Greta Oto Firmware 设计说明

Jun Mo

Globsky Technology Inc.

2021/7/20

版权信息

1. 软件整体架构

软件主要分成四部分：第一部分是基带控制，主要是卫星信号的捕获以及跟踪；第二部分是位置解算，主要是导航电文解码、原始观测量生成以及根据原始观测量计算接收机位置速度；第三部分是用户接口协议及输入输出；第四部分是底层支持，包括硬件和操作系统的抽象层。通过将访问底层基带硬件的操作抽象出来，使得 firmware 可以运行在真实的硬件上，也可以运行在 C Model 或者以 SignalSim 为基础构成的信号模拟平台上。

1.1 源代码组织

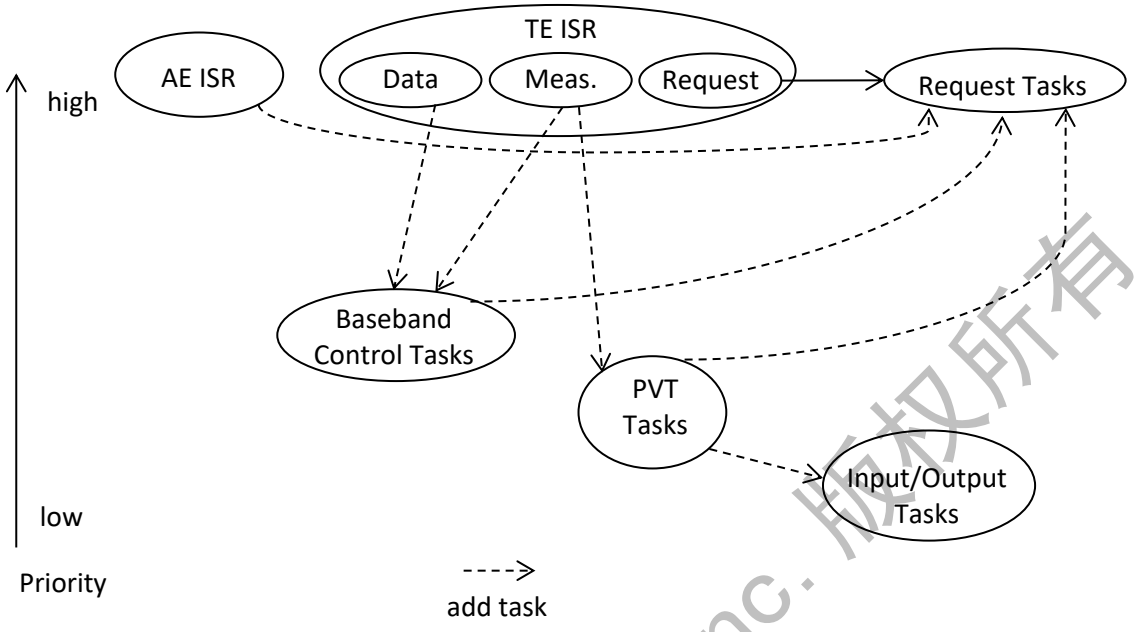
软件根据不同函数的功能组织在不同的目录下，不同目录的不同源代码对应的主要功能如下：

Abstract：硬件抽象层和操作系统平台抽象层
Baseband：基带控制，源文件和头文件分别存放在 src 和 inc 目录下
    AEManager.c：捕获引擎控制相关函数
    ChannelManager.c：通道控制相关函数
    FirmwarePortal.c：firmware 调用入口和与系统平台接口的函数
    TaskQueue.c：任务队列控制及调用函数
    TEManager.c：跟踪引擎控制相关函数
Common：Firmware 公用的类型及宏定义
PVT：位置解算，源文件和头文件分别存放在 src 和 inc 目录下

1.2 软件组织调用架构

软件在初始化完成后，主要的运行都是根据基带中断推动运行的。中断处理程序分别处理捕获引擎中断，跟踪引擎产生的 data ready 中断、measurement 中断和 request 中断。其中捕获引擎的中断与跟踪引擎是异步发生的，跟踪引擎中断后跟踪引擎都会停止运行，等待软件发出恢复运行的命令。硬件中断处理程序会处理需要立即响应的任务，并且根据需求触发不同优先级的任务线程，并在线程中进行非紧急的任务。由于对基带硬件的访问（包括修改寄存器和 State Buffer 的内容）基本上都需要在跟踪引擎停止运行的时候进行，因此除中断任务以外，其他线程的任务完成后如果需要访问基带硬件，都需要设置标志或者将运行的任务加入任务队列并在 Request 中断中进行调用。

除了 request 任务外，其他任务一般分成三个不同优先级的层级。而 request 任务通常是为了处理需要与硬件同步的访问跟踪引擎硬件的操作，直接在 request 中断中进行调用。中断和任务的相互关系如下图所示：



基带控制任务主要处理与基带控制相关的任务，比如说比特同步、导航电文码流解码、通道失锁和重捕判断等。环路滤波可以放在中断里面，也可以放在基带控制任务线程中。PVT 任务主要处理导航电文帧同步和帧解析、原始观测量计算以及位置解算，同时非紧急的基带相关任务（如 PPS 控制）也可以放在 PVT 任务里面。输入输出任务主要处理输出内容的打印，输入数据和输入命令的解析等等对于延时不敏感的任务处理。

