);
            al0=ial0*c_2m30;
            ial1= int(sf[4][3] & 0x0000FF);
            al1=ial1*c_2m27;
            ial2= int(sf[4][4] >> 16);
            al2=ial2*c_2m24;
            ial3=int((sf[4][4] & 0x00FF00) >> 8);
            al3=ial3*c_2m24;
            ibt0= int(sf[4][4] & 0x0000FF);
            b0=ibt0*2048.;
            ibt1= int(sf[4][5] >> 16);
            b1=ibt1*16384.;
            ibt2=int((sf[4][5] & 0x00FF00) >> 8);
            b2=ibt2*65536.;
            ibt3= int(sf[4][5] & 0x00FF);
            b3=ibt3*65536.;
            ia1=   sf[4][6];
            if (bit_test_l(ia1,24)) ia1=ia1 | 0xFF000000L;
            a1=ia1*c_2m50;
            ia0=  (sf[4][7] << 8) | (sf[4][8] >> 16);
            a0=ia0*c_2m30;
            itot=  int((sf[4][8] & 0x00FF00) >> 8);
            tot=itot*4096;
```

```
        iWNt=  int(sf[4][8] & 0xFF);
        WNt=iWNt;
        idtls=  int(sf[4][10] >> 16);
        if (idtls >128) idtls=idtls |0xFF00;
        dtls=idtls;
        iWNlsf=int((sf[4][9] & 0x00FF00) >> 8);
        WNlsf=iWNlsf;
        iDN   = int(sf[4][9] & 0x0000FF);
        DN=iDN;
        idtlsf= int(sf[4][9] >> 16);
        if (idtlsf >128) idtlsf=idtlsf |0xFF00;
        dtlsf=idtlsf;
  }
  else if ( i4page == 63 )
  {
        ASV[1]= int((sf[4][3] & 0x00F000) >>12);
        ASV[2]= int((sf[4][3] & 0x000F00) >>8);
        ASV[3]= int((sf[4][3] & 0x0000F0) >>4);
        ASV[4]= int( sf[4][3] & 0x00000F);
        ASV[5]= int( sf[4][4] >>20);
        ASV[6]= int((sf[4][4] & 0x0F0000L) >>16);
        ASV[7]= int((sf[4][4] & 0x00F000) >>12);
        ASV[8]= int((sf[4][4] & 0x000F00) >> 8);
        ASV[9]= int((sf[4][4] & 0x0000F0) >> 4);
        ASV[10]=int(sf[4][4] & 0x00000F);
        ASV[11]=int(sf[4][5] >>20);
        ASV[12]=int((sf[4][5] & 0x0F0000L) >>16);
        ASV[13]=int((sf[4][5] & 0x00F000) >>12);
        ASV[14]=int((sf[4][5] & 0x000F00) >> 8);
        ASV[15]=int((sf[4][5] & 0x0000F0) >> 4);
        ASV[16]=int(sf[4][5] & 0x00000F);
        ASV[17]=int(sf[4][6] >>20);
        ASV[18]=int((sf[4][6] & 0x0F0000L) >>16);
        ASV[19]=int((sf[4][6] & 0x00F000) >>12);
        ASV[20]=int((sf[4][6] & 0x000F00) >> 8);
        ASV[21]=int((sf[4][6] & 0x0000F0) >> 4);
        ASV[22]=int(sf[4][6] & 0x00000F);
        ASV[23]=int(sf[4][7] >>20);
        ASV[24]=int((sf[4][7] & 0x0F0000L) >>16);
        ASV[25]=int((sf[4][7] & 0x00F000) >>12);
        ASV[26]=int((sf[4][7] & 0x000F00) >> 8);
        ASV[27]=int((sf[4][7] & 0x0000F0) >> 4);
        ASV[28]=int( sf[4][7] & 0x00000F);
        ASV[29]=int( sf[4][8] >>20);
        ASV[30]=int((sf[4][8] & 0x0F0000L) >>16);
        ASV[31]=int((sf[4][8] & 0x00F000) >>12);
        ASV[32]=int((sf[4][8] & 0x000F00) >> 8);
        SVh[25]=int(sf[4][8] & 0x00003F);
        if( SVh[25]==0x3f) gps_alm[25].inc=0.0;
        SVh[26]=int(sf[4][9] >>18);
        if( SVh[26]==0x3f) gps_alm[26].inc=0.0;
        SVh[27]=int((sf[4][9] & 0x03F000L) >>12);
        if( SVh[27]==0x3f) gps_alm[27].inc=0.0;
        SVh[28]=int((sf[4][9] & 0x000FC0) >>6);
        if( SVh[28]==0x3f) gps_alm[28].inc=0.0;
        SVh[29]= int(sf[4][9] & 0x00003F);
        if( SVh[29]==0x3f) gps_alm[29].inc=0.0;
```

