DGPS Receiver Software Description

Jim Bixby
bix@san.rr.com
November, 1998

1. Introduction

This document describes the software component of a differential GPS (DGPS) beacon receiver. The receiver can receive DGPS beacon transmissions in the marine beacon band (285-325 kHz) and demodulate the transmissions to extract the digital data stream with the DGPS information. A microprocessor takes that data, and outputs SC104-format digital data to a “DGPS-ready” GPS receiver at 4800 or 9600 baud. Receiver status is displayed on a 16x2 LCD display.

2. Hardware

See the file Hardware.doc in the zip archive for a description of the hardware, as well as a description of the files contained in the zip archive. It is assumed in this description that the reader has read Hardware.doc.

3. General Software Overview

The receiver employs a PIC 16F84 processor from Microchip, and the code is targeted to that processor. Interrupt processing is used to handle the bulk of the processor tasks and is termed subsequently in this description as foreground processing. When the foreground is not busy, control passes to the “main” program, which works in the background to perform status monitoring.

Serial digital data arrives from the receiver on RB4 (PORTB, bit 4), at a rate of 50, 100, or 200 bits/second (in the US, currently, only 100 and 200 bps transmissions are used). A primary task of the processor is to synchronize to, and read, the incoming serial stream. It does this by synchronizing TMR0 to produce an TMR0 Overflow interrupt in the nominal middle of each incoming bit, so that the receiver data can be sampled. In order to do that, the processor must determine the incoming bit rate as well as perform the TMR0 synchronization. The bit synchronization code is activated on each transition of the incoming data.

The bit synchronization algorithm starts with as initial assumption that the bit rate is 50 bps. An interrupt occurs on each incoming bit transition, at which time the phase of TMR0 is read, and compared against the ideal bit-edge value. An adjustment is made to the TMR0 contents depending on whether the actual incoming edge is leading or lagging the ideal location. Also, a check is made using a maximum-likelihood confidence counter to see if this phase is consistent with the bit synchronizer being “locked” to the incoming data -- that is, that the proper period as well as phase has been determined. After a preponderance of indications that the synchronizer is locked, the software finally becomes convinced and declares the synchronizer to be locked, after which the magnitude of the phase corrections is reduced to maintain lock. Alternatively, the code may determine that the synchronizer cannot lock under the bit rate assumption, at which time it changes the bit rate assumption to try again to achieve lock. In this way, the bit synchronizer hunts for and determines the correct bit rate and the correct bit phase.

Once the software bit synchronizer is locked, the code can then read each incoming bit. The incoming data consists of 30-bit words, organized into frames which have variable word length. The first two words of every frame are a two-word frame header, and this header format is identical for every frame. Figure 3.1 shows the header format.

After bit synchronizing, the next objective is to find the word boundaries (the incoming data consists of 30-bit words, with 24 bits of data and 6 bits of parity). It accomplishes this by initially, on each incoming bit, checking for correct parity over the previously received 32 bits (the parity computation involves the last two bits of the previous word and all the bits of the current word). If a parity match is found, the code assumes this might be a valid word boundary, so it goes into a mode to start testing that boundary, every 30 bits, to see if parity remains good. After a preponderance of good parity tests, the code declares itself to be in Word Sync. Alternatively, after a number of failed tests, it declares this candidate position to be incorrect and starts anew looking for a parity match.

After word synchronization is achieved, the next goal is to achieve frame synchronization. DGPS frames are a variable number of words in length, with the length of the frame contained in a frame header, which also contains a fixed 8-bit preamble and a 3-bit sequence counter which increments on each frame (figure 3.1 above). The frame synchronization code initially looks for the preamble in the first 8 bits in each incoming word. When found, it thinks this might actually be the start of a frame, so it saves the sequence number and the frame length. Then, <frame length> words later, it looks again for the preamble, and for the correct sequence number. After a preponderance of such tests being successful, it declares itself to be in Frame Sync. While in frame sync, the Station ID is read from the incoming data and displayed on the LCD. Alternatively, if the preponderance of tests fail, the frame synchronization code declares itself to be out of sync, and starts all over looking for the next candidate start of frame header.

Bit Number	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
	(First in time)																													(Last in time)
First Word	Preamble								Message Type						Station I.D.										Parity					
Second Word	Modified Z-Count													Seq Num			Frame Length					StaHlth			Parity					

Preamble: 0 1 1 0 0 1 1 0	Station I.D.: 10-bit number identifying the beacon transmitter
Message Type: Contains the Frame ID code, identifying the data content in the frame	Seq Num: Counts 0-7, incrementing on each frame
Modified Z-Count: Increments each 0.6 sec	Frame Length: the total frame is 2 words longer than this value
Parity: six bits, computed over the 24 data bits of this word and bits 29 and 30 of the previous word	StaHlth: Station Health Indicates a bad station, or, if good, gives info about the resolution of corrections

Figure 3.1
DGPS Header Format

The bit synchronizer algorithm is triggered by transitions on the incoming data, while the processing to achieve word and frame synchronization is accomplished on TMR0 overflow interrupts when the value of each bit is read. The former occurs at the edge of incoming bit boundaries, and the latter occurs in the middle of incoming bit boundaries. After the word and frame synchronization code has executed on a TMR0 overflow event, the interrupt handler then looks to see if there is any data which needs to be sent to the LCD or to the SC104 serial output. One character can be sent to each port, on each TMR0 interrupt. SC104 data is sent every six incoming data bits, whether the processor believes itself to be synchronized or not. The GPS receiver has its own logic to independently perform word and frame synchronization and extract the data to perform the differential correction. LCD data is sent whenever the background routine determines that a change is needed to the LCD display. The background handles the job of determining exactly what should be sent to the LCD, while the foreground handles the task of actually sending the character.

The TMR0 interrupt handler also schedules a check on each pass to see if the FM Demodulator phase lock loop is indeed locked. It does this by enabling the hardware RB0/INT interrupt. After the TMR0 code has finished and global interrupts are turned back on, the processor will almost immediately be interrupted by the next rising edge of the Carrier VCO signal. When the Carrier VCO transitions high, the Detector signal should be in a low state, if the Carrier VCO is locked to the Detector signal. This is tested and the same maximum-likelihood confidence test is used to determine if the PLL is indeed locked. Further interrupts from this source are then disabled, so that only one test is made per bit period to reduce the load on the processor while still sufficient tests per second to get a timely update of the display.

