GPS Signal Description

1. The baseband signal **transmited** by the satellite is given as

$$S(t) = S_{PPS}(t) + jS_{SPS}(t) \tag{1}$$

- $S_{SPS}(t) = \sum_{i=-\infty}^{\infty} c_{sps}(|i|_{L\_sps}).d([i]_{CD\_sps}).rect_{T_{c,sps}}(t - iT_{c,sps})$ — Standard Positioning Service
- $S_{PPS}(t) = \sum_{i=-\infty}^{\infty} c_{pps}(|i|_{L\_pps}).d([i]_{CD\_pps}).rect_{T_{c,pps}}(t - iT_{c,pps})$ —- Precision Positioning Service

2. Let $x_{in}[n]$ be the incoming signal at the **receiver** end and is given as

$$x_{in}[n] = A(t)s_T(t - \tau(t))e^{j(2\pi f_D(t)t+\phi(t))}|_{t=nT_s} + n(t)|_{t=nT_s} \tag{2}$$

where

- $A(t)$ is Amplitude
- $s_T(t)$ is Complex baseband signal
- $\tau(t)$ is code delay(time varying)
- $f_D(t)$ is Doppler shift(time varying)
- $\phi(t)$ is carrier phase shift(time varying)
- $n(t)$ is Random noise with zero mean
- T_s is Sampling period
- f_s is Sampling frequency

Pseudo code for GPS Signal Acquisition

Functions for computing the PRN codes of GPS satellite

(a) g1_lfsr()

```
uint16 state = 0x3FF
uint8 out[128]
uint8 new_bit
for i=0 to i=1022
    if i % 8 == 0
        out[i/8] = 0x00;
    out[i/8] |= ((state ≫ 9) & 0x1) << (7 - i%8)
    new_bit = ((state ≫ 9) ⊕ (state ≫ 2)) & 0x1
    state = ((state ≪ 1) | new_bit) & 0x3FF
end for
return out
```

(b) g2_lfsr(**uint8** tap0,**uint8** tap1)

```
uint16 state = 0x3FF
uint8 out[128]
tap0 = tap0-1
tap1 = tap1-1
for i=0 to i=1022
    if i % 8 == 0
        out[i/8] = 0x00;
    out[i/8] |= (((state ≫ tap0) ⊕ (state ≫ tap1)) & 0x1) << (7 - i%8)
    new_bit = ((state ≫ 9) ⊕ (state ≫ 8) ⊕ (state ≫ 7) ⊕ (state ≫ 5) ⊕ (state ≫ 2) ⊕ (state ≫ 1)) & 0x1
    state = ((state ≪ 1) | new_bit) & 0x3FF
end for
return out
```

(c) combine_g1_g2(**uint8** *g1,**uint8** *g2)

    uint8 out[128]
    for i=0 to **i**=127
        out[**i**] = g1[**i**] $\oplus$ g2[**i**]
    end for
    return out

Function for the Acquisition of GPS signals

GPS_signal_acquisition(int8 *incoming_samples,int16 start_freq,int16 end_freq,int16 step,int16 N)

start function

/** Compute power of incoming signal **/

uint8 *incoming_signal_power

/** The below function was later implemented to take 1 byte and seperate I and Q and compute power and give output in 4 bits. **/

CEVA_DSP_LIB_MAT_CX_MUL_TRANS_Q15(**incoming_samples**,1,**N**,**incoming_signal_power**)

The power of incoming signal should be **incoming_signal_power** > **threshold** . If true, **proceed to** below steps else, **stop** the process.

