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

1.1 Functions for computing the PRN codes of GPS satellite

1. (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=1022
    out[i] = g1[i] ⊕ g2[i]
end for
return out
```

1.2 Main Algorithm For GPS acquisition

1. **int8** incoming_samples[4096] /** Receive the 2 msec of GPS L1 samples from DFE. **/

   /** Compute power of incoming signal **/

2. **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. **/

3. **CEVA_DSP_LIB_MAT_CX_MUL_TRANS_Q15(incoming_samples,1,2048,incoming_signal_power)**
4. The power of incoming signal should be **incoming_signal_power** > **threshold** . If true, **proceed to** below steps else, **stop** the process.
5. **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} };
6. **uint8** g1[128] /* Array for g1 LFSR */
7. **uint8** g2[128] /* Array for g2 LFSR */
8. g1 = g1_lfsr() /* Fucntion call */
9. **uint8** tap0,tap1
10. **uint8** gold_code[128]
11. **int8** bpsk_code[1023]
12. **int8** upsampled_code[2048]
13. **int8** code_fft[2048]
14. **uint8** coherent = 2
15. **uint8** non_coherent = 1
16. **int16** angles[4096]
17. **int8** cos_sin_out[4096]
18. **int8** shifted_signal[4096]
19. **int8** coherent_product[2048]
20. **int8** non_coherent_product[2048]
21. **int8** signal_one[2048]
22. **int8** signal_one_fft[2048]

23. **int8** Mul_signal[2048]
24. **int8** IFFT_signal[2048]
25. **int8** power[2048]
26. **int16** max_power[5] = {0}
27. **int8** visible_satellites_withMaxPower[5] = {0}
28. **int16** codePhase[5] = {0}
29. **int16** frequency_offset[5] = {0}
30. **uint8** visible_PRN_codes[5][128] = {{0}} /* Matrix to store the PRN codes of visible satellites */
31. **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. **/
uint8 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

max_of_max = 0

for doppler = -25000 to **doppler** = 25000

/**** 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,2048)

conjugate(code_fft,2048)

/********* Computing the x[n]e^{-j2πF_D t}, for n = 0 to 2047 *********/
for i = 0 to i = 4095
    angles[i] = (2 * pi * doppler * i * 10430)/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_Q15(angles,cos_sin_out,4096)

/**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,4096)

for i = 0 to i = 2047
    non_coherent_product[i] = 0
end for
start_index = 0
end_index = start_index +2047
for non_coh = 0 to non_coh = non_coherent
    for i = 0 to i = 2047
        coherent_product[i] = 0
    end for
    for coh = 0 to coh = coherent
        z = 0
        for i = start_index to i = end_index
            signal_one[z] = shifted_signal[i]
            z = z + 1
        end for
        fft(signal_one_fft,signal_one,2048)
        Complex_mul(Mul_signal, signal_one_fft , code_fft,2048)
        for i = 0 to i = 2047
            coherent_product[i] = coherent_product[i] + Mul_signal[i]
        end for
        start_index = start_index + 2048
        end_index= end_index + 2048
    end for

    ifft(IFFT_signal,coherent_product,2048)

    absolute(power , IFFT_signal , 2048)

    for i = 0 to i = 2047
        non_coherent_product[i] = non_coherent_product[i] + power[i]
    end for
```

```
            int8 max = 0
            for n=0 to n = 2047
                if non_coherent_product[n] > max:
                    max = non_coherent_product[n]
                    max_index = n
                end if
            end for
            if (max_index - 5) >= 0
                start = max_index - 5
            else
                start = 0
            if (max_index + 5) < 2048
                end = max_index + 5
            else
                end = 2047
            elements_to_delete = end + start + 1
            for i = end + 1 to i = 2047
                non_coherent_product[i - elements_to_delete] = non_coherent_product[i]
            end for
            noise = 0
            for i = 0 to i = 2042
                noise = noise + non_coherent_product[i]
            end for
            max_value = max / noise
        end for
        if max_value > max_of_max
            max_of_max = max_value
            max_of_max_index = max_index
            doppler_frequency = doppler
        end if
        doppler = doppler + 500
    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
```