The final event handled by the foreground is switch presses. There are two switches to control the receiver: UP and DOWN. Pressing UP moves the receiver up one channel (channels are spaced 1 kHz apart) and pressing DOWN moves the receiver down one channel. A switch press causes a processor interrupt, at which time the processor reads the switches to determine which was pressed, computes the new receiver frequency, and sends the needed information to the digital frequency synthesizer which creates the first local oscillator for the receiver. There is provision in the code for a third switch, MODE, intended for possible future expansion of the code to allow, for instance, for automatic scan for a good channel, or test modes. MODE is not implemented in this version of the software. This receiver is also totally manual in operation, and does not accept NMEA control information from a GPS receiver, another arena for possible modification of the software.

4. Initialization

The initialization sequence is to:

Wait for power to come up
Initialize stuff
Turn on interrupts
Initialize the LCD
Display the initial frequency and bit rate

Init:

Called on power up only.

Reads the jumper to determine the SC104 output bit rate (4800 or 9600 baud) and sets Bit_K, a register used to determine the length of one output bit.

Then, initialization data is sent to the frequency synthesizer chip, and the last channel to which the receiver was tuned is read from the processor's eeprom and used to initialize the receiver frequency. Init then resets all of the synchronization status information of the receiver, starts the bit synchronizer off at 50 bps, and exits.

Init_LCD:

Called during the initialization phase only.

First, the LCD is initialized according to the spec for LCDs which use the Hitachi 44780 or compatible controller chip, which covers just about all character displays. The LCD is initialized for 8-bit data interface mode, 5x8 font, no cursor, no blinking. Then, the static portion of the display is sent to the LCD. Figure 4.1 shows the display in normal operation. For the station ID field, this design follows the philosophy of being very critical about when it displays the ID, and if it has the slightest indication that the ID field may be invalid, it displays dashes instead of a possibly incorrect station ID. So initially, the Station ID field is displayed as dashes.

```
302k  ID SICBWFP
102b 262 +++++++
```

Figure 4.1
LCD Display

5. Interrupt Processing

Once the receiver is initialized, the bulk of the processing happens in the foreground via the interrupt handler. Figure 5.1 shows the flowchart of the interrupt processing sequence. On any interrupt from any of the sources, all possible sources are checked. After a source has been determined, the code continues to check the other possible sources anyway, because receiver noise can cause bit edge interrupts at odd times, for instance.

If one could place an ‘activity’ probe on the interrupt service routine, there would be bursts of activity at the middle of incoming bits, when TMR0 overflows to mark the middle of bits, and at bit edges in the middle of the TMR0 count. If requests are pending to send out an LCD character and also an SC104 character, and also it happens to be a word boundary so that the parity, word sync and frame sync algorithms need to run, then it can take about 2 milliseconds to complete this work. The time is dominated by the time needed to send out the SC104 character, which at 4800 baud is 1.87 ms (time is needed for 9 bits, since the routine does not wait around for the stop bit to be timed once sent). Since a bit period at 200 bps for the incoming data is only 5 ms, the TMR0 routine gets done just in time for a bit transition to arrive.

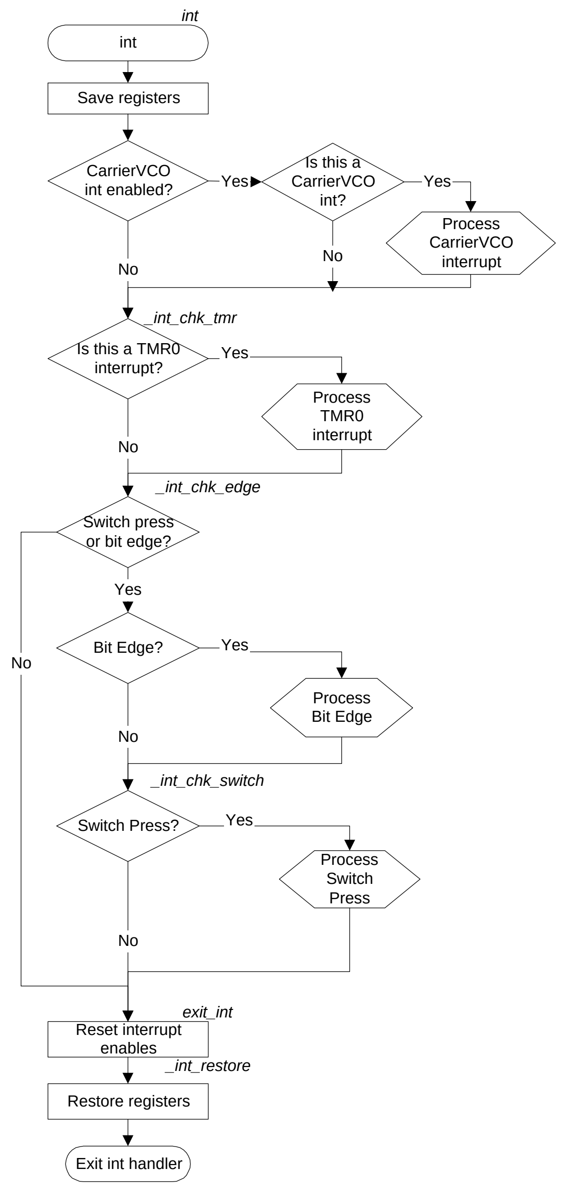


Figure 5.1
int Flowchart

6. CarrierVCO Interrupt

Figure 6.1 shows the flowchart for processing the CarrierVCO interrupt. The CarrierVCO signal is at 3khz, locked to the incoming carrier from the receiver. This interrupt service routine checks to see if the CarrierVCO is indeed locked.

In lock, the CarrierVCO signal should be 90 degrees out of phase from the receiver carrier output signal (named Detector). So, if this interrupt is enabled, it just looks at the Detector signal to see if it is low, and counts a confidence counter up or down. If the confidence counter overflows, then lock is declared, and if it underflows, out-of-lock is declared, for display on the LCD.

This interrupt is enabled when the TMR0 service routine completes, and after executing itself it then disables further CarrierVCO interrupts, until the TMR0 routine reenables it. This limits the rate of checking to the incoming bit rate, so that the processor will not spend an inordinate amount of time doing this test.

