Greta Oto 卫星导航接收机设计手册

Jun Mo

Globsky Technology Inc.

2021/5/20

前言

正如封面图片所展示的，Greta Oto 是一种透明翅膀的蝴蝶。之所以给这个项目起了这个名字，是因为以下的渊源。我在近十年前，开始设计一个通用的卫星导航接收机架构，起初这个架构是为了采用尽可能简单的结构实现适用于多种卫星信号的接收机。这个架构我为它起名为 Butterfly，用以形容它的轻巧和灵活。基于 Butterfly 架构，后来又不断地根据新的卫星信号体制进行各种扩展，实现了一些颇为成功的多系统多频率的卫星导航接收机产品。我去年发布的 OPENGNSS 项目，也是基于这样的一个架构。在发布了 OPENGNSS 项目的基带硬件的全部 C 模型和部分 RTL 以后，在后续的维护过程中我发现，因为缺乏具体的产品需求，OPENGNSS 项目只适合作为一个原理性的学习平台和一个可工程化的产品架构的范例。如果需要将其进一步进行工程化的完善，就需要加入更多的细化需求，比如说对于射频前端的限制、对于软件和硬件规模的限制、对于所需支持频点和信号的限制等等。但是如果这样的话，就将减弱 OPENGNSS 作为一个架构型学习平台的通用性。而且，OPENGNSS 涉及到很多的功能比如对宽带射频的支持、对抗干扰的支持、对多系统多频的支持等在具体工程实现中如果不加取舍，就会削弱其实用性。因此，我决定从 OPENGNSS 或者其平台架构 Butterfly 衍生出一个新的专门针对消费级导航型接收机的项目进行更加完善的工程化，进一步细化其需求为接收以 L1 频点为中心的导航型卫星信号，包括 GPS L1C/A、GPS L1C、Galileo E1 和北斗的 B1C 信号。同时考虑到消费级接收机越来越多地支持双频功能，可以在未来扩展支持 L5 频段的 GPS L5、Galileo E5a 和北斗的 B2a 信号。根据进一步细化的需求，我大幅度地修改了 C 模型，并基本上重写了整个的基带 RTL 逻辑。比如，同样是捕获引擎，OPENGNSS 中给出的 RTL 实现实例与 Greta Oto 项目中给出的 RTL 实现就是同一架构下针对不同需求的两种截然不同的实现。这个项目同样也是一个开源项目，未来会陆续发布完整的基带硬件 C 模型、基带硬件 RTL、基带控制固件和 PVT 位置解算软件。如果有条件的话，还会增加可以实际运行的硬件平台。同时，也希望这个平台通过对外授权实现产品化后的反馈，将其不断地完善。由于这个项目是基于 Butterfly 架构，同时通过开源将所有的设计透明化，所以选择了 Greta Oto 这个名字。我也希望通过这个项目，将具有美感的设计展现在大家面前，就如大自然设计出了如此美丽的蝴蝶一般。

版权信息

1. 整体设计

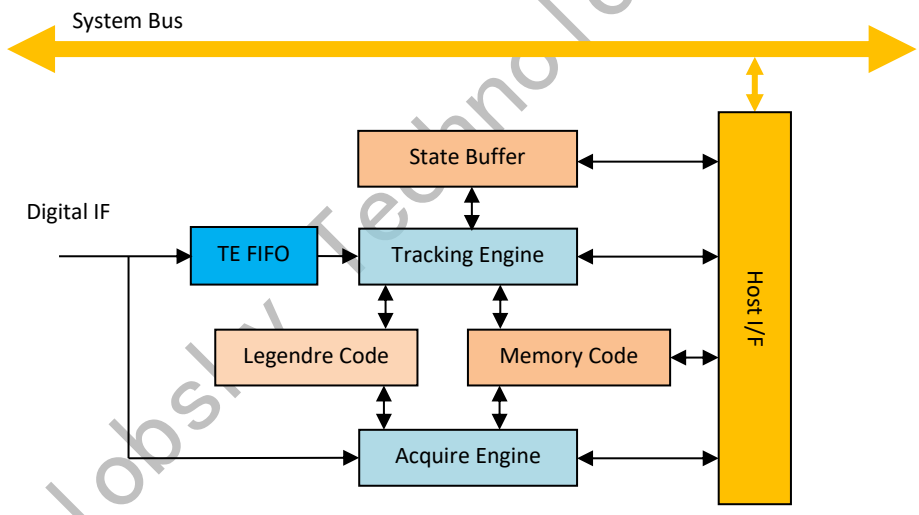
本章从整体设计的角度介绍了 Greta Oto 卫星导航接收机的整体设计架构。

1.1 需求功能描述

在对接收机的整体设计进行介绍之前，首先对接收机的功能需求进行一个简单的描述。接收机主要用于导航型应用，可以接收 L1 频点三系统的四个信号，包括 GPS（包括 SBAS）L1C/A、GPS（包括 QZSS）L1C、Galileo E1 和北斗的 B1C 信号。由于接收机支持的信号为 L1 频点的 BPSK 或 BOC(1,1)调制的信号，信号带宽为 2MHz 或者 4MHz，因此设计采样率为稍大于 4.092MHz 的近零中频复数采样信号。每一个样点为 I/Q 各 4 比特 sign/mag 格式（每个样点 8 比特，从高位到低位的排列为 I 路符号、I 路幅度、Q 路符号、Q 路幅度）。1 比特符号位为 0 表示正，为 1 表示负。3 比特幅度从 000 到 111 分别表示 1、3、5…15。信号前端滤波器带宽为正负 2MHz。实际的射频前端的参数不一定符合上述需求，这时可以通过预处理模块将前端中频采样数据调整到符合上述要求。

1.2 基带硬件总体介绍

下图给出了 Greta Oto 卫星导航接收机基带的整体架构：



基带硬件主要包含捕获引擎和跟踪引擎，两个模块都接收从射频前端来的数字中频信号。其中捕获引擎用于信号的捕获，并通过同步信号与进入到跟踪引擎的信号进行时间同步来进行捕获转跟踪的操作。跟踪引擎采用时分复用的方式，用 4 个物理通道实现多个通道卫星信号的跟踪。在功能上，跟踪引擎支持 BPSK 调制和 BOC(1,1)调制的信号，支持硬件相干累加及 NH 码剥离，支持导频信号或数据信号的跟踪以及在跟踪导频信号的同时完成导航电文的解码。跟踪引擎中的寄存器状态存放在 State Buffer 中，可以由 CPU 进行访问。