由于假定应用处理器不支持浮点协处理器，因此原则上中断处理程序和基带控制任务都采用定点计算以缩短处理时间。这样的设计对于包含浮点协处理器的 CPU 也适用，因为有些 RTOS 中的任务调度器（Scheduler）对于中断处理不对浮点协处理器的寄存器提供额外的保护，因此通常会避免在中断处理程序中进行浮点运算。

1.3 任务队列

一个优先级中不同的任务通常彼此之间没有优先级的区别，另外对于需要执行的任务的请求来源通常是异步的（如中断和操作系统线程之间），同时需要执行的任务可能性有很多，因此将具有同样优先等级的任务组织成任务队列，由一个线程根据需要依次调用。每一个任务由以下数据结构进行存储：

```
typedef struct tag_TASK_ITEM
{
    TaskFunction CallbackFunction;
    void *ParamAddr;  // address of parameter in buffer (align to DWORD)
    int ParamSize;    // size of parameter
    struct tag_TASK_ITEM *pNextItem; // pointer to next item in link list
} TASK_ITEM, *PTASK_ITEM;
```

其中包含了任务函数指针，调用参数存储位置的指针，调用参数 Buffer 的大小。由于任务以链表形成进行组织，因此还包含了指向链表中下一个任务的指针。

上述的多项任务由任务队列控制结构体进行管理。对于上述任务数据，任务队列维护两个链表，分别是空闲链表和待执行任务链表，同时为了方便将新任务添加到待执行任务链表的末尾，还有一个指向待执行任务链表末尾的指针。

由于任务的运行与任务的添加函数是异步运行的，因此任务添加函数在添加完成任务后，不负责继续维持添加的任务运行所需要的数据，取而代之的是将数据拷贝到一个循环队列中去。

任务添加函数首先从空闲列表取到一个空闲的任务项，将内容进行相应的赋值后添加到待执行任务列表的末尾。同时，对于任务所需要的数据会在一个循环 Buffer 中寻找下一块连续的可以放下任务数据的空间并且将内容拷贝进去。

任务释放函数会将待执行链表中的第一个任务项放回到空闲链表中，同时释放任务数据回到循环 Buffer 中。之所以通过一个读指针和一个写指针维护一个循环队列，是因为新任务有可能在待执行任务在执行过程中间进行添加，此时已执行完的任务释放的任务数据空间就可以重复使用。

任务执行函数会从待执行链表中依次取出任务项调用执行，并在执行以后将任务项进行释放，直到待执行链表变空为止。相应的任务线程会等待一个信号量（通常由任务添加函数设置），然后调用任务执行函数，并重复上述流程。

由于添加任务时相应的任务参数的数据会拷贝到任务数据空间中，因此一般的参数数据（或数据结构）内容设计的原则是：如果相应的数据可能在后续被任务添加程序修改，或者数据量比较小，那么就拷贝数据，否则就拷贝指针。

2. 信号捕获流程

信号的粗捕获、精捕获、捕获成功的判断以及捕获转跟踪。

3. 信号的跟踪

3.1 信号跟踪状态及状态转换

3.2 比特同步和导航电文解码

考虑到比特同步的判断可能采用比较复杂的方法，同时比特同步判断不属于需要紧急完成的任务，因此没有必要在中断处理程序中完成。每次处理相干累加的中断程序收集 20 个 1 毫秒的相关数据后将比特同步任务发送给基带任务队列。同时发送的还有和数据同步的时间标签。比特同步任务完成后，根据时间标签返回相对于时间标签的比特同步位置。后续的中断处理程序可以在忽略比特同步任务延迟的情况下正常调整以比特同步位置为起始的相干积分周期。

比特同步暂时采用最简单的策略：判断 20 个毫秒间隔上的翻转次数，当某一个位置的翻转次数超过 5 次，且在全部翻转中的占比超过一半，即判定比特同步成功。

3.3 鉴相、鉴频和延迟鉴别器

鉴相采用的是传统的反正切的鉴相器。反正切运算采用定点的 CORDIC 算法。对于 GPS L1C/A 信号，采用二象限的反正切，对于导频信号，采用四象限的反正切。