/******* Declarations of variables *******/

uint8 SVs[32][2] = { {2, 6}, {3, 7}, {4, 8}, {5, 9}, {1, 9}, {2, 10}, {1, 8}, {2, 9}, {3, 10}, {2, 3}, {3, 4}, {5, 6}, {6, 7}, {7, 8}, {8, 9}, {9, 10}, {1, 4}, {2, 5}, {3, 6}, {4, 7}, {5, 8}, {6, 9}, {1, 3}, {4, 6}, {5, 7}, {6, 8}, {7, 9}, {8, 10}, {1, 6}, {2, 7}, {3, 8}, {4, 9} };

uint8 g1[128] /* Array for g1 LFSR */

uint8 g2[128] /* Array for g2 LFSR */

g1 = g1_lfsr() /* Fucntion call */

uint8 tap0,tap1

uint8 gold_code[128]

int8 bpsk_code[1023]

int8 upsampled_code[N]

int8 code_fft[N]

uint8 no_of_coherent = 2

uint8 no_of_non_coherent = 1

int16 angles[4096]

int8 cos_sin_out[2N]

int8 shifted_signal[2N]

int8 coherent_product[N]

int8 non_coherent_product[N]

int8 signal_one[N]

int8 signal_one_fft[N]

int8 Mul_signal[N]

int8 IFFT_signal[N]

int8 power[N]

int16 max_power[5] = {0}

int8 visible_satellites_withMaxPower[5] = {0}

int16 codePhase[5] = {0}

int16 frequency_offset[5] = {0}

```
uint8 visible_PRN_codes[5][128] = {{0}}      /* Matrix to store the PRN codes of visible satellites */
int16 peak_indices[5]
int16 other_indices[N-5]

/******This loop will iterate for all 32 satellites and find the frequency offsets and codephase for all visible satellites
*******/

for sv=0 to sv=31
    /******** PRN code generation ********/
    uint8 index=0      /* index for iterating the SVs arraay */
    tap0 = SVs[sv][index]
    tap1 = SVs[sv][index++]
    g2 = g2_lfsr(tap0,tap1) /* Fucntion call */
    gold_code = combine_g1_g2(g1,g2) /* Fucntion call */

    /** Apply BPSK modulation to the gold code. In bpsk modulation 0 is mapped to 1 and 1 is mapped to -1.
    In order to compute fft in 4 bits for each byte the first 4 bits is real and second 4 bits is imaginary. **/

    uint16 p = 0
    for i=0 to i=127
       for j=7 to j >= 0
         if (gold_code[i] >> j) & 1
           bpsk_code[p] = 0XF0
         else
           bpsk_code[p] = 0x10
         if p == 1022
           break
         p = p + 1
         j = j - 1
       end for
    end for

/* Upsampling the PRN code */
for i = 0 to i = 1022
    upsampled_code[2i] = bpsk_code[i]
    upsampled_code[2i + 1] = bpsk_code[i]
end for
upsampled_code[2046] = 0
upsampled_code[2047] = 0

/**** The FFT function computes the fft of upsampled_code of size 2048 and stores the output in code_fft of size
2048 such that first nibble is real number and second number is imaginary number ****/
fft(code_fft,upsampled_code,N)

conjugate(code_fft,N)
max_of_max = 0
for doppler = start_freq to doppler = end_freq

    /********* Computing the x[n]e^{-j2πF_D t}, for n = 0 to 2047 *********/
    for i = 0 to i = 2N -1
```

```
    angles[i] = (2 * pi * doppler * i * scalingFactor)/2048000
end for

/** The output of the cossin function should be the array of size 4096 and in each byte 1st 4 bits is cos values
and 2nd 4 bits is sin values **/
CEVA_DSP_LIB_COSSIN_Q7(angles,cos_sin_out,2N)

/**Multiply the incoming signal with cos_sin_out such that the resultant signal should have the size of 4096
with each element of size 1 byte such that in each byte first nibble is real number and second nibble is
imaginary ***/
Complex_mul(shifted_signal,cos_sin_out,incoming_samples,2N)

/**** Initialize the array with zeros *****/
non_coherent_product[0:2048] = 0
start_index = 0
end_index = start_index +N-1
for non_coh = 0 to non_coh = no_of_non_coherent - 1
    /**** Initialize the array with zeros *****/
    coherent_product[0:N] = 0
    for coh = 0 to coh = no_of_coherent - 1

      /***** Collecting 1 msec of samples ******/
      signal_one_msec[0:N] = shifted_signal[start_index : end_index ]

      fft(signal_one_fft,signal_one_msec,N)