```
                SVh[30]= int(sf[4][10] >>18);
                if( SVh[30]==0x3f) gps_alm[30].inc=0.0;
                SVh[31]=int((sf[4][10]& 0x03F000L) >>12);
                if( SVh[31]==0x3f) gps_alm[31].inc=0.0;
                SVh[32]=int((sf[4][10]& 0x000FC0) >>6);
                if( SVh[32]==0x3f) gps_alm[32].inc=0.0;
        }
        }
            i5data=int(sf[5][3] >> 22);
            i5p=int((sf[5][3] & 0x3F0000L) >> 16);
            chan[ch].page5=i5p;
            if (i5page != i5p && i5data==1)
            {
                i5page=i5p;
                if ( i5page == 51 )
                {
                    iatoa=int((sf[5][3] & 0xFF00) >>8);
//          atoa=iatoa*4096;
                    SVh[1]=int(sf[5][4] >>18);
                    if( SVh[1]==0x3f) gps_alm[1].inc=0.0;
                    SVh[2]=int((sf[5][4] & 0x03F000L)>>12);
                    if( SVh[2]==0x3f) gps_alm[2].inc=0.0;
                    SVh[3]=int((sf[5][4] & 0x000FC0)>>6);
                    if( SVh[3]==0x3f) gps_alm[3].inc=0.0;
                    SVh[4]= int(sf[5][4] & 0x00003F);
                    if( SVh[4]==0x3f) gps_alm[4].inc=0.0;
                    SVh[5]= int(sf[5][5] >>18);
                    if( SVh[5]==0x3f) gps_alm[5].inc=0.0;
                    SVh[6]=int((sf[5][5] & 0x03F000L)>>12);
                    if( SVh[6]==0x3f) gps_alm[6].inc=0.0;
                    SVh[7]=int((sf[5][5] & 0x000FC0)>>6);
                    if( SVh[7]==0x3f) gps_alm[7].inc=0.0;
                    SVh[8]= int(sf[5][5] & 0x00003F);
                    if( SVh[8]==0x3f) gps_alm[8].inc=0.0;
                    SVh[9]= int(sf[5][6] >>18);
                    if( SVh[9]==0x3f) gps_alm[9].inc=0.0;
                    SVh[10]=int((sf[5][6] & 0x03F000L)>>12);
                    if( SVh[10]==0x3f) gps_alm[10].inc=0.0;
                    SVh[11]=int((sf[5][6] & 0x000FC0)>>6);
                    if( SVh[11]==0x3f) gps_alm[11].inc=0.0;
                    SVh[12]= int(sf[5][6] & 0x00003F);
                    if( SVh[12]==0x3f) gps_alm[12].inc=0.0;
                    SVh[13]= int(sf[5][7] >>18);
                    if( SVh[13]==0x3f) gps_alm[13].inc=0.0;
                    SVh[14]=int((sf[5][7] & 0x03F000L)>>12);
                    if( SVh[14]==0x3f) gps_alm[14].inc=0.0;
                    SVh[15]=int((sf[5][7] & 0x000FC0)>>6);
                    if( SVh[15]==0x3f) gps_alm[15].inc=0.0;
                    SVh[16]= int(sf[5][7] & 0x00003F);
                    if( SVh[16]==0x3f) gps_alm[16].inc=0.0;
                    SVh[17]= int(sf[5][8] >>18);
                    if( SVh[17]==0x3f) gps_alm[17].inc=0.0;
                    SVh[18]=int((sf[5][8] & 0x03F000L)>>12);
                    if( SVh[18]==0x3f) gps_alm[18].inc=0.0;
                    SVh[19]=int((sf[5][8] & 0x000FC0)>>6);
                    if( SVh[19]==0x3f) gps_alm[19].inc=0.0;
                    SVh[20]= int(sf[5][8] & 0x00003F);
```