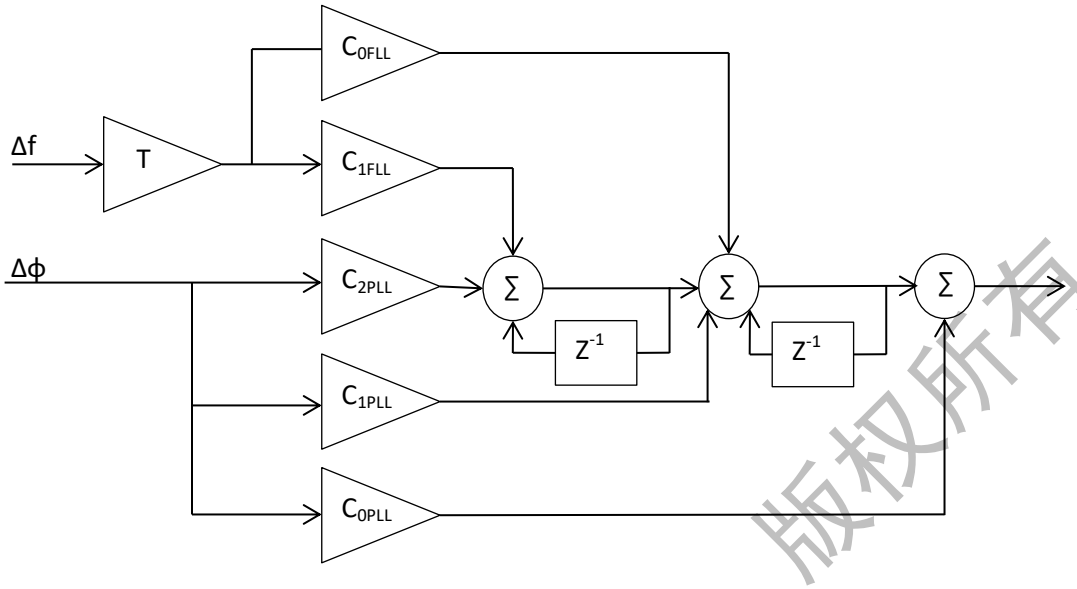
CORDIC 算法采用 14 次旋转运算就以及足够精确。反正切的输出为-32768~32767 对应−π~π的范围，这样当需要对角度进行加减的时候，整周的部分会自动溢出到高 16 比特，或者说低 16 比特依然是对应−π~π的范围。

鉴频有两种方式：一种是通过点叉积的方式，即将两次相干积分的结果分别求点积和叉积然后再求二者的反正切。点叉积适合比较强的信号或者初始频差比较大的情况。另外一种是 FFT 的方式，即将若干个相干积分结果做 FFT 运算（如相干积分个数少于 FFT 点数则在后面补 0），然后对峰值的频格（Frequency Bin）和其左右两个频格上的能量（或幅度）进行插值运算得到相应的频差。插值运算采用的公式是$\frac{L-R}{2P-L-R}$，或者为了使鉴频曲线变得更加平直，采用$\tan^{-1}\frac{L-R}{2P-L-R}$。采用上述公式的好处是，鉴频结果相对于参与 FFT 的相干积分的个数不敏感，并且当频偏为半个频格（即 P=L 或 P=R）的时候，鉴频的结果就是±1，不需要额外的归一化。

延迟鉴别器采用类似的方法，公式采用$\frac{E-L}{2P-E-L}$，同样具有鉴频结果相对于相关器间隔和相关峰形状不敏感，只需要根据相关器间隔进行简单归一化等优点。

3.4 跟踪环路和环路滤波系数选择

下图显示了一个典型的二阶 FLL 辅助三阶 PLL 的环路滤波器。



上述环路的参数，是根据给定的噪声带宽 Bn 以及环路更新间隔计算的。计算的公式如下：

First order:

$\omega_n = Bn/0.25 \quad c_0 = \omega_n$

Second order:

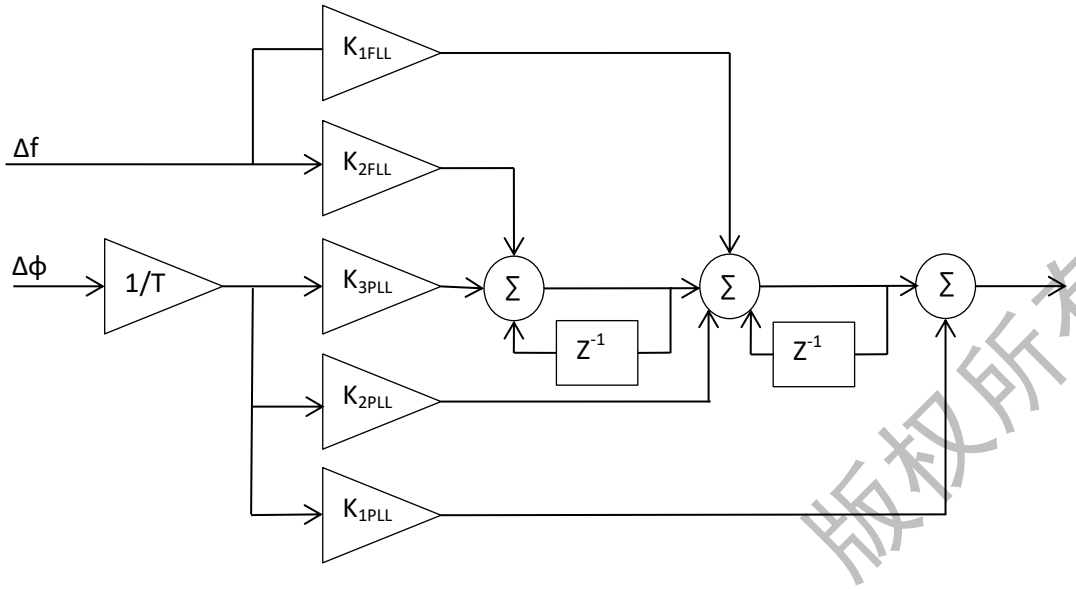
$\omega_n = Bn/0.53 \quad c_0 = 1.414\omega_n \quad c_1 = \omega_n^2 T$

Third order:

$\omega_n = Bn/0.7845 \; c_0 = 2.4\omega_n \quad c_1 = 1.1\omega_n^2 T \quad c_2 = \omega_n^3 T^2$

上式中的 T 是环路更新间隔。

为了方便环路参数的计算，将环路改成以下的结构和计算方法：



由于输入的频率差和相位差都除一个 T 的比例因子，因此该比例因子合并到后续的环路参数中。所有，有如下的对应关系：

First order:

$\omega_n = Bn/0.25 \quad k_1 = c_0T = \frac{1}{0.25}B_nT$

Second order:

$\omega_n = Bn/0.53 \quad k_1 = c_0T = \frac{1.414}{0.53}B_nT \quad k_2 = c_1T = \frac{1}{0.53^2}(B_nT)^2$

Third order:

$\omega_n = Bn/0.7845 \quad k_1 = c_0T = \frac{2.4}{0.7845}B_nT \quad k_2 = c_1T = \frac{1.1}{0.7845^2}(B_nT)^2 \quad k_3 = c_2T = \frac{1}{0.7845^3}(B_nT)^3$

可以看出，上述参数都是和噪声带宽 Bn 以及环路更新时间的乘积B_nT有关。这里B_nT是一个无量纲的量。

上述的参数计算是根据连续信号 FLL/PLL 的系统响应进行计算的，但是在离散系统中，由于 s 平面的极点映射到的 Z 平面的极点形成的系统响应不一致，所以上述计算得到的参数用于离散系统 PLL 的时候会造成滤波响应与期望不完全一致，甚至会造成系统不稳定。

一个解决的方法是对于上述的延时累加改成双线性累加。另外一个解决方法是参照论文《Controlled-Root Formulation for Digital Phase-Locked Loops》（by S. A. Stephens and J. B. Thomas）中提供的参数进行计算。下图给出了论文中通过计算得到的参数表格，可以根据B_nT的值查表之间得到相应的环路参数。

TABLE VIII
Loop-Filter Constants for DPLL With Rate-Only Feedback and Standard-Underdamped Response

No computation delay

	1st order	2nd order		3rd order			4th order			
$B_L T$	K_1	K_1	K_2	K_1	K_2	K_3	K_1	K_2	K_3	K_4
0.005	0.01976	0.01312	8.661e-05	0.01283	7.365e-05	1.590e-07	0.01165	6.831e-05	2.006e-07	2.951e-10
0.010	0.03918	0.02589	3.396e-04	0.02530	2.886e-04	1.242e-06	0.02299	2.673e-04	1.561e-06	4.572e-09
0.015	0.05822	0.03831	7.482e-04	0.03742	6.356e-04	4.084e-06	0.03399	5.879e-04	5.117e-06	2.237e-08
0.020	0.07689	0.05039	0.001302	0.04918	0.001106	9.429e-06	0.04468	0.001021	1.177e-05	6.827e-08
0.025	0.09520	0.06214	0.001992	0.06062	0.001691	1.794e-05	0.05506	0.001560	2.233e-05	1.610e-07
0.030	0.1132	0.07358	0.002810	0.07172	0.002383	3.020e-05	0.06516	0.002195	3.747e-05	3.225e-07
0.035	0.1308	0.08472	0.003746	0.08252	0.003175	4.674e-05	0.07496	0.002921	5.730e-05	5.774e-07
0.040	0.1481	0.09556	0.004793	0.09302	0.004060	6.800e-05	0.08450	0.003731	8.382e-05	9.522e-07
0.045	0.1651	0.1061	0.005944	0.1032	0.005032	9.438e-05	0.09377	0.004619	1.160e-04	1.474e-06
0.050	0.1818	0.1164	0.007191	0.1132	0.006085	1.262e-04	0.1028	0.005578	1.546e-04	2.173e-06
0.060	0.2143	0.1362	0.009950	0.1322	0.008410	2.075e-04	0.1201	0.007691	2.526e-04	4.218e-06
0.070	0.2456	0.1551	0.01302	0.1503	0.01099	3.137e-04	0.1365	0.01003	3.796e-04	7.321e-06
0.080	0.2758	0.1731	0.01637	0.1674	0.01380	4.460e-04	0.1521	0.01256	5.367e-04	1.171e-05
0.090	0.3051	0.1902	0.01995	0.1837	0.01679	6.055e-04	0.1669	0.01526	7.244e-04	1.761e-05
0.100	0.3333	0.2066	0.02372	0.1991	0.01995	7.924e-04	0.1809	0.01809	9.428e-04	2.523e-05
0.150		0.2788	0.04471	0.2657	0.03734	0.002135	0.2417	0.03356	0.002476	9.483e-05
0.200		0.3387	0.06740	0.3183	0.05595	0.004103	0.2899	0.04990	0.004650	2.270e-04
0.250		0.3912	0.09013	0.3606	0.07452	0.006583	0.3288	0.06604	0.007307	4.273e-04
0.300		0.4464	0.1108	0.3951	0.09238	0.009451	0.3606	0.08142	0.01030	6.937e-04
0.350				0.4241	0.1093	0.01259	0.3870	0.09580	0.01351	0.001020
0.400				0.4499	0.1253	0.01584	0.4093	0.1091	0.01683	0.001397
0.450							0.4282	0.1213	0.02021	0.001815
0.500							0.4446	0.1325	0.02359	0.002264
0.600							0.4726	0.1523	0.03034	0.003197