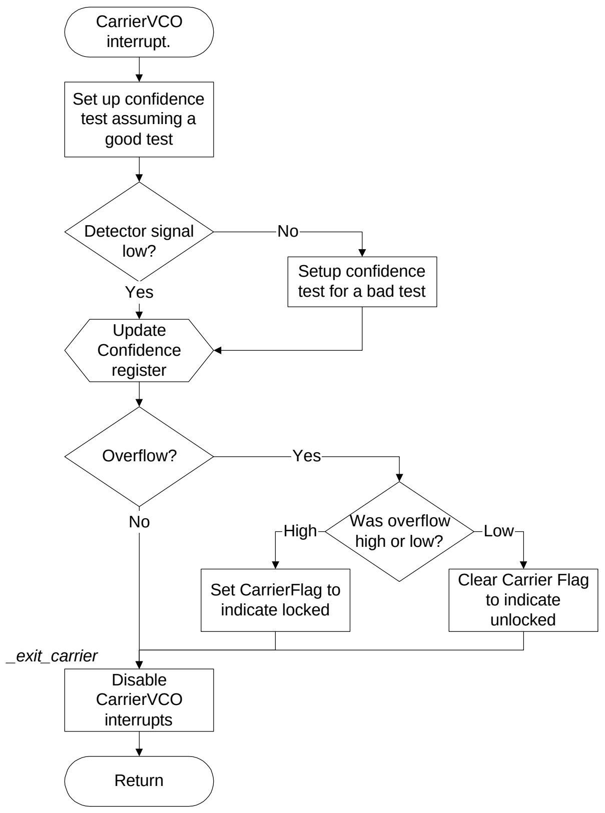

CarrierVCO interrupt.
Set up confidence test assuming a good test
Detector signal low?
No
Setup confidence test for a bad test
Yes
Update Confidence register
Overflow?
Yes
No
High
Was overflow high or low?
Low
Set CarrierFlag to indicate locked
Clear Carrier Flag to indicate unlocked
_exit_carrier
Disable CarrierVCO interrupts
Return

Figure 6.1
CarrierVCO Interrupt

7. TMR0 Interrupt

Figure 7.1 shows the TMR0 Interrupt flowchart. This interrupt occurs in the middle of each incoming data bit from the receiver, and is where the bulk of the synchronization code takes place. After immediately reloading the timer to start another bit interval, it reads the incoming data bit, and shifts the bit into the SC104 output character, which when sent out contains the last six received bits. When the six bits have been received, a request flag is set to tell the serial output routine to send it later.

Then, the bit is shifted into a long accumulator (Data_A thru Data_E) which contains the last 40 bits received. The objective of the word sync algorithm is to figure out when the 24 data bits of an incoming word are aligned in Data_B, Data_C and Data_D. This is explained in detail later in this document. After word sync has been achieved, the service routine then does a similar algorithm to determine where frame boundaries are. Finally, the Serial Output routines handles any pending output requests and the interrupt is released.

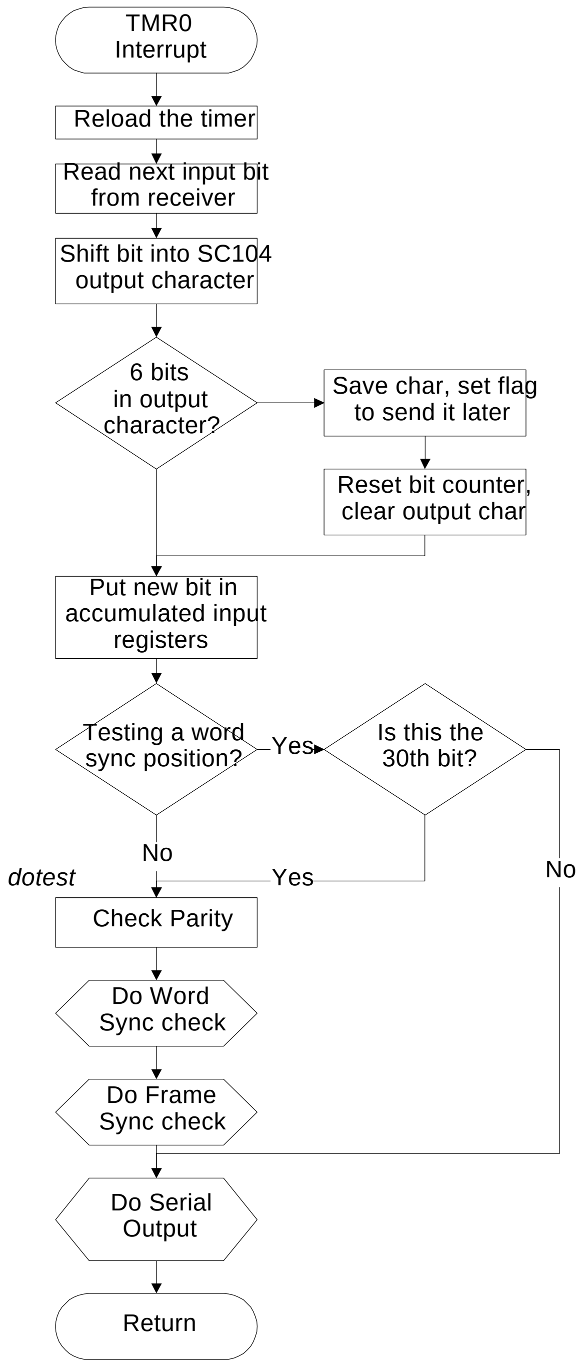

TMR0 Interrupt
Reload the timer
Read next input bit from receiver
Shift bit into SC104 output character
6 bits in output character?
Save char, set flag to send it later
Reset bit counter, clear output char
Put new bit in accumulated input registers
Testing a word sync position?
Yes
Is this the 30th bit?
No
Yes
No
dotest
Check Parity
Do Word Sync check
Do Frame Sync check
Do Serial Output
Return

Figure 7.1
TMR0 Interrupt

8. Word Sync Algorithm

Figure 8.1 shows the Word Sync algorithm flowchart. The processor starts out checking on every incoming bit to see if parity is good, which means that this bit position might indeed be a word boundary. Once a position with good parity is found, a flag TrialWordSync is set to indicate that we should switch to checking only every 30 bits. This flag stays true until the confidence test indicates that this is not in fact a valid word boundary. The results of the confidence test are monitored to make the decisions as to whether to declare this position correct, or incorrect and start the synchronization process over, or to just continue testing.