```
                        if( SVh[20]==0x3f) gps_alm[20].inc=0.0;
                        SVh[21]= int(sf[5][9] >>18);
                        if( SVh[21]==0x3f) gps_alm[21].inc=0.0;
                        SVh[22]=int((sf[5][9] & 0x03F000L)>>12);
                        if( SVh[22]==0x3f) gps_alm[22].inc=0.0;
                        SVh[23]=int((sf[5][9] & 0x000FC0)>>6);
                        if( SVh[23]==0x3f) gps_alm[23].inc=0.0;
                        SVh[24]= int(sf[5][9] & 0x00003F);
                        if( SVh[24]==0x3f) gps_alm[24].inc=0.0;
            }
            else
            {
            isv=i5page;
            gps_alm[isv].week=gps_week%1024;
            iae=int(sf[5][3] & 0xFFFF);
            gps_alm[isv].ety=iae*c_2m21;
            iatoa=int(sf[5][4] >> 16);
            gps_alm[isv].toa=iatoa*4096.0;
            iadeli=int(sf[5][4] & 0xFFFF);
            gps_alm[isv].inc=(iadeli*c_2m19+0.3)*pi;
            iaomegad=int(sf[5][5] >> 8);
            gps_alm[isv].rra=iaomegad*c_2m38*pi;
            gps_alm[isv].health=int(sf[5][5] & 0xFF);
            iasqr=sf[5][6];
            gps_alm[isv].sqa=iasqr*c_2m11;
            if (gps_alm[isv].sqa>0.0)
gps_alm[isv].w=19964981.84/pow(gps_alm[isv].sqa,3);
            iaomega0=sf[5][7];
            if (bit_test_l(iaomega0,24)) iaomega0=iaomega0 | 0xFF000000L;
            gps_alm[isv].lan=iaomega0*c_2m23*pi;
            iaw=sf[5][8];
            if (bit_test_l(iaw,24)) iaw=iaw | 0xFF000000L;
            gps_alm[isv].aop=iaw*c_2m23*pi;
            iam0=sf[5][9];
            if (bit_test_l(iam0,24)) iam0=iam0 | 0xFF000000L;
            gps_alm[isv].ma=iam0*c_2m23*pi;
            iaaf0=int((sf[5][10] >> 13) | ((sf[5][10] & 0x1C)>>2));
            if (bit_test_l(iaaf0,11)) iaaf0=iaaf0 | 0xF800;
            gps_alm[isv].af0=iaaf0*c_2m20;
            iaaf1=int((sf[5][10] & 0xFFE0) >> 5);
            if (bit_test_l(iaaf1,11)) iaaf1=iaaf1 | 0xF800;
            gps_alm[isv].af1=iaaf1*c_2m38;
        }
      }
       }

}
```

Parity Checking

```
/*******************************************************************************
FUNCTION parity_check(void)
RETURNS  None.

PARAMETERS None.
```

```
PURPOSE  checks the parity of the 5 subframes of the nav message

WRITTEN BY
      Clifford Kelley

*******************************************************************************/

void  parity_check(void)
{
  long pb1=0x3b1f3480L,pb2=0x1d8f9a40L,pb3=0x2ec7cd00L;
  long pb4=0x1763e680L,pb5=0x2bb1f340L,pb6=0x0b7a89c0L;
  int  parity,m_parity;
  char d29=0,d30=0,sfm,word,b_1,b_2,b_3,b_4,b_5,b_6;
  int err_bit;
  for (sfm=1;sfm<=5;sfm++)
  {
        p_error[sfm]=0;
        for (word=1;word<=10;word++)
        {
             m_parity=int(sf[sfm][word] &0x3f);
       b_1=exor(d29,sf[sfm][word] & pb1) << 5;
       b_2=exor(d30,sf[sfm][word] & pb2) << 4;
       b_3=exor(d29,sf[sfm][word] & pb3) << 3;
       b_4=exor(d30,sf[sfm][word] & pb4) << 2;
       b_5=exor(0,sf[sfm][word] & pb5) << 1;
       b_6=exor(d29^d30,sf[sfm][word] & pb6);
       parity=b_1+b_2+b_3+b_4+b_5+b_6;
       err_bit=0;
       if (parity != m_parity)
       {
                   err_bit=1;
       }
       p_error[sfm]=(p_error[sfm] << 1) + err_bit;
       if (d30==1) sf[sfm][word]=0x03fffffc0L & ~sf[sfm][word];
       sf[sfm][word]=sf[sfm][word]>>6;
       d29=(m_parity & 0x2) >>1;
       d30=m_parity & 0x1;
     }
   }
}

/*******************************************************************************
FUNCTION exor(char bit, long parity)
RETURNS  None.