可以看到，随着B_nT的增加，环路变得不稳定，需要更高的环路滤波器的阶数。

3.5 环路滤波器更新算法

根据环路滤波器的计算公式，对于三阶 PLL，输入的相位差$\Delta\phi$和输出的控制频率 f 之间的关系是$f = f_0 + k_1\frac{\Delta\phi}{T} + k_2\sum\frac{\Delta\phi}{T} + k_3\sum\sum\frac{\Delta\phi}{T}$

其中f_0为 PLL 运行开始时的初始频率设置。如果阶数小于 3 阶，则更高次的系数为 0。

对于 DLL，上述公式依旧成立，只是$\Delta\phi$变成以码片（或 1/2 码片）表示的码相位差，而 f 则表示本地码频率（或两倍码频率）。

由此可以得到，对于 PLL，第 n 次更新以后的频率为$f_n = f_0 + k_1\frac{\Delta\phi}{T} + k_2\sum\frac{\Delta\phi}{T} + k_3\sum\sum\frac{\Delta\phi}{T}$

设$f_n^{'} = f_0 + k_2\sum\frac{\Delta\phi}{T} + k_3\sum\sum\frac{\Delta\phi}{T}$

则可以得到$f_n' = f_{n-1}' + k_2\frac{\Delta\phi}{T} + k_3\sum\frac{\Delta\phi}{T}$，控制频率为$f_n = f_n' + k_1\frac{\Delta\phi}{T}$

对于二阶 FLL，上述公式则变为$f = f_0 + k_1\sum\Delta f + k_2\sum\sum\Delta f$

同样，第 n 次更新的频率和上次更新的频率之间关系为：$f_n = f_{n-1} + k_1\Delta f + k_2\sum\Delta f$

这样，控制频率只需要根据上一次更新的值进行迭代更新就可以了。

实际进行更新的时候，使用的增益系数$k = k_L \cdot k_C / k_D$，其中k_L、k_C和k_D分别为环路滤波系数，控制频率到 NCO 频率控制字的转换系数以及鉴别器增益。在对环路滤波进行定点化的时候，会将上述增益一并考虑进去。

3.6 环路滤波器的定点化

为了以定点计算环路滤波，需要正确选择环路参数的比例因子，以在不至于溢出的情况下提供足够的量化精度。为此，需要计算上述参数的选择范围。

下面对 FLL/PLL/DLL 的$B_n T$分别进行分析。

一般来说，对于比较强的信号或者在 pull-in 阶段，会采用比较短的环路更新周期（积分时间），同时采用比较大的环路带宽，而在进入稳定跟踪或者跟踪弱信号时，环路更新周期（积分时间）比较长，但是环路带宽会比较小。

对于 FLL，下面给出几个典型的积分时间

Coh	FFT	Non-coh	Bn (Hz)	T (ms)	BnT	Mode
1	2	5	25	10	0.25	Wideband pull-in (cross-dot)
1	5	8	8	40	0.32	Narrowband pull-in
4	5	16	0.8	320	0.256	weak data signal tracking
10	5	6	0.8	300	0.24	weak pilot signal tracking
1	2	1	5	2	0.01	Minimum possible BnT

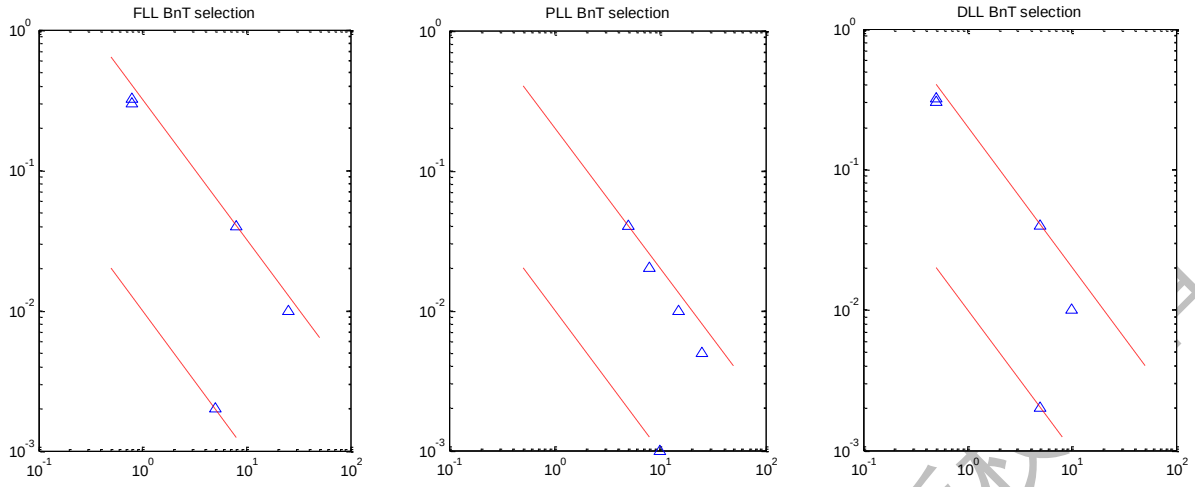
对于 PLL，下面给出几个典型的积分时间

Coh	FFT	Non-coh	Bn (Hz)	T (ms)	BnT	Mode
5	-	-	25	1	0.125	Wideband pull-in
10	-	-	15	5	0.075	Narrowband pull-in
20	-	-	8	20	0.16	weak data signal tracking
40	-	-	5	40	0.2	weak pilot signal tracking
1	-	-	10	1	0.01	Minimum possible BnT