2. 跟踪引擎设计

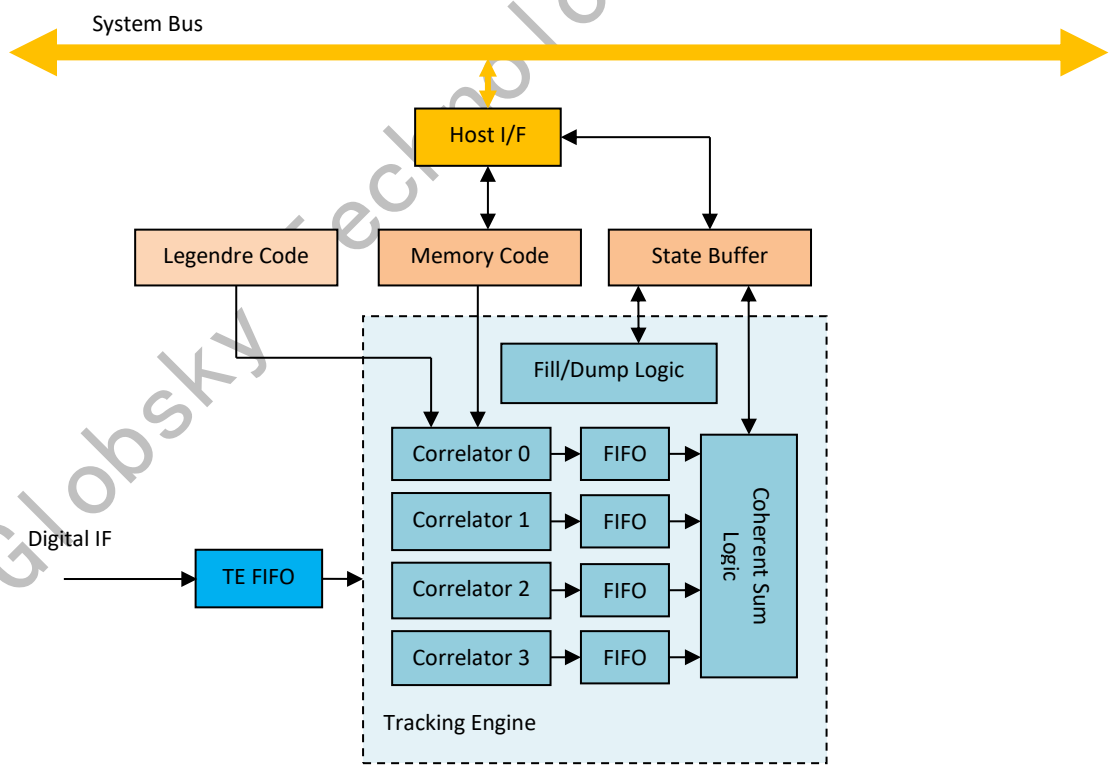
跟踪引擎实现的功能主要是对中频信号剥离多普勒后与不同码相位的本地码进行相关，并对多个毫秒的相关结果进行累加。跟踪引擎的主体是相关器。为了实现相关器的时分复用，需要将中频采样数据存储下来在需要进行相关的时候反复读取。通过相关器的复用，可以节省逻辑。

在跟踪引擎工作的时候，采用软硬件时分的方式，也就是硬件相关器完成全部通道 1ms 的相关运算后，停下来给 CPU 发送中断。CPU 接收到中断后，可以对跟踪引擎的内部寄存器和通道状态进行读写访问，以此进行相关值读取、环路控制、增减通道等操作。操作完成后，CPU 在中断返回前恢复跟踪引擎的运行。

跟踪引擎的相关器每一个时钟周期处理一个中频采样数据，在采样率为 4MHz 的情况下，每秒需要 4M 个时钟周期处理一个通道。如果考虑 3 个系统（GPS、Galileo 和北斗），每系统 16 通道，则总共大约需要 192M 个时钟周期。也就是说，如果跟踪引擎的主频运行在 100MHz 左右时，需要多个物理相关器同时运行才能满足处理能力的需求。这里的设计采用 4 个并行的物理相关器，在主频 100MHz，配置 48 通道时典型的运行时间大约为每一毫秒占用 0.5 毫秒的时间，可以留下足够的空余时间进行软件调度和响应。

2.1 跟踪引擎总体设计

跟踪引擎的总体设计结构如下图所示：



跟踪引擎主要包括以下几个部分：Correlator 0~3 表示 4 个物理相关通道，用于对中频数据进行相关操作，每一个相关通道包含 8 个相关器；将相关结果输出到相干累加逻辑的数据 FIFO；用于进行相干累加的相干累加逻辑；在通道复用时对通道寄存器内容进行填充和读回的 Fill/Dump 逻辑。除此以外，由于进行通道复用，所以每一个通道用于控制进行相关的寄存器值存放在一块 SRAM 中（称为 Status Buffer）。通道在执行相关前将寄存器的数据从 SRAM 填充到寄存器中，在执行完 1 毫秒的相关运算后，再将寄存器的值写回到 SRAM 中。因此，从 CPU 的角度来看，访问 SRAM 的内容就等同于通道寄存器的内容（仅限于在硬件停止运行时）。只要读写 SRAM 的内容，就相当于读写通道寄存器。另外，还有用于存储数字中频的 TE FIFO，以及存储用于生成 Weil 码和 Memory Code 的 ROM/RAM。

跟踪引擎的工作流程是这样的：首先等待 TE FIFO 缓冲的中频数据达到 1ms，然后 TE FIFO 通知相关器启动工作。相关器首先从 SRAM 填充四个通道的寄存器参数，然后从 TE FIFO 读数据执行 1ms 数据的相关，完成后将寄存器的内容写回到 SRAM，然后再填充后面四个通道的寄存器参数，重复读取 TE FIFO 的数据进行相关。重复这一流程直到全部的通道完成相关。然后选择另外一个卫星系统，重复上述流程完成该卫星系统全部通道的相关，依次执行全部卫星系统（如三系统 GPS、BDS 和 GLONASS 则将上述流程执行三轮）。上述流程执行完成后，跟踪引擎进入挂起状态并中断 CPU，CPU 在完成中断处理后恢复跟踪引擎的运行。