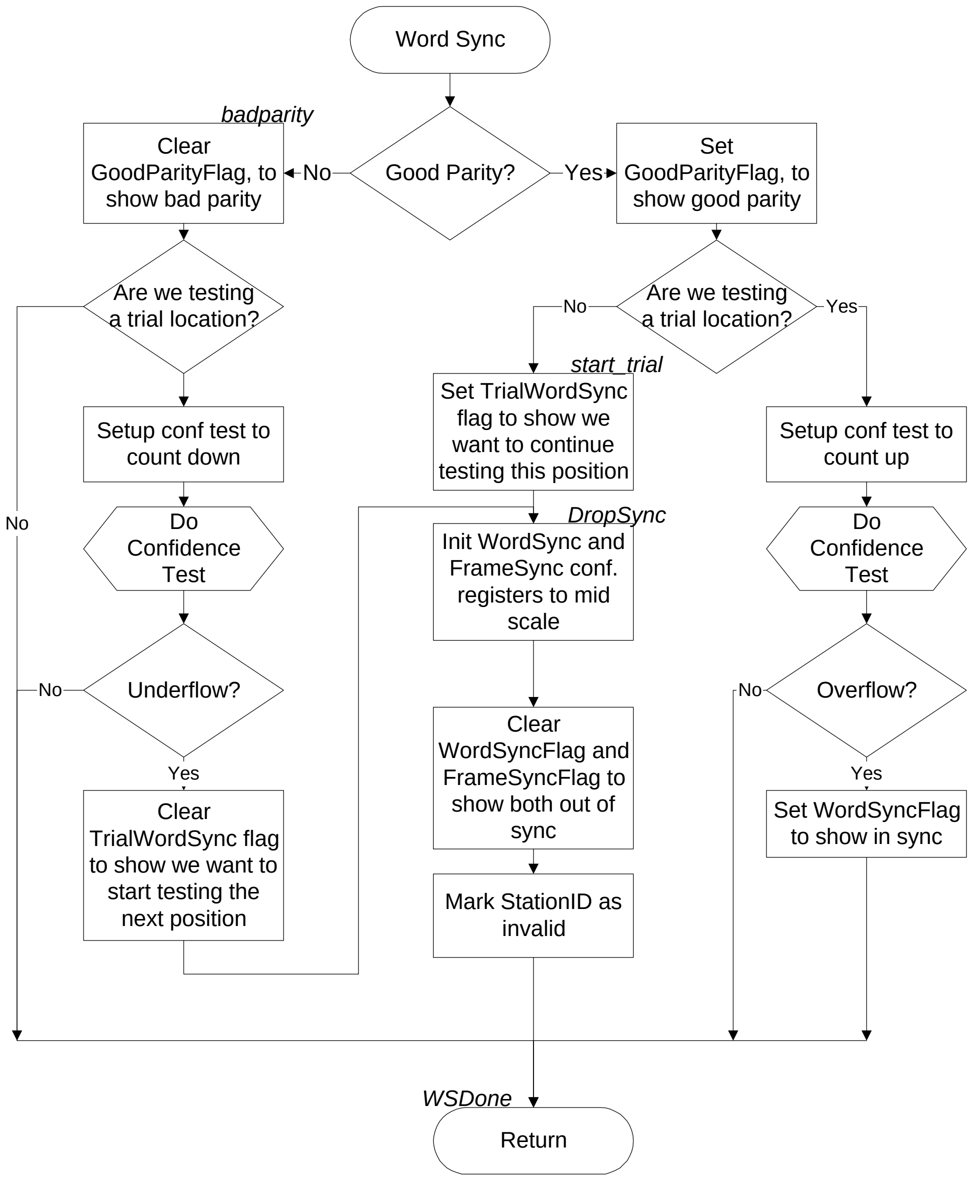
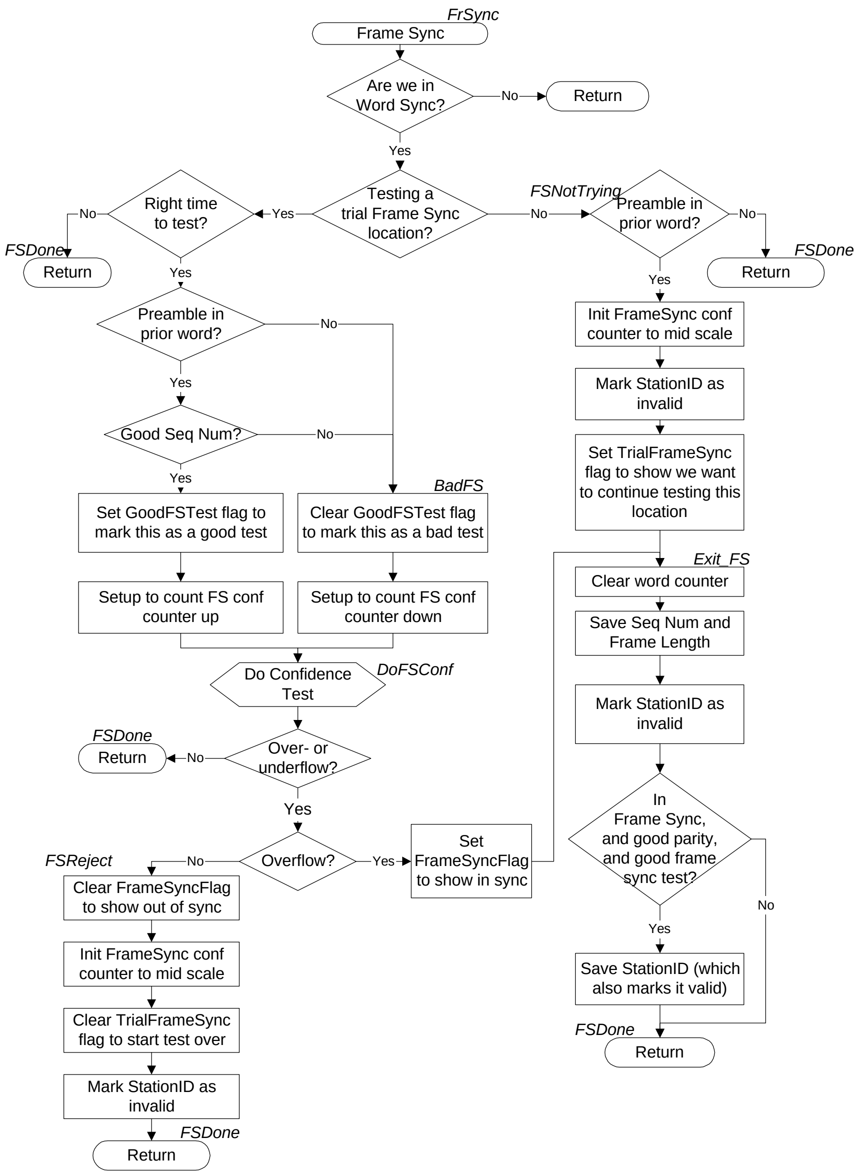