对于 DLL，下面给出几个典型的积分时间

Coh	FFT	Non-coh	Bn (Hz)	T (ms)	BnT	Mode
1	2	5	10	10	0.1	Wideband pull-in (cross-dot)
1	5	8	5	40	0.2	Narrowband pull-in
4	5	16	0.5	320	0.16	weak data signal tracking
10	5	6	0.5	300	0.15	weak pilot signal tracking
1	2	1	5	2	0.01	Minimum possible BnT

显示在下图中：



上图中三个对应下限的红线都是 0.01，FLL/PLL/DLL 对应上限的红线分别是 0.32、0.2 和 0.2。

根据 BnT 的上限和下限，可以从表中查到环路系数的范围。

载波和码的频率控制字增益由采样率确定，如果采样率为f_s，则相应的频率控制字增益$k_C = \frac{2^{32}}{f_s}$。采样率最小为 4 倍码速率，因此根据采样率的不同k_C的最大值大约为 1050。

鉴相器是通过 CORDIC 算法计算反正切得到的角度，由于需要乘以 1/T 折算到 Hz 为单位，因此角度以周角为单位，如间隔 T=20ms 时角度差为 1/10 周，则$\Delta f = \frac{\Delta\phi}{T} = \frac{0.1}{0.02} = 5Hz$。由于 CORDIC 的输出为 2^{16} 表示一个周角，则鉴相器增益$k_D = 2^{16}T_c$，也就是鉴相器输出$P = 2^{16}T_c \cdot \Delta f$。这里$T_c$是相干积分时间。

鉴频器根据鉴频的方式不同，计算方式也不同。当进行点叉积鉴别的时候，可以得到$\Delta f = \frac{1}{T_c}\tan^{-1}\frac{S_1 \times S_2}{S_1 \cdot S_2}$。同样，通过 CORDIC 方法输出的角度增益为 2^{16}，因此鉴相器增益$k_D = 2^{16}T_c$，也就是鉴频器输出$F = 2^{16}T_c \cdot \Delta f$。当进行 FFT 鉴频的时候，一般通过 8 点 FFT 进行计算。此时，8 个频格（frequency bin）之间总的频率范围为$\frac{1}{T_c}$，则每一个频格之间的频率差距为$\frac{1}{8T_c}$。如果鉴频器采用的差值鉴频公式为$\tan^{-1}\frac{L-R}{2P-L-R}$，其中 *L* 和 *R* 分别代表峰值左右的两个频格的信号幅度。则频率相对于中心频格最大偏离±1/2 频格的时候，有 P=L 或者 P=R，因此$\frac{L-R}{2P-L-R}$的范围在±1 之间，相应反正切的输出范围在±1/8 周，也就是±8192。因此可以得到一个频格对应的鉴频输出范围为 16384，相应的鉴频器增益为$16384 \times 8T_c = 2^{17}T_c$。为了与其他的鉴频鉴相的增益统一，可以将输出右移一位，同样可以得到$k_D = 2^{16}T_c$。需要注意的是如果最大频点不在 0 频率对应的频格上，则每偏一个频格需要补偿 8192（将反正切结果右移一位后）。

延迟鉴别器采用的鉴别公式为$\frac{E-L}{2P-E-L}$，其中 *E* 和 *L* 分别代表超前和滞后相关器的信号幅度，则当偏离峰值相关器±1/2 相关器间隔范围的时候，上述公式的输出的范围为±1。也就是说当相关器间隔为 1/2 码片，1/4 码片和 1/8 码片的时候，鉴别器增益分别为 2、4、8 倍的半码片。为保

证输出有效数字位数和用整数除法进行鉴别器输出，分子$E-L$在做除法前分别左移 13、12、11 位，使得鉴别器增益变成 2^{14}。如果最大峰值不在峰值相关器（Correlator 4）上，则每偏一个相关器，需要相应补偿 2^{14}、2^{13}、2^{12}。延迟鉴别器输出到环路滤波器的时候，还需要除以环路更新周期，也就是总的积分时间 T。因此延迟鉴别器输出增益$k_D = 2^{14}T$。

当计算得到k_C和k_D的值，并确定k_L的范围以后，就可以计算环路增益系数 k 并确定环路增益的计算方法了。一般来说，大部分 PLL 会采用两阶或是三阶，FLL 会采用两阶，DLL 会采用两阶或者带载波辅助的一阶。