在跟踪引擎中，存放各个通道的寄存器值镜像的 SRAM 空间称为 State Buffer。存放输出的相关结果的 SRAM 空间称为 Coherent Buffer。在实际实现中，State Buffer 和 Coherent Buffer 占用同一块 SRAM，具体的地址分配方式可以参见地址映射的章节。

下面分成子模块具体介绍跟踪引擎的实现。

2.2 TE FIFO

TE FIFO 用于存储输入的中频样点，并且在累积到一定数量（通常是 1 毫秒，称为一个 block）后，通知相关器开始工作。TE FIFO 的基本原理是维护一个读指针和一个写指针。当预处理器输出新的中频样点时，采样数据写入 TE FIFO 的 SRAM，并且写指针加 1。当从 TE FIFO 读取数据的时候，读指针加 1。由于需要进行溢出保护，所以 TE FIFO 还会维护一个样点个数计数器。

由于跟踪引擎中的相关器采用时分复用的方式，因此运行的时序是受到软件和硬件调度影响的。整个跟踪引擎中唯一均匀稳定进行计数的就是 TE FIFO 的写指针。因此写指针也兼有在接收机维护本地时间时进行时钟计数的功能，其以标称的中频频率为时钟源对接收机时间进行维护。将写指针与其他的事件进行同步的操作是通过写地址锁存来实现的（详见写地址锁存一节）。

根据通道时分复用需求，TE FIFO 具有以下设计功能和特性：

2.2.1 Reset/Enable/Clear

Reset 信号将 FIFO 的全部寄存器恢复到初始状态，即支持异步 Reset（在接收机硬件初始化时），也支持同步 Reset（由软件控制的 Reset）。Enable 用于控制 TE FIFO 是否工作。当 Enable

无效时，TE FIFO 没有 read/write 的操作，也不会对输入中频数据进行计数。TE FIFO 的 Enable 状态除了受软件控制以外，还可以用于同步。

TE FIFO 的每个通道还可以用 Clear 信号进行清除。当 Clear 有效时，除控制寄存器外全部的寄存器恢复到初始状态。

2.2.2 Dummy Write

Dummy Write 主要用于辅助跟踪引擎实现 trickle power 功能。当 Dummy Write 有效时，中频数据输入到 TE FIFO 时只改变写地址寄存器和数据个数寄存器的值，实际的中频数据不写入 TE FIFO 的 SRAM。在采用 trickle power 跟踪时，为了节省功耗，接收机可以间断性地关断 RF，仅保留采样钟。此时采样钟作为计时的时钟由 TE FIFO 中的写地址寄存器维护本地时间，但中频采样数据是无效的，因此也不需要写入 SRAM，以此节省功耗。

2.2.3 溢出及门限保护

TE FIFO 的容量受 TE FIFO 的 SRAM 的容量限制。当写入的数据超过 TE FIFO 的容量时，就会发生溢出，并置位溢出标志位。溢出标志位需要通过复位 FIFO 或向溢出标志寄存器的相应位写 1 来清除。在 TE FIFO 中，存满 1 毫秒的数据后启动相关器。在相关器运行的过程中，需要反复读取同一毫秒的数据。此时，下一毫秒的数据会存入 TE FIFO。由于平均来说，相关器应该在 1 毫秒内完成所有通道的相关运算，所以 TE FIFO 的容量应该不少于 2 毫秒。考虑到相关器完成相关后，捕获引擎会停止并等待 CPU 控制再次启动，因此考虑到中断响应时间的不确定性，通常可以再加一些容量作为缓冲保护。比如，对于 4MHz，I/Q 共 8 比特的采样来说，10kB 容量的 SRAM 可以存放大约 2.5 毫秒的中频数据。

FIFO 溢出会造成新的采样数据覆盖可能正在使用的旧的采样数据，因此相关的结果会因此产生错误。TE FIFO 在维护的样点个数寄存器的计数范围比 FIFO 的长度要大出许多，所以 FIFO 溢出只会造成数据丢失，不会造成样点计数错误。由于卫星导航信号采用扩频通信的调制方式，所以少数数据样点的丢失不会对相关结果产生太大的影响，也保护了在 FIFO 溢出情况下的跟踪稳定性。

门限保护则是判断 TE FIFO 内的数据个数是否大于某一预制的门限。当数据个数大于门限时置位，小于门限时清除，该位是只读位。门限保护位可以在中断处理中实现灵活的响应。当中断处理的某些操作比较费时间，但又不是非常重要时，可以通过读取门限保护位判断是否还有足够的时间执行相应的操作。如果有时间，则执行，否则将相应的操作延后，并且立即恢复跟踪引擎的运行并进行中断返回。

2.2.4 Rewind/Skip

根据跟踪引擎的工作流程，由于所有通道都处理同一段中频数据，因此在通道时分复用的情况下，TE FIFO 需要多次输出同一个 block 的中频数据，直到全部的通道处理完成。然后回输出下一个 block 的数据。

在具体实现中，是这样处理的：TE 在输出样点的同时，读指针向前计数。输出完一个 block 的数据后，如果还有更多的卫星通道需要处理，则执行 Rewind 操作，读指针返回原先的位置，否则执行 Skip 操作，即读指针保持当前位置，并且 data count 计数器一次性减掉一个 block 的数据个数。

2.2.5 写地址锁存

在卫星导航接收机工作的时候，本地时的维护是非常重要的功能。而本地时追根溯源是通过对采样时钟的计数维护的。在基带实现中，采样时钟计数就反映在 TE FIFO 的写指针寄存器上。因此在特定时刻锁存写地址就成为了时间系统维护的重要功能。

为了与其他模块的时间进行同步，写指针可以由 4 个不同的事件进行锁存，分别是 CPU 发送锁存指令，外部事件（Event Mark），PPS 脉冲沿以及 AE 同步信号（通常是 AE Buffer 写入第一个样点时，用于捕获转跟踪的计算）。每一个事件都会将 FIFO 写指针锁存在各自的写指针锁存寄存器中。每一个写地址锁存寄存器会锁存三部分内容：写地址循环次数计数、写地址当前计数和写地址间隔计数，三者分别占用 12 比特、14 比特和 6 比特。