Word Sync
Good Parity?
No
Yes
badparity
Clear GoodParityFlag, to show bad parity
Set GoodParityFlag, to show good parity
Are we testing a trial location?
No
Yes
start_trial
Set TrialWordSync flag to show we want to continue testing this position
Setup conf test to count down
Setup conf test to count up
Do Confidence Test
DropSync
Init WordSync and FrameSync conf. registers to mid scale
Underflow?
Overflow?
Clear WordSyncFlag and FrameSyncFlag to show both out of sync
Clear TrialWordSync flag to show we want to start testing the next position
Set WordSyncFlag to show in sync
Mark StationID as invalid
WSDone
Return

Figure 8.1
Word Sync Algorithm

9. Frame Sync Algorithm

Figure 9.1 shows the Frame Sync algorithm flowchart. If we are not in Word Sync, it exits immediately. Otherwise, it uses a flag TrialFrameSync to determine if we are testing every word boundary to find the first candidate frame boundary, or if we have advanced to testing candidate frame boundaries. The test for frame sync is a bit more complicated than that for word sync. DGPS frames contain a variable number of words, indicated in the frame header, which also contains a three-bit sequence number which increments on each frame. The frame sync test criteria is the preamble is found in the right position, and that the sequence number is correct, at the right word indicated by the frame length from the previous frame.


FrSync
Frame Sync
Are we in Word Sync?
No
Return
Yes
Testing a trial Frame Sync location?
Yes
No
FSNotTrying
Right time to test?
No
FSDone
Return
Yes
Preamble in prior word?
No
Yes
Good Seq Num?
No
Yes
BadFS
Set GoodFSTest flag to mark this as a good test
Clear GoodFSTest flag to mark this as a bad test
Setup to count FS conf counter up
Setup to count FS conf counter down
Do Confidence Test
DoFSConf
FSDone
Return
No
Over- or underflow?
Yes
FSReject
No
Overflow?
Yes
Set FrameSyncFlag to show in sync
Clear FrameSyncFlag to show out of sync
Init FrameSync conf counter to mid scale
Clear TrialFrameSync flag to start test over
Mark StationID as invalid
FSDone
Return
Preamble in prior word?
No
FSDone
Return
Yes
Init FrameSync conf counter to mid scale
Mark StationID as invalid
Set TrialFrameSync flag to show we want to continue testing this location
Exit_FS
Clear word counter
Save Seq Num and Frame Length
Mark StationID as invalid
In Frame Sync, and good parity, and good frame sync test?
No
Yes
Save StationID (which also marks it valid)
FSDone
Return

Once frame sync is achieved, this routine also picks out and saves the StationID for display on the LCD. It is very fussy about accepting data for display, and if it gets a hint that the StationID field may be incorrect, it marks it as invalid by setting the sign bit. The background routine detects this change, and replaces the ID display with dashes.

Figure 9.1
Frame Sync Algorithm

10. Serial Output

Figure 10.1 shows the Serial Output flowchart, which is very straight forward. Output is allowed only during the TMR0 interrupt service routine, one character at a time to each of the LCD display and the serial SC104 output. LCD character outputs are requested only by the background routine, by setting LCDFlag, and SC104 outputs are requested only by the TMR0 service routine. Upon outputting the requested character, this routine clears the request flags to indicate that the operation was completed.

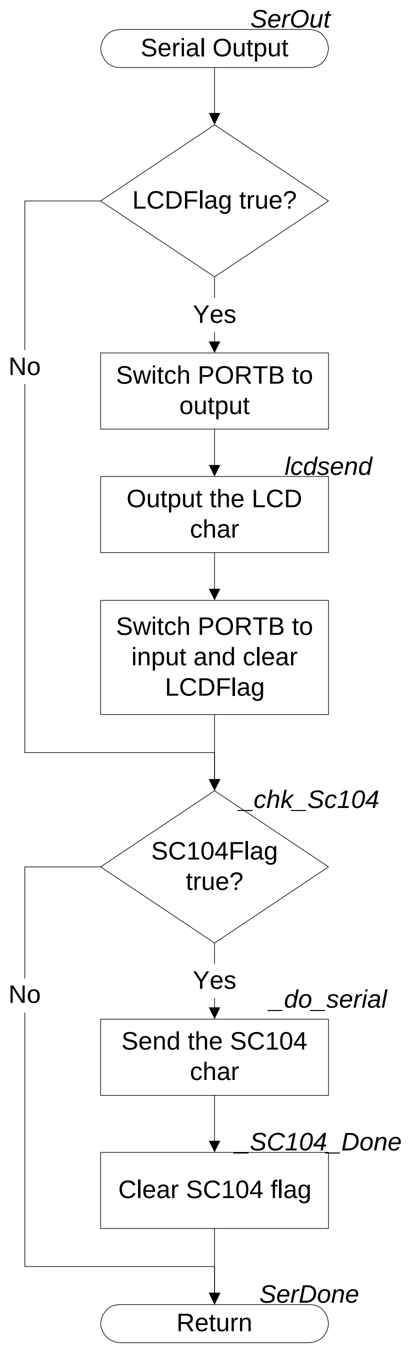


Figure 10.1
Serial Output Flowchart

11. Bit Edge Interrupt

Figure 11.1 shows the flowchart for processing a bit edge, which is used for adjusting the TMR0 phase so that it overflows in the middle of incoming data bits, and is the process of achieving bit synchronization.