PARAMETERS
                  bit     char
                  parity  long

PURPOSE
                  count the number of bits set in the parameter parity and
                  do an exclusive or with the parameter bit

WRITTEN BY
      Clifford Kelley

*******************************************************************************/
```

```
int exor(char bit, long parity)
{
  char i;
  int result;
  result=0;
  for (i=7;i<=30;i++)
  {
    if (bit_test_l(parity,i)) result++;
  }
  result=result%2;
  result=(bit ^ result) & 0x1;
  return(result);
}
```

4.9.5 Interrupt Install/Remove

Since the need to talk to the GP2021 to obtain correlation data is only a function of time and not the result of any particular event the timing function for generating this interrupt was set up to use the IBM PCs own real time clock which uses interrupt IRQ0. Since the clock is only updated every second this function is taken over by OpenSourceGPS. The interrupt install function removes the link from IRQ0 from the real time clock and re-directs it to OSGPS. The interrupt remove function reverses this process and restores the real time clock.

```
/*******************************************************************************
FUNCTION Interrupt_Install()
RETURNS  None.

PARAMETERS None.

PURPOSE
      This function replaces the current IRQ0 Interrupt service routine with
      our own custom function. The old vector is stored in a global variable
      and will be reinstalled at the end of program execution. IRQ0 is
      enabled by altering the interrupt mask stored by the 8259 interrupt
      handler.

*******************************************************************************/
#ifdef BCPP
void Interrupt_Install()
{
  unsigned char     int_mask,i_high,i_low;
  i_high=interr_int>>8;
  i_low=interr_int&0xff;
  Old_Interrupt = getvect(8 + IRQLEVEL);
  disable();
  setvect(8 + IRQLEVEL,GPS_Interrupt);
  int_mask = inportb(0x21);     // get hardware interrupt mask
  int_mask = int_mask & ~(1 << IRQLEVEL);
  outportb(0x21,int_mask);      // send new mask to 8259
  enable();
// modify the timer to divide by interr_int
  outportb(0x43,0x34);
  outportb(0x40,i_low);
  outportb(0x40,i_high);
  outportb(0x20,0x20); // Clear PIC
```

```
}
#endif
#ifdef VCPP  // MS
void Interrupt_Install()
{
      unsigned char     int_mask, i_high, i_low;
      i_high = interr_int >> 8;
      i_low  = interr_int & 0xff;
      Old_Interrupt = _dos_getvect( 8 + IRQLEVEL);
      _disable();
      _dos_setvect( 8 + IRQLEVEL, GPS_Interrupt);
      int_mask = _inp(0x21);     // Get hardware interrupt mask

      int_mask = int_mask & ~(1 << IRQLEVEL);

      _outp( 0x21, int_mask );

      _enable();

      // Modify the timer to divide by interr_int
      _outp(0x43,0x34 );

      _outp( 0x40,i_low );

      _outp( 0x40,i_high );

      _outp( 0x20,0x20 ); // Clear the PIC

}

#endif


/*******************************************************************************
FUNCTION Interrupt_Remove()
RETURNS  None.

PARAMETERS None.

PURPOSE
      This function removes the custom interrupt vector from the vector
      table and restores the previous vector.

*******************************************************************************/
#ifdef BCPP

void Interrupt_Remove()
{
  unsigned char     int_mask;

  outportb(0x20,0x20);           // clear interrupt and allow next one
  int_mask = inportb(0x21);      // get hardware interrupt mask
  int_mask = int_mask | (1 << IRQLEVEL);
  disable();
//outportb(0x21,int_mask);       // send new mask to 8259
  setvect(8 + IRQLEVEL,Old_Interrupt);
```

```
  enable();                        // allow hardware interrupts
  outportb(0x20,0x20);             // clear interrupt and allow next one
  outportb(0x43,0x34);             // reset clock
  outportb(0x40,0xff);
  outportb(0x40,0xff);
}
#endif
#ifdef VCPP
void Interrupt_Remove()
{
      unsigned char     int_mask;

      _outp( 0x20,0x20);
      // PGB outportb(0x20,0x20);      // clear interrupt and allow next one
      int_mask = inp(0x21);
      // PGB int_mask = inportb(0x21);      // get hardware interrupt mask
      int_mask = int_mask | (1 << IRQLEVEL);
      _disable();
      //outportb(0x21,int_mask);       // send new mask to 8259
      _dos_setvect(8 + IRQLEVEL,Old_Interrupt);
      _enable();                       // allow hardware interrupts
      _outp(0x20,0x20);            // clear interrupt and allow next one
      _outp(0x43,0x34);            // reset clock
      _outp(0x40,0xff);
      _outp(0x40,0xff);
}

#endif
```

4.9.6 GP2021 Register Commands

In some cases when speed is required the GP2021 must be accessed directly using the to_gps and from_gps functions. These are the basic building blocks for all of the functions that follow.