首先计算 FLL/PLL 的环路更新系数。由于$k = k_L \cdot k_C/k_D$，因此带入k_C和k_D后可以得到$k = k_L \cdot \frac{2^{32}}{f_s} \cdot \frac{1}{2^{16}T_c} = k_L \cdot \frac{2^{16}}{f_sT_c}$。这里$f_sT_c$表示的是每毫秒的采样点数目乘以相干累加的毫秒数，范围大约在 4000 到 200000，或者 2^{12}~2^{18}之间。参考从环路参数表中查到的k_L值，可以判断 k_1的范围大约在 2^3到 2^{-4}之间。k_2的范围大约在 2 到 2^{-7}之间。考虑到一般应用场景动态较大时环路带宽通常比较宽，而动态较小时或者积分时间较长，或者不需要 3 阶环路，因此 BnT 在 0.05 以下可以不考虑 3 阶环路。由此得到 k_3的范围大约在 2^{-8}到 2^{-12}之间。由于上述 k 值不能用整数表示，因此需要以一定的比例因子扩大。

首先，鉴相器的输出在-32768 到 32767 之间，可以最大乘以 2^{16}不会溢出，因此 k_1可以有 2^{13}的增益，即$K_1 = 2^{13}k_1$。同样 k_2可以有 2^{15}的增益，即$K_2 = 2^{15}k_2$。由于 k_3需要乘鉴频器输出的累加值，以累加值可能扩大 2^9计算，那么累加值可以乘以 2^7不溢出，也就是可以有 2^{15}的增益，即$K_3 = 2^{15}k_3$。最终得到输出给 NCO 的频率控制字为$W = W_n + (K_1 \cdot P_n) \gg 13$，$W_n = W_{n-1} + (K_2 \cdot P_n + K_3 \cdot ACC_n) \gg 15$，$ACC_n = ACC_{n-1} + P_n$。其中$P_n$为进行第 n 次更新的鉴相器输出，$ACC_n$为前 n 次鉴相器输出的累加值。

以采样率$f_s = 4.113MHz$，$T_c = 20ms$，$B_n = 7.5Hz$的 3 阶环为例，得到 BnT=0.15，$f_sT_c = 82260$。此时可以得到$K_1 = 0.2657 \cdot \frac{2^{16}}{f_sT_c} \cdot 2^{13} = 1734$，$K_2 = 0.03734 \cdot \frac{2^{16}}{f_sT_c} \cdot 2^{15} = 975$，$K_1 = 0.002135 \cdot \frac{2^{16}}{f_sT_c} \cdot 2^{15} = 56$。

Annex A.1 的 Matlab 程序说明了定点频率控制字的计算 PLL 过程。

FLL 鉴频器的输出范围、鉴别器增益、系数范围计算与 PLL 类似，这里不再赘述。计算频率控制字的方法为$W_n = W_{n-1} + (K_1 \cdot P_n + K_2 \cdot ACC_n/4) \gg 13$，$ACC_n = ACC_{n-1} + P_n$。其中$P_n$为进行第 n 次更新的鉴频器输出，$ACC_n$为前 n 次鉴频器输出的累加值。Annex A.2 的 Matlab 程序以 1ms 相干累加长度，20Hz 带宽的点叉积鉴频计算 FLL 定点频率控制字的过程。

对于 DLL，也是用类似的方法计算。首先带入k_C和k_D到公式$k = k_L \cdot k_C/k_D$后可以得到$k = k_L \cdot \frac{2^{32}}{f_s} \cdot \frac{1}{2^{14}T} = k_L \cdot \frac{2^{18}}{f_sT}$。由于$f_sT$的范围大致在 2^{13}到 2^{21}之间，因此可以判断 k_1的范围大约在 2^1到 2^{-7}之间。k_2的范围大约在 2^{-3}到 2^{-10}之间。可以设定 k_1和 k_2的增益都是 2^{15}，即可得到输出给 NCO 的频率控制字为$W = W_0 + (K_1 \cdot D_n + K_2 \cdot ACC_n) \gg 15$，$ACC_n = ACC_{n-1} + D_n$。其中$D_n$为进行第 n 次更新的鉴相器输出，$ACC_n$为前 n 次鉴相器输出的累加值，$W_0$为码 NCO 频率控制字标

称值。Annex A.3 的 Matlab 程序以 5ms 相干累加，8 次非相干累加，2Hz 的滤波器带宽计算 DLL 定点频率控制字的过程。

4. 位置解算

5. 系统和硬件抽象平台接口

在调试和验证 Firmware 的时候，可以在实际的硬件平台上运行，也可以在 PC 仿真环境中运行。为了让 Firmware 在运行和调试的时候不依赖于具体的底层功能，因此会将底层抽象出来，提供统一的调用接口，但是对应不同的平台提供接口函数的不同实现。根据需要抽象的功能不同，分为硬件抽象层和系统抽象层两类。其中硬件抽象层主要是提供硬件逻辑（捕获单元、跟踪单元等、外设等）的抽象，系统抽象层主要是提供系统功能（线程调度、IPC 等）的抽象。

硬件抽象层提供的接口函数的定义在 HWCtrl.h 中，提供了基带逻辑总线读写、AE Buffer 和 TE Buffer 的读写、外设控制等等，而具体的函数实现在与具体的平台相对应的.c 或.cpp 文件中，根据需要运行的平台不同，在工程中添加相应实现的源代码文件。

系统抽象层提供的接口函数定义在 PlatformCtrl.h 中，提供了线程控制、中断控制、IPC 等功能的接口，同样具体的函数实现在与特定平台和 OS 相对应的.c 或.cpp 文件中。