      Complex_mul(Mul_signal, signal_one_fft , code_fft,N)

      coherent_product = coherent_product + Mul_signal
      start_index = start_index + N
      end_index= end_index + N
    end for

    ifft(IFFT_signal,coherent_product,N)

    square_real_imaginary(sig_power , IFFT_signal , N)

    non_coherent_product = non_coherent_product + sig_power

    /***** Finding the maximum value in non_coherent_product *********/
    int8 max = 0
    for n=0 to n = N-1
      if non_coherent_product[n] > max:
        max = non_coherent_product[n]
        max_index = n
      end if
    end for

    /****** Compute SNR of the signal ********/

    /****** Finding the peak_indices *******/
    for i = -2 to i = 2
```

```
            int16 index = (max_index + i + N)%N
            peak_indices[i + 2] = index
          end for
          /****** Finding the other_indices *******/
          j = 0
          for i = 0 to i = N
            if (i ! = peak_indices[0] && i ! = peak_indices[1] && i ! = peak_indices[2] && i ! = peak_indices[3] &&
            i ! = peak_indices[4])
              other_indices[j++] = i
          end for
          /***** removing the peak_indices *********/
          for i = 0 to i = 5
            non_coherent_product[ peak_indices[i]] = 0
          end for
          /******** computing the noise **********/
          noise = 0
          for i = 0 to i = N
            noise = (noise + non_coherent_product[i])/(N-5)
          end for
          max_value = max/noise
        end for

        /****** Finding the maximum value out of all 101 frequency offset *******/
        if max_value > max_of_max
          max_of_max = max_value
          max_of_max_index = max_index
          doppler_frequency = doppler
        end if
        doppler = doppler + step
      end for
      /*********Find top 5 power,codephase,sat id and doppler frequency *************/
      for i = 0 to i = 4
        if max_of_max > max_power[i]
          for j = 4 to j = i-1
            max_power[j] = max_power[j-1]
            visible_satellites_withMaxPower[j] = visible_satellites_withMaxPower[j-1]
            codePhase[j] = codePhase[j-1]
            frequency_offset[j] = frequency_offset[j-1]
            j = j-1
          end for
          max_power[i] = max_of_max
          visible_satellites_withMaxPower[i] = sv
          codePhase[i] = max_of_max_index
          frequency_offset[i] = doppler_frequency
          for b = 0 to b = 128
            visible_PRN_codes[i][b] = gold_code[b]
          end for
          break the loop
        end if
      end for
    end for
    return visible_satellites_withMaxPower , codePhase , frequency_offset , visible_PRN_codes
end function
```

Cold start

1. Receive the 2 msec of GPS L1 samples from DFE.

   /*** Do the acquisition for -25KHz to 25KHz in the step of 500Hz

2. **visible_satellites_withMaxPower , codePhase , frequency_offset , visible_PRN_codes** = **GPS_signal_acquisition(incoming_samples,-25000,25000,500,2048)**
3. frequency_drift = mean(frequency_offset)
4. Correct the clock with above frequency_drift
5. Collect 2 msec of samples.

   /*** Do the acquisition for -5KHz to 5KHz in the step of 500Hz

6. **visible_satellites_withMaxPower , codePhase , frequency_offset , visible_PRN_codes** = **GPS_signal_acquisition(incoming_samples,-5000,5000,500,2048)**
7. Pass the above values to the tracking.

Warm start

1. Receive the 2 msec of GPS L1 samples from DFE.

   /*** Do the acquisition for -5KHz to 5KHz in the step of 500Hz

2. **visible_satellites_withMaxPower , codePhase , frequency_offset , visible_PRN_codes** = **GPS_signal_acquisition(incoming_samples,-5000,5000,500,2048)**
3. Pass the above values to the tracking.