```
//inline void to_gps(int add,int data)
void to_gps(int add,int data)
{
 outpw(0x304,add);
 outpw(0x308,data);
}

//inline int from_gps(int add)
int from_gps(int add)
{
  outpw(0x304,add);
  return(inpw(0x308));
}

inline int accum_status(void)
{
  return(from_gps(0x82));
}

void all_accum_reset(void)
```

```
{
}

void data_tst(int data)
{
  to_gps(0xf2,data);
}

unsigned int  ch_epoch(char ch)
{
  return(from_gps((ch<<3)+4));
}

unsigned int  ch_epoch_chk(char ch)
{
  return(from_gps((ch<<3)+7));
}

long   ch_carrier_cycle(char ch)
{
  long result;
  result=from_gps((ch<<3)+6);
  result=result<<16;
  result=result+from_gps((ch<<3)+2);
  return(result);
}

int  ch_code_DCO_phase(char ch)
{
  return(from_gps((ch<<3)+5));
}

void ch_code_incr_hi(char ch,int data)
{
  to_gps((ch<<3)+0x5,data);
}

void ch_code_incr_lo(char ch,int data)
{
  to_gps((ch<<3)+0x6,data);
}

int ch_code_phase(char ch)
{
 return(from_gps((ch<<3)+0x1));
}

int ch_carrier_DCO_phase(char ch)
{
  return(from_gps((ch<<3)+0x3));
}

void carr_incr_hi(char ch,int data)
{
  to_gps((ch<<3)+0x3,data);
}
```

```
void carr_incr_lo(char ch,int data)
{
  to_gps((ch<<3)+0x4,data);
}

void ch_cntl(char ch,int data)
{
//  printf("ch=%d  port=%x\n",ch,port(ch<<3));
  to_gps(ch<<3,data);
}


void all_cntl(int data)
{
  to_gps(0x70,data);

}
void multi_cntl(int data)
{
  to_gps(0x60,data);

}

int ch_i_track(char ch)
{
  return(from_gps((ch<<2)+0x84));
}

int ch_q_track(char ch)
{
  return(from_gps((ch<<2)+0x85));
}

int ch_i_prompt(char ch)
{
  return(from_gps((ch<<2)+0x86));
}

int ch_q_prompt(char ch)
{
  return(from_gps((ch<<2)+0x87));
}

void ch_accum_reset(char ch)
{
  to_gps((ch<<2)+0x85,0);
}

void ch_code_slew(char ch,int data)
{
  to_gps((ch<<2)+0x84,data);
}

void all_code_slew(int data)
{
  to_gps(0x70,data);
}
```

```
void data_retent_w(int data)
{
  to_gps(0xe4,data);
}

int data_retent_r(void)
{
  return(from_gps(0xe4));
}

void data_bus_test_w(int data)
{
  to_gps(0xf2,data);
}

int data_bus_test_r(void)
{
  return(from_gps(0xf2));
}


inline int meas_status(void)
{
  return(from_gps(0x81));
}

void program_TIC(long data)
{
  unsigned int high,low;
  high=int(data>>16);
  low =int(data & 0xffff);
  to_gps(0x6d,high);
  to_gps(0x6f,low);
}

void reset_cntl(int data)
{
  to_gps(0x7f,data);
}

void ch_carrier(char ch,long freq)
{
  int freq_hi,freq_lo;
  unsigned int add;
  freq_hi=int(freq>>16);
  freq_lo=int(freq&0xffff);
  add=(ch<<3)+3;
  outpw(0x304,add);
  outpw(0x308,freq_hi);
  add++;
  outpw(0x304,add);
  outpw(0x308,freq_lo);
}

void ch_code(char ch,long freq)
{
```

```
  int freq_hi,freq_lo;
  unsigned int add;
  freq_hi=int(freq>>16);
  freq_lo=int(freq&0xffff);
  add=(ch<<3)+5;
  outpw(0x304,add);
  outpw(0x308,freq_hi);
  add++;
  outpw(0x304,add);
  outpw(0x308,freq_lo);
}

void ch_epoch_load(char ch,unsigned int data)
{
  to_gps((ch<<3)+7,data);
}

void ch_on(char ch)
{
  ch_status=ch_status | bit_pat[ch];
  reset_cntl(ch_status);
}

void ch_off(char ch)
{
  ch_status=ch_status & !bit_pat[ch];
  reset_cntl(ch_status);
}

void system_setup(int data)
{
  to_gps(0x7e,data);
}

void test_control(int data)
{
  to_gps(0x7c,data);
}

void status_latch(void)
{
  to_gps(0x80,0);
}

void io_config(int data)
{
  to_gps(0xf0,data);
}
```