在当前写地址计数到 FIFO 长度后，就会循环回零重新开始，所以为了计数更长的时间，写地址每次循环回 0 的时候，循环次数就会加一。这样按照 FIFO 长度为 2.5 毫秒计算，加上循环计数总的计数时长就可以达到大约 10 秒。

由于一般采样钟的频率会比基带处理器时钟慢很多（如 100MHz 主频和 4MHz 采样的倍数关系为 25 倍），因此为了提高分辨率，锁存写指针时还会同时锁存写地址间隔计数器的值。写地址间隔计数器在每一个系统时钟增加 1，并且在每次写指针加 1 时清 0，用于提高锁存时刻的时间精度。这样，在一些需要更高精度时间控制的情况下（如 PPS 同步，捕获转跟踪）就可以获得系统时钟分辨率下的本地时间。

2.2.6 Block 调整

通常情况下，TE FIFO 每次输出一个 block 的样点，每个跟踪通道对应的码片寄存器，码相位寄存器和载波相位寄存器的值对应于最后一个样点的码相位和载波相位，也就是用于计算观测时刻的原始观测量的数据。因此，观测时刻对应了每一个 block 的最后一个样点。

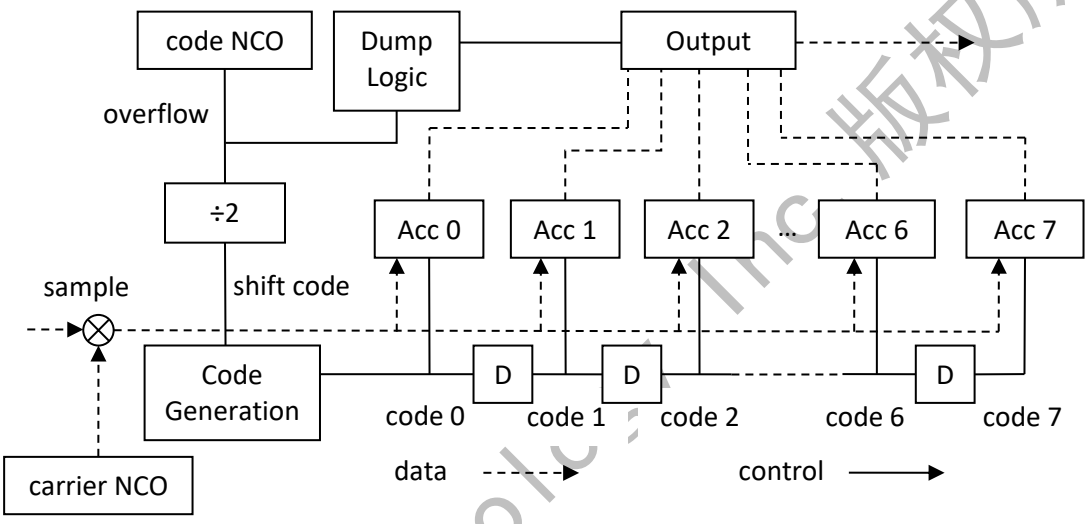
一般来说 Block size 的取值是 1 毫秒的标称样点个数，不需要进行调整。如果钟差过大，可以通过软件调整一整个毫秒来拉近观测时刻与整秒的时间差。但是对于某些特殊应用，需要观测时刻与整秒尽量接近。此时，可以通过调整 Block size 来进行观测时刻的精细调整。

Block size 改变的大小是以样点为单位，范围是-128~127。改变只有一次有效，由软件写入相应的寄存器，而此寄存器在完成了一轮 block 的输出后，由硬件自动清零。

2.3 通道相关器

通道相关器的主要功能是将 TE FIFO 中的中频数据，剥离多普勒以后，与不同延迟的本地码做相关，并在每一个相关周期结束时将相关累加的结果输出。由于不同延迟的本地码的码周期结束的时间不同，所以相关通道中不同的相关器的输出为依次输出。相关周期可以和码周期是一致的，也可以短于一个码周期。比如 Galileo 的情况下，一个码周期为 4092 个码，而相关周期可以设为 1023 个码，也就是 1 毫秒，L1C 和 B1C 的情况也是类似的。但是无论周期长度是否一致，相关周期的边沿都应该对齐码周期的边沿。

相关器的设计框图如下图所示：



输入样点进入到相关器的第一步是经过一个载波 NCO 进行多普勒剥离。载波 NCO 的核心是一个 32bit 的累加器，每次累加的值为频率控制字，累加器的值为当前的载波相位。累加器的高 6 比特用于一个 64 个索引的 sin/cos 的查找表，sin/cos 值的增益为 12。查找表的输出与输入的中频信号相乘后产生零多普勒的信号后与本地 PRN 码进行相关。

本地 PRN 码通过码 NCO 控制。码 NCO 的输入是采样时钟，输出是 2 倍码速率的时钟（通过溢出逻辑信号 overflow 表示）。2 倍码速率的时钟控制一个存放本地码的移位寄存器进行移位，因此相邻的移位寄存器之间的码依次落后 1/2 码间隔。同时 overflow 信号过 2 分频后控制码生成器产生下一个 PRN 码。

相关器框图中的 Acc 表示累加器，每一个累加器对应一个相关器。累加器将输入的数据与对应的本地码进行相关后再累加，累加的结果中 I/Q 各 16 比特。当累加到一个相关周期结束时，累加器的值输出并且清零准备下一个相关周期的累加。需要注意的是，由于相关周期与相关器各自的本地码同步，所以各个相关器的累加结果并不是同时输出，而是依次输出的。