On each detected the edge, the routine reads the TMR0 value, and compares it against an ideal value (SyncCenter). If the location of the bit edge indicates that the TMR0 value should be adjusted (which changes the timing of when TMR0 overflow interrupts occur relative to the incoming bit stream), an adjustment is made.

This bit synchronizer uses two constants: PhaseLimit and Window. PhaseLimit is the absolute value of the magnitude of corrections allowed to be made to TMR0, and Window is the absolute value of the number of counts to either side of SyncCenter during which a bit edge event is determined to indicate that the synchronizer is locked -- edges which occur outside of Window are taken as indications that the bit synchronizer is not locked. PhaseLimit must be smaller than Window for this routine to operate correctly.

If an incoming edge is within PhaseLimit and the synchronizer is not locked, then the measured error is used to change TMR0. If the incoming edge is outside of PhaseLimit and the synchronizer is not locked, then the correction to TMR0 is limited to PhaseLimit, rather than the full error. This is done to prevent spurious noise pulses from unduly influencing TMR0 even when not locked.

If the synchronizer is locked, then the magnitude of the correction to TMR0 is limited to only plus or minus one count -- once lock has been achieved, we really don’t want noise disrupting the phase of TMR0.

The test for determining whether lock has been achieved or not is the (now familiar) confidence test used for all other sync determinations by this program. If the algorithm determines it is out of lock, it tries to lock at the next higher bit rate, unless it was already trying at 200 bps in which case it rolls back to 50 bps. In this way, the algorithm automatically hunts for and determines the correct bit rate.

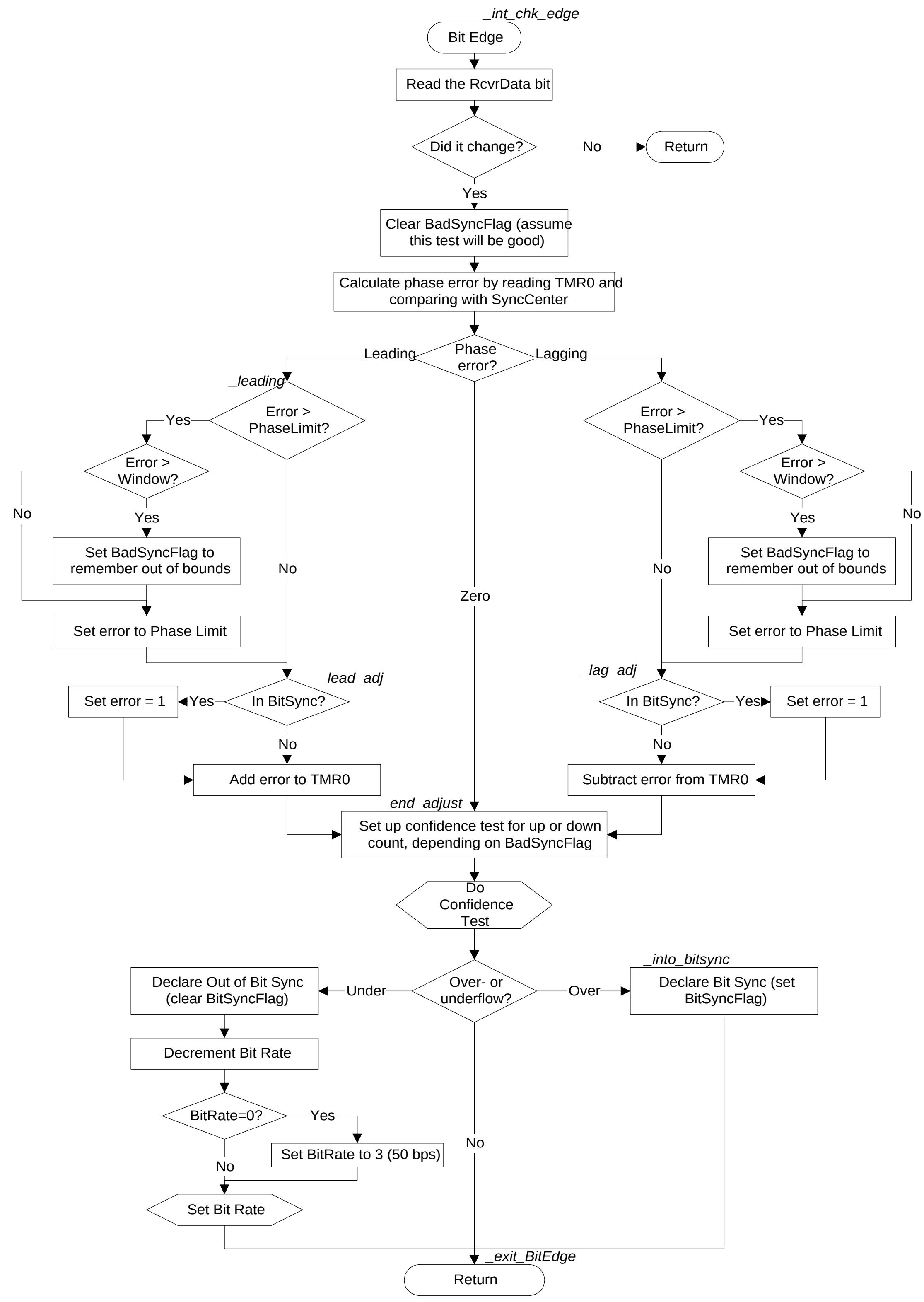


Figure 11.1
Bit Edge Interrupt Flowchart

12. Switch Press Interrupt

Figure 12.1 shows the Switch Press flowchart. Switch presses on the UP or Down pushbuttons cause an interrupt throught the PIC's interrupt-on-change capability. This routine simply determines which (if any) of the switches were pressed, and increments Chan, the channel number, appropriately. Then it calls Set_PLL to output the new frequency to the LO synthesizer chip, and sets a flag (LCDFreqUpdRqst) to request the background routine to update the LCD display.

This receiver operates on channel spacings of 1 khz, so there are 41 channels in the marine beacon band of 285-325 khz. Chan has values of zero to 40 to specify the channel, with Chan=0 corresponding to 285 khz. If Chan is incremented above 40 it rolls over to zero, and if it is decremented below zero it rolls around to 40.