通过抽象层，main()函数只需要以下面简单的方式构造就可以运行起来：

```
#include <stdio.h>
#include <math.h>
#include <string.h>

#include "RegAddress.h"
#include "InitSet.h"
#include "HWCtrl.h"
extern "C" {
#include "FirmwarePortal.h"
}

void main()
{
	SetInputFile("..\\..\\..\\matlab\\signal_src\\sim_signal_L1CA.bin");

	FirmwareInitialize();
	EnableRF();
}
```

函数 SetInputFile()指定仿真对象（在实际硬件上没有实际作用）。

函数 FirmwareInitialize()进行 firmware 的初始化，包括创建相应的线程和信号量，注册中断处理函数以及初始化变量等。

函数 EnableRF()使能射频前端，产生中频采样信号和采样钟从而启动基带硬件并触发相应的中断处理函数。而在仿真平台上会运行底层仿真功能。

Annex A.1 PLL 频率控制字定点计算示例代码

```
% settings
T=0.02; Bn=7.5; BnT=Bn*T;
% coefficients
k1=0.2657; k2=0.03734; k3=0.002135;
% initial Doppler frequency
f0=120;
% first round
dp1=0.1;df1=dp1/T;
acc=df1;acc2=df1;
f1=f0+k1*df1+k2*acc+k3*acc2;
% second round
dp2=0.12;df2=dp2/T;
acc=acc+df2;acc2=acc2+acc;
f2=f0+k1*df2+k2*acc+k3*acc2;
% third round
dp3=0.075;df3=dp3/T;
acc=acc+df3;acc2=acc2+acc;
f3=f0+k1*df3+k2*acc+k3*acc2;
% convert to frequency control word
fs=4.113e6;
w=[f1*2^32/fs f2*2^32/fs f3*2^32/fs];
% fixed point
% coefficients
kk1=1734;kk2=975;kk3=56;
% discriminator output
p1=dp1*65536;p2=dp2*65536;p3=dp3*65536;
% first round
fc0=f0*2^32/fs;
accp=p1;w1=fc0+floor((kk2*p1+kk3*accp)/2^15);
fc1=w1+floor((kk1*p1)/2^13);
% second round
accp=accp+p2;w2=w1+floor((kk2*p2+kk3*accp)/2^15);
fc2=w2+floor((kk1*p2)/2^13);
% third round
accp=accp+p3;w3=w2+floor((kk2*p3+kk3*accp)/2^15);
fc3=w3+floor((kk1*p3)/2^13);
% compare floating point w with fixed point result [fc1 fc2 fc3]
```

Annex A.2 FLL 频率控制字定点计算示例代码

```
% settings
T=0.002; Bn=20; BnT=Bn*T;
T=0.001; % coherent length
% coefficients
k1=0.09556; k2=0.004793;
% initial Doppler frequency
f0=120;
% first round
df1=12;
acc=df1;acc2=df1;
f1=f0+k1*acc+k2*acc2;
% second round
```

```
df2=14;
acc=acc+df2;acc2=acc2+acc;
f2=f0+k1*acc+k2*acc2;
% third round
df3=9;
acc=acc+df3;acc2=acc2+acc;
f3=f0+k1*acc+k2*acc2;
% convert to frequency control word
fs=4.113e6;
w=[f1*2^32/fs f2*2^32/fs f3*2^32/fs];
% fixed point
% coefficients
kk1=12473;kk2=2503;
% discriminator output
f1=df1*65536*T;f2=df2*65536*T;f3=df3*65536*T;
fc0=f0*2^32/fs;
% first round
accf=f1;fc1=fc0+floor((kk1*f1+kk2*accf/4)/2^13);
% second round
accf=accf+f2;fc2=fc1+floor((kk1*f2+kk2*accf/4)/2^13);
% third round
accf=accf+f3;fc3=fc2+floor((kk1*f3+kk2*accf/4)/2^13);
% compare floating point w with fixed point result [fc1 fc2 fc3]
```

Annex A.3 DLL 频率控制字定点计算示例代码

```
% settings
T=0.04; Bn=2; BnT=Bn*T;
% coefficients
k1=0.1731; k2=0.01637;
% initial Doppler frequency
c0=2.046e6;
% first round
d1=0.2; df1=d1/T;
acc=df1;
w1=c0+k1*df1+k2*acc;
% second round
d2=0.15; df2=d2/T;
acc=acc+df2;
w2=c0+k1*df2+k2*acc;
% third round
d3=0.12; df3=d3/T;
acc=acc+df3;
w3=c0+k1*df3+k2*acc;
% convert to frequency control word
fs=4.113e6;
w=[w1*2^32/fs w2*2^32/fs w3*2^32/fs];
% fixed point
% coefficients
kk1=9038;kk2=855;
% discriminator output
di1=d1*2^14;di2=d2*2^14;di3=d3*2^14;
fc0=c0*2^32/fs;
% first round
```

```
accd=di1;fc1=fc0+floor((kk1*di1+kk2*accd)/2^15);
% second round
accd=accd+di2;fc2=fc0+floor((kk1*di2+kk2*accd)/2^15);
% third round
accd=accd+di3;fc3=fc0+floor((kk1*di3+kk2*accd)/2^15);
% compare floating point w with fixed point result [fc1 fc2 fc3]
```