2.3.1 码滑动

一般情况下，进行码跟踪的时候，是通过调整码频率来锁定相关峰位于指定的 P 相关器的位置（比如说在 8 个相关器的情况下，相关器 4 为峰值相关器）。但是在捕获转跟踪以后，或者进行相关器捕获时需要滑动本地码与信号样点之间的延时关系，因此在跟踪通道中设计了 Correlator Jump 的逻辑来对本地码进行滑动。本地码滑动的的长度由一个 8bit 数控制，为二进制补码。0 表示不滑动。如果 Jump Count 为正，在开始相关运算对 FIFO 的数据进行累加前，先强制本地码进行滑动（即强制置 overflow 为 1）Jump Count 次，由 overflow 驱动的所有逻辑正常工作。此时相关峰的位置向后（也就是相关器索引号增加的方向）滑动 Jump Count 个数目。如果 Jump Count 为负，则在相关器正常工作过程中，强制出现的前-jump count 个 overflow 无效，即码 NCO 溢出的前若干次 overflow 不置 1，并增加 Jump Count 直到 0 为止。这样的方式可以将本地码相对于输入的信号向前滑动所需的相关器个数的间距。

2.3.2 窄相关

相关器的实现是用不同延迟的本地码和输入的采样数据进行相关，不同相关器间本地码的延迟反映了相关器的间隔。按照每次码 NCO 溢出一次进行一次移位的方法，由于 overflow 信号的频率是两倍码速率，这就意味着相关器间隔为 1/2 码片。为了达到更精确的跟踪效果和更好的抗多径效果，往往需要做窄相关。窄相关设计就是为了这个功能实现的。

当配置码 NCO 溢出速率为两倍码速率的时候，码 NCO 的频率控制字为$F_c = \frac{2f_c}{f_s}2^{32}$。这里，$f_s$为采样率，$f_c$为码频率（对 L1 频点的所有信号都是 1.023MHz）。

在上述本地码发生器的实现中，可以等效地把本地的相关器 n（n=0~7）的码 NCO 的相位表示为：

$$\Phi_n = \frac{N_n}{2^{32}} = [NCO - d_n \cdot 2^{32}]/2^{32}$$

其中 NCO 是本地 NCO 的计数，d_n是相关器 n 与峰值相关器之间的间隔。当$d_n = 1$时，意味着该相关器需要多溢出一次，才能达到与 p 相关器相同的码相位，也就是滞后 1/2 码片。同样的$d_n = -1$时对应的就是超前 1/2 码片的相关器。对于普通 1/2 码片的相关器设计，d_n是整数值。

如果在上述公式中，取$d_n = 1/2$。此时，Φ_n表示的就是滞后一半的相关器间隔所对应的码相位，而$N_n = NCO - 1/2 \cdot 2^{32}$的取值就反映了$\Phi_n$码相位所对应的码序列的选择。如果$N_n \geq 0$，表示当前码相位对应的码取值与峰值相关器码序列中对应的码相同，而如果$N_n < 0$，表示当前码相位对应的 NCO 少溢出一次，因此对应的码取值与滞后 1/2 码片的码序列中对应的码相同。

进一步简化后，可以得到滞后 1/4 码片的码取值的规则，也就是当$NCO \geq 2^{31}$时，取峰值相关器对应的码，当$NCO < 2^{31}$时，取滞后相关器对应的码。

同样的，取$d_n = -1/2$时，可以得到如果$NCO + 2^{31} < 2^{32}$，也就是$NCO < 2^{31}$时，表示当前码相位对应的码取值与峰值相关器码序列中对应的码相同，而如果$NCO + 2^{31} \geq 2^{32}$，也就是

NCO ≥ 2^{31}时，表示对应 NCO 相位相对峰值相关器相位出现了一次额外的溢出，因此对应的码取值与超前 1/2 码片的码序列中对应的码相同。

由于码 NCO 的取值为 0~2^{32}-1，所以上述的判断可以根据码 NCO 的最高位来决定：

码序列	NCO 最高位	码取值
1/4 超前码	0	当前码
	1	超前码
1/4 滞后码	1	当前码
	0	滞后码

同样的，可以取 n-p=±1/4，这时候，需要判断码 NCO 的最高两位决定 1/4N 超前码和落后码的取值如下表：

码序列	NCO 最高两位	码取值
1/8 超前码取值	其他	当前码
	11	超前码
1/8 滞后码取值	其他	当前码
	00	滞后码

因此，当峰值相关器设置为相关器 4 时，可以让相关器 3/5 分别采用超前滞后 1/4 码片的码序列，而相关器 2/6 分别是超前滞后 1/2 码片的码序列。此时相关器间隔为 1/4 码片。同样的，可以让相关器 3/5 分别采用超前滞后 1/8 码片的码序列，而相关器 2/6 分别是超前滞后 1/4 码片的码序列。此时相关器间隔为 1/8 码片。

2.3.3 BOC 码

除了 GPS L1 C/A 码以外，其他的 L1 频点信号都是 BOC 信号。所以，跟踪的时候可以选择以类 BPSK 方式跟踪一个旁瓣，或者采用 BOC 方式跟踪完整的 BOC 信号。

当采用 BOC 跟踪方法的时候，本地码在移入码移位寄存器的时候，将 BOC 调制也同时叠加到伪随机码上。因此，码 NCO 产生两倍码速率的信号并进行二分频作为码生成信号时，是每移位两次生成一个新的码。所以，在一个码的前一半将码生成器的输出送给码移位寄存器，而在后一半将其反相后送给码移位寄存器。

需要注意的是，在采用 BOC 跟踪的时候，由于相关峰的宽度要比普通 BPSK 调制信号的相关峰宽度窄很多，所以建议此时采用 1/8 码片间隔的窄相关以实现正常的跟踪。

2.3.4 数据和导频通道

除了 L1 C/A 码以外，其他的 L1 频点信号都同时存在数据通道和导频通道的信号。通常在跟踪的时候，都是跟踪导频通道信号的。这是因为导频通道的信号可以采用更长的相干积分时间，而且导频信号的幅度往往也比数据信号要高。

在跟踪导频通道的时候，一般同时需要对数据通道上的导航电文进行解码。因此，在这种情况下，用于相关器 0 的本地码不再是原先的本地码，而是改用第二个码发生器产生的伪随机码。此时，码发生器 1 配置为产生导频通道的伪随机码，码发生器 2 配置为产生同一颗卫星的数据通道的伪随机码。两个码发射器采用同样的码计数。需要注意的是，相关器 0 采用的码发生器 2 的码需要延迟两个码片，这样码延迟可以和位于相关器 4 的峰值相关器对齐。