At the conclusion of incrementing or decrementing Chan, the routine debounces the switch by sitting in a loop waiting for both switches to be up before exiting.

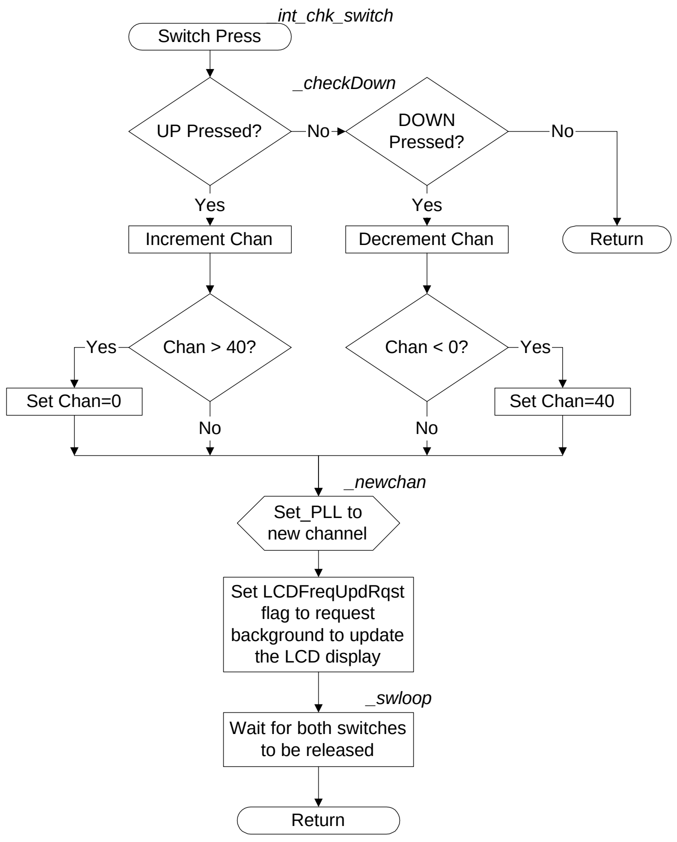


Figure 12.1
Switch Press Interrupt Flowchart

13. Confidence Test

Figure 13.1 shows the Confidence Test flowchart. This test is used to make decisions about whether the CarrierVCO is locked to the incoming signal, about bit sync lock status, and about word and frame sync lock status. This test amounts to a maximum likelihood test, and it tests a hypthosis until sufficient confidence has been achieved to declare the hypothesis true or false. At any point in time, the confidence counter value represents a measure of the probability that the hypothesis is true. It is a very sophisticated test which happens also to be computationally very efficient and easy to optimize by trial and error.

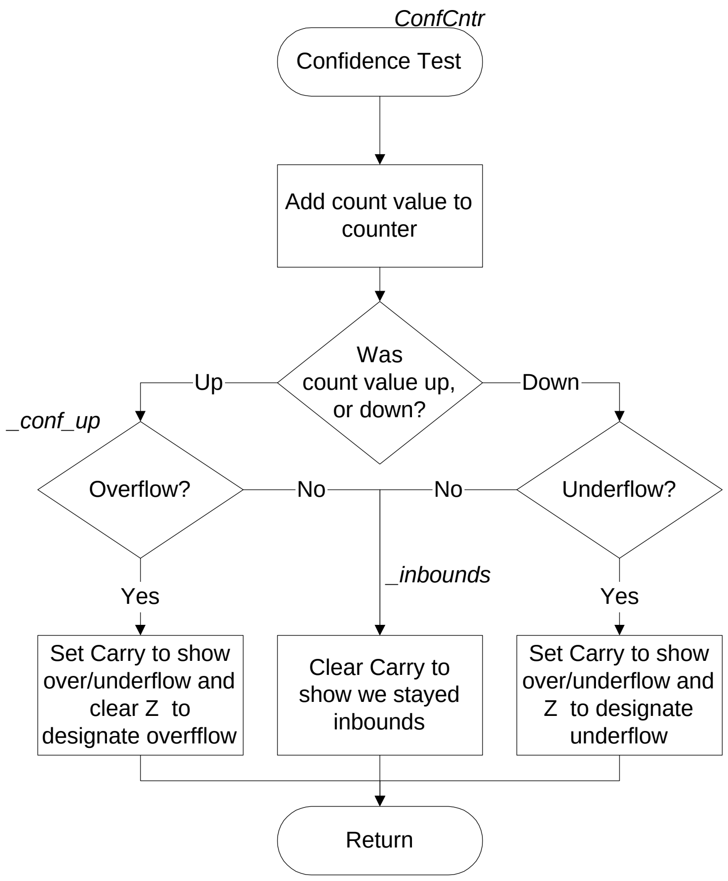

ConfCntr
Confidence Test
Add count value to counter
Was count value up, or down?
Up
Down
_conf_up
Overflow?
No
No
Underflow?
_inbounds
Yes
Yes
Set Carry to show over/underflow and clear Z to designate overfflow
Clear Carry to show we stayed inbounds
Set Carry to show over/underflow and Z to designate underflow
Return

Figure 13.1
Confidence Test Flowchart

Lets take the case of bit synchronization. In that case, we start with the hypothesis that the bit synchronizer has determined the correct phase of the incoming data and is locked. If bit edges fall within a window of the right time, such events tend to confirm the hypothesis. If bit edges fall outside of the window, such events tend to make us disbelieve the hypothesis. At any time, there are probabilities that a bit edge will occur within the window, influenced by both channel noise which can cause spurious edges to occur and by the actual phase of the bit synchronizer (ie, actually locked or not). If a bit edge occurs within the window, then an amount (up-count value) is added to the confidence register. If a bit edge occurs outside of the window, then an amount (down-count value) is subtracted from the confidence register. If the counter overflows at the high end, then we say we have developed enough confidence to declare the synchronizer to be locked, and if the counter underflows at the bottom end, then we say we have developed enough confidence to declare the synchronizer to be out of lock. In between, we say that we simply have not developed sufficient confidence to make a call. Mathematically, this counter operation amounts to adding and subtracting logarithms of conditional probabilities of a sequence of observations.

The ratio of the up-count to down-count values is determined by the probabilities of each indication, given the hypothesis that we are locked. Thus, if the channel is very quiet and false bit edges never occur, the ratio could be very large. On the other hand, if there is noise, then the ratio should be reduced to reflect the notion that each event is telling us less, in the presence of noise.

The overall magnitude of the count values relative to the counter overflow limits determines how quickly the test can reach a conclusion, and their optimum values are also set by the apriori probability that the hypothesis is true, by the amount of noise in the indications, and by the desired confidence level.

14. Set_PLL

This routine converts the Chan value to a frequency, and outputs it to the frequency synthesizer chip. Since it is only called on frequency changes, it also initializes the bit synchronizer to start testing at 50 bps, and resets the sync status of all of the sync indicators.

15. Set Bit Rate Routine

This routine sets up TMR0 and its prescaler to count at a period equal to the assumed bit rate.

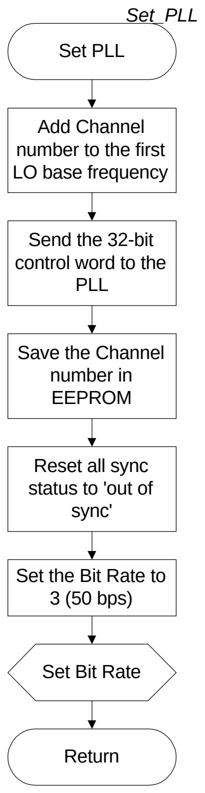

Set_PLL
Set PLL
Add Channel number to the first LO base frequency
Send the 32-bit control word to the PLL
Save the Channel number in EEPROM
Reset all sync status to 'out of sync'
Set the Bit Rate to 3 (50 bps)
Set Bit Rate
Return

Figure 14.1
Set PLL
Flowchart

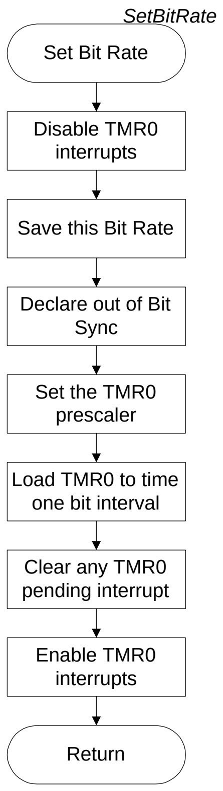


Figure 15.1
Set Bit Rate
Flowchart

16. Background Processing

Figure 16.1 shows the flowchart for background processing, which is being executed whenever the interrupt handler is not busy. Basically, it constantly monitors for changes in synchronization and lock status, and updates the LCD display.

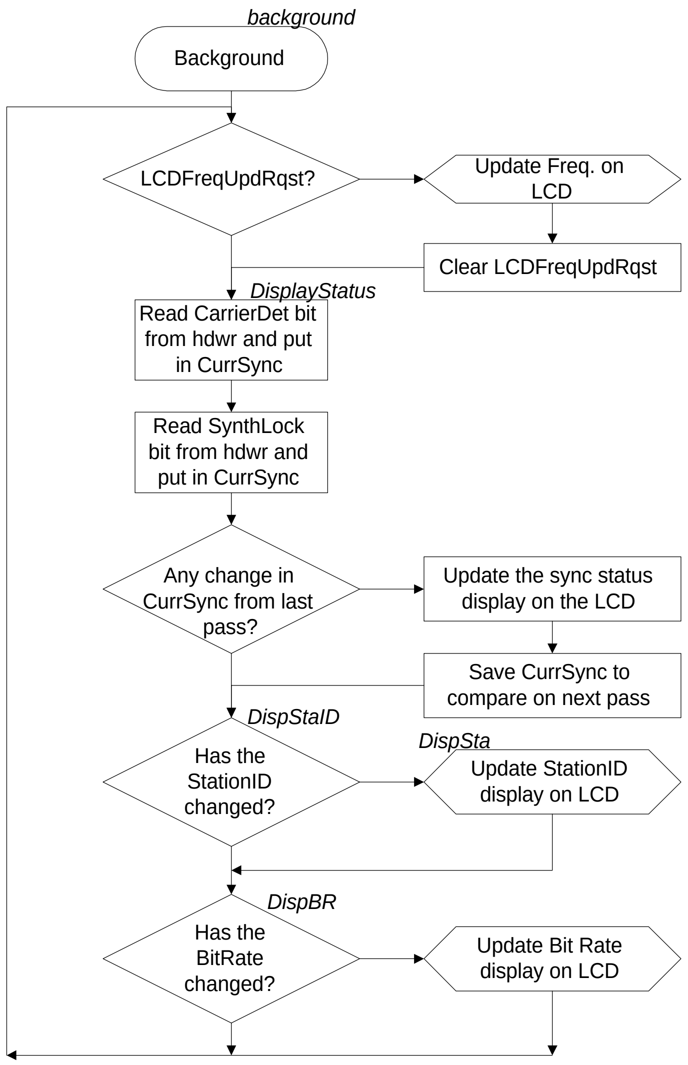

background
Background
LCDFreqUpdRqst?
Update Freq. on LCD
Clear LCDFreqUpdRqst
DisplayStatus
Read CarrierDet bit from hdwr and put in CurrSync
Read SynthLock bit from hdwr and put in CurrSync
Any change in CurrSync from last pass?
Update the sync status display on the LCD
Save CurrSync to compare on next pass
DispStaID
Has the StationID changed?
DispSta
Update StationID display on LCD
DispBR
Has the BitRate changed?
Update Bit Rate display on LCD

Figure 16.1
Background Flowchart

References:

RTCM Recommended Standards for Differential NAVSTAR GPS Service, Version 21. RTCM Spcial Committee No. 104. May be purchased from

Radio Technical Commission for Maritime Services
655 Fifteenth Street, NW, Suite 300
Washington, D.C. 20005

MC145162 Data Sheet
May be downloaded from
http://mot2.indirect.com/books/dl110/pdf/mc145162rev3-1.pdf

PIC 16F84 manual
May be downloaded from
http://www.microchip.com/download/lit/micros/30430c.pdf

Interface Control Document, IDC200
May be downloaded from
http://www.navcen.uscg.mil/gps/geninfo/gpsdocuments/icd200/icd200c.pdf
(It contains the details of the parity calculation, which is the same as used in GPS receivers.)

LCD Technical Reference FAQ:
ftp://toleak.etec.wwu.edu/pub/lcd_faq.txt