2.3.5 NH 码的处理

除了 L1 C/A 码以外，其他的 L1 频点的导频信号上都附加了额外的 NH 码，所以为了让硬件完成相干累加，需要在 1 毫秒的相关结果上对 NH 码进行剥离。在相关器硬件中实现了一个 25 比特的 NH 编码寄存器，用于存储最长达 25 比特的 NH 码序列；一个 5 比特的 NH 码长度寄存器，用于指定 NH 码 的长度；一个 5 比特的 NH 码计数器，用于指定当前使用的 NH 码在 NH 码序列中的位置。

当使能了 NH 码的时候，当前的 NH 码的输出直接异或在码发生器的输出上来实现 NH 码的剥离，每当码发生器完成了一个码周期后，NH 码计数器加 1，直到计数到 NH 码长度后清零。

对于 Galileo 的 E1C 信号，NH 码的长度就是 25，因此只需要在 NH 编码寄存器中填充 NH 码序列，然后根据当前的 NH 码位置设置正确的 NH 码计数器就可以了。对于 L1C 和 B1C 信号，由于此时 NH 码的长度为 1800 比特，所以需要将 NH 码切成若干段，并用软件进行切换。需要注意的是，尽管 NH 编码寄存器的长度是 25，但是不能将 NH 码切成 25 比特的段。这是因为，由于每一个 block 的中频数据和卫星信号的码周期不是对齐的，因此每 25 个 NH 比特的时间间隔内处理的中频数据实际上是跨过了 26 个 NH 比特。所以，建议的处理方法是将 1800 比特的 NH 码切成 90 个长度为 20 比特的段，然后每 20 个码周期（200 毫秒），当 NH 码计数值大于 20 后，在 NH 编码寄存器的高位移入新的 20 个比特，然后将 NH 码计数值减 20 进行接续。

2.3.6 相关结果输出

每一个相关器在完成一个码周期的信号相关累加后，将输出累加的结果并清零，为下一个码周期的相关做好准备。由于各个相关器对应的不同延迟的码的码周期结束的时间不同，也就造成了相关器 0 到相关器 7 的相关结果是依次输出的。每个相关通道的相关结果都会输出到各自的 FIFO，然后由相干累加逻辑进行相干累加的运算。每当一个相关周期结束的时候，会有一个 dumping 信号置位，并输出相关器 0 的结果。然后每当 overflow 信号有效，意味这下一个相关器完成了同样的相关周期，因此输出下一个相关器的结果，一直到最后一个相关器结果输出后 dumping 信号复位，直到下一个相关周期结束。

伴随着 I/Q 各 16 比特的相关结束送到 FIFO 的还有当前逻辑通道索引和指示该相关结果对应哪一个相关器的索引值（实际实现中以上两个索引合并为相关值在 Coherent RAM 里面的地址），以及两个标志。第一个标志是相干累加起始标志，该标志表示当前的相关结果是否是相干累加中的第一个数据，也就是数据直接填充到相干累加结果上还是累加到相干累加结果上。第二个标志是覆盖保护标志。

一般情况下，一个数据 block 的长度是 1 毫秒，而一个相关周期也是 1 毫秒，因此每处理一个 block 的中频采样数据，每个相关器的值会各输出一次。但是，有三种情况下，一个 block 的处理可能输出同一个相关器两次相关结果。情况一是设置了 Correlator Jump 且 Jump Count 为正时，此时会有强制置位的 overflow 信号推动；情况二是 block adjust 为正，此时一个 block 的实际长度超出了 1 毫秒；情况三是虽然 block 的长度正好为 1 毫秒，但是码多普勒为正。这三种情况下，都有可能出现在一个 block 的数据起始正好完成了一个相关周期，而在一个 block 的结尾完成了下一个相关周期。此时，如果第二次输出为相干累加的第一次累加，则会覆盖掉之前累加的结果。相关值覆盖保护就是在这种情况下保护之前的累加结果不被覆盖。

每一个相关通道在处理一个 block 的数据中第一次输出相关结果时，都会记下第一次产生相关结果的相关器的索引，当该相关器再次输出相关结果，且是相干累加的第一个数据时，会将相应的覆盖保护标志置位，相干累加模块会根据标志位进行相应的保护操作。

2.4 码生成器

码生成器是通道相关器的一部分，但由于涉及到多种不同的码的生成，因此以单独的一节进行介绍。实际上，通道相关器的设计是具有通用性的，配合不同的码发生器就可以跟踪不同的卫星信号。

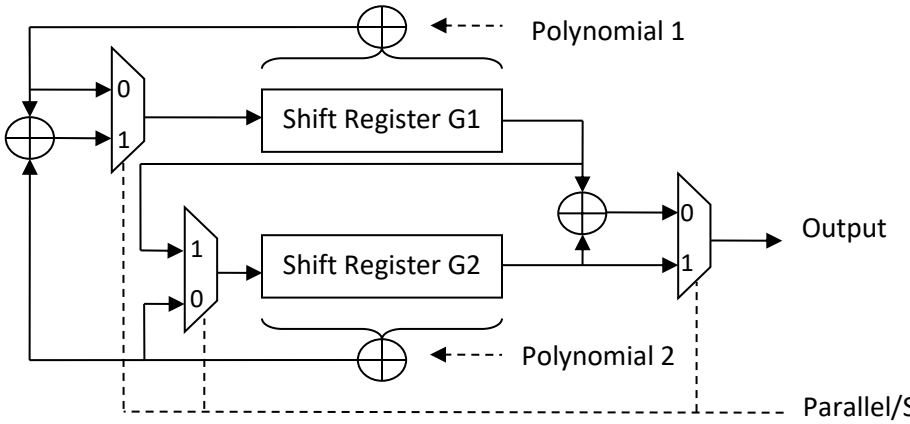
为了支持全部 L1 和 L5 频点的信号，设计了三种不同的码生成器：第一种是通用移位寄存器，用来生成 Gold 码，用于 GPS L1C/A 和 GPS、Galileo 和北斗在 L5 频点信号的伪随机码；第二种是 Weil 码生成器，用来生成 GPS L1C 和北斗 B1C 信号的伪随机码；第三种是 Memory Code 生成器，用来生成 Galileo E1 信号的伪随机码。

三种码生成器的外部接口是类似的，都包含配置输入、载入和读取状态寄存器的接口（用于时分复用通道切换）、控制生成下一个码的信号以及初始化信号（用于捕获引擎中的码生成）。而码的产生也是根据配置寄存器的内容和状态寄存器的内容决定的。三种不同的码配置寄存器和状态寄存器的含义不同。

下面分别对这三种码生成器进行详细介绍。

2.4.1 通用移位寄存器

通用生成器通过配置两个 14 位 Gold 码生成器，产生所需要的 PRN 码序列。下图为 PRN code 生成器整体结构框图：



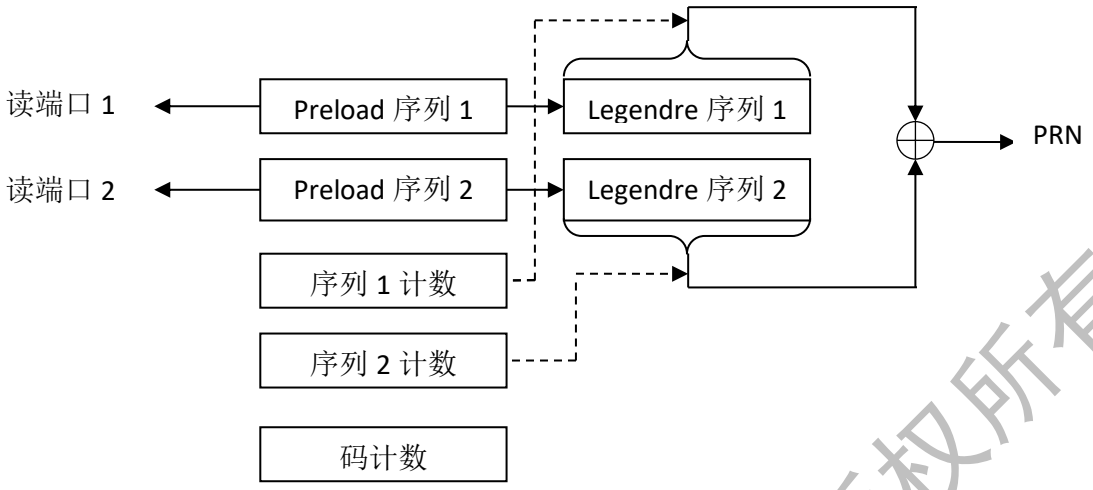
PRN 生成器由两个 14 比特移位寄存器构成。移位寄存器可以配置初相及生成多项式。另外还有一个总的计数器，计到相应计数值后将两个移位寄存器都重载初相。两个移位寄存器分别工作，输出为各自输出相异或。为了保持和其他项目的兼容性，PRN 生成逻辑还支持串行模式，将两个 14 比特的移位寄存器串行连接成一个 28 比特的移位寄存器。

对于每一个卫星系统，生成多项式和码长是固定的，所以相应的寄存器内容由全局寄存器统一设定。而对于同一卫星系统的不同卫星的 PRN 码产生的时候，配置寄存器的内容是两个反馈移位寄存器的初相（即寄存器的初始值），状态寄存器的内容是两个反馈移位寄存器的当前相位（即寄存器的当前值）和计数器的计数值。

2.4.2 Weil 码生成器

由于 GPS 和北斗的 Weil 码生成的逻辑类似，因此用同一个模块产生，通过端口 weil_type 的不同电平选择 Weil 码的类型。两个码各自的 Legendre 序列存储在外部 ROM 中，两个 Legendre 序列通过各自不同的读端口进行读取，每次读取 16bit。访问 GPS 还是 BDS 的 Legendre 序列 ROM 也将由信号 weil_type 决定。由于数据和导频各需要一个 Weil 码生成器，因此一个跟踪通道会有两个 Weil 码生成模块各自产生数据和导频信道的码，也就是总共有 4 个外部 Legendre 序列 ROM 访问端口。这 4 个端口可以通过一个 arbiter 复用访问同一块 Legendre 序列的 ROM。如果有四个物理跟踪通道同时运行，可以再加一级 arbiter。按照典型的采样率为 4 倍码速率计算，每 4 个时钟周期移出一个 bit，则每一个 Legendre 序列读端口平均需要 64 个时钟周期读一次 Legendre 序列，因此四个物理通道的 16 个 Legendre 序列读取单元通过 arbiter 复用可以满足需求。

Weil 码生成逻辑的结构框图如下：



序列 1 计数和序列 2 计数分别对应两个 Legendre 序列的计数值，而码计数值计数的整个码序列的计数。Legendre 序列 1 和 Legendre 序列 2 分别是一个 16 比特的寄存器，通过序列计数的低位选择其中的一个比特输出。当一个序列寄存器的值输出到最后一个比特以后，则通过各自的 Preload 序列寄存器进行加载，这样可以避免在连续输出的时候，由于来不及读取后续的数据导致输出错误。在 Preload 序列寄存器将其中的值加载到 Legendre 序列寄存器后，会根据序列计数的值读取下一个 16 比特的序列值。

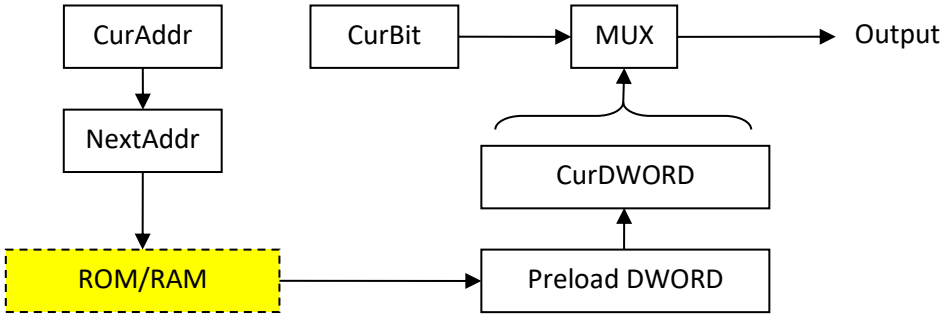
控制生成当前码的是配置寄存器和状态寄存器的组合。对于不同的卫星，配置寄存器是相位差和截取点（B1C 的码）或者相位差和插入点（L1C 的码）。状态寄存器是两个 Legendre 序列寄存器和码计数器。其他寄存器的内容在载入码计数之后会自动计算和填充。

在正常进行通道切换载入状态寄存器的时候，Legendre 序列寄存器的内容会一并载入，同时也会自动读取 Preload 寄存器的内容。在运行过程中，会不断更新 Legendre 序列寄存器的内容直到运行结束，将 Legendre 序列寄存器和码片计数器的内容会进行存储待下次载入的时候进行恢复。

需要注意的是，在 Weil 码生成器初始化到初相的时候，只需要指定配置寄存器的内容，码计数器会自动清零， Legendre 序列寄存器的内容也会自动读取。不需要额外载入。这一功能会用在捕获引擎的码生成上。

2.4.3 Memory Code 生成器

Memory code 用于生成 Galileo E1 信号的数据通道或导频通道的伪随机码。通过预先存储在 ROM 或 RAM 里面的码作为本地码。由于码长度是 1023 的整数倍，所以存放在 ROM/RAM 中的数据以 128 字节为一个 segment，每个 segment 的最后一个 DWORD 的 LSB 忽略，这样一个 segment 正好可以放 1023bit 的码。Memory Code 生成逻辑的结构框图如下：



存放 Memory Code 的存储器的宽度是 32 位，因此 Memory Code 的读取是以 32 比特的 DWORD 为单位，然后数据进行串行地输出。CurDWORD 是一个 32 比特的寄存器，用于存放当前正在输出的 DWORD 的数据，CurBit 选择 CurDWORD 中的哪一个比特作为输出。为了避免在当前 DWORD 输出完成加载下一个 DWORD 的时候出现中断，所以会预读取下一个 DWORD 的内容到 Preload DWORD 寄存器中。CurDWORD 输出完成后，马上将 Preload DWORD 的内容拷贝到 CurDWORD 中，并申请读取下一个 DWORD 的内容。

控制生成当前码的是配置寄存器和状态寄存器的组合，对于不同 SVID 的卫星产生数据或导频通道伪随机码的时候，配置寄存器的内容是码存储在 Memory 中的起始地址以及码的长度（以 1023 个比特为单位，伽利略设置为 4），状态寄存器的内容是当前输出的 CurDWORD 的内容以及码片计数器的计数值。

在正常进行通道切换载入状态寄存器的时候，CurDOWRD 寄存器的内容会一并载入，同时也会自动读取 Preload DWORD 寄存器的内容。在运行过程中，会不断更新 CurDWORD 寄存器的内容直到运行结束，将 CurDWORD 寄存器和码片计数器的内容会进行存储待下次载入的时候进行恢复。

需要注意的是，在 Memory Code 生成器初始化到初相的时候，只需要指定伪随机码在 Memory 中的起始地址，码片计数会自动清零， CurDWORD 寄存器的内容也会自动读取。不需要额外载入。这一功能会用在捕获引擎的码生成上。

在 RTL 实现中，由于四个物理通道中每一个包含两个 Memory Code 生成器（分别用于生成数据和导频通道的伪随机码），因此这 8 个 Memory Code 生成器可能需要同时读取 SRAM，这里采用一个简单的 arbiter 来解决读冲突的问题。存放 Memory Code 的 ROM/RAM 的大小由码的总长度来决定。对于 E1 信号，扩频码长度为 4092bit，即占用 512 字节，以一个频点最多支持 16 个通道计算，需要 8kB 的 RAM。如果采用 ROM，则 50 个数据通道的码和 50 个导频通道的码都需要存储，因此 ROM 的总的大小为 50kB。

2.5 相干累加

相干累加模块的任务是将相关器输出的数据与 Coherent RAM 里面的数据进行相干累加。。由于各个相关通道可能同时输出相关结果，因此每个物理相关通道连接到各自的相关数据 FIFO。相关数据 FIFO 的深度为 8，宽度为 44bit（包括 32 比特的相关结果，10 比特的地址以及两个标志位）。

相干累加模块将数据累加到相应的地址上。如果累加起始标志置位，则表示该结果为第一次的累加结果，只进行存储，不需要累加。如果覆盖保护标志置位，则表示要进行相关值覆盖保护。此时为了避免覆盖上一次的相干累加结果，本次的相关结果不会进行累加运算，而是把相关结果以及对应的存储地址存放到另外的寄存器中。由于在绝大多数情况下，需要进行覆盖保护的概率不大，所以对于所有通道的所有相关器，只存储一个结果。

软件在读取相干累加结果的时候，需要同时看覆盖保护寄存器中是否有有效值。如果有的话，软件在读取原有的相干累加结果后，需要将覆盖保护数据寄存器的值写回到覆盖保护地址寄存器中去。这样，在后续的数据处理中，硬件就会自动将之后的相关结果继续进行累加了。

由于相干累加模块需要同时处理 4 个物理通道相关器的输出，因此需要在 4 个物理通道相关输出 FIFO 中进行查询。读取数据既可以按照轮询的方式，也可以按照固定优先级的方式。当相关结果输出 FIFO 的深度是 8（也就是相关器个数）的时候，无论采用哪种方式都不会造成数据丢失。

相干累加的时候不进行溢出保护，这是因为即使进行了溢出保护，如果 E/P/L 三个相关器全部溢出，则溢出保护也无法帮助正确进行码鉴别。软件主要是通过设置 pre-shift 和 post-shift 移位值来防止溢出。

2.6 通道切换

通道切换是实现通道时分复用的核心功能，主要是由 Fill/Dump 模块完成的（这里的 Dump 与相关数据的 dump 不同，是通道状态 dump）。总共有 32 个通道，全局的通道使能寄存器表示哪一个通道处于 active 状态。每当 TE FIFO 有了一个完整的 block 的数据后，控制逻辑寻找逻辑通道号最小的被使能的四个通道，从 State Buffer 顺序加载四个通道物理相关器中寄存器的内容。完成了四个物理通道的相关运算后，将状态寄存器的内容写回到 State Buffer 中。此后，如果有未处理通道，则给 FIFO 发送 Rewind 命令，然后继续进行后续相关处理，否则给 FIFO 发 Skip 命令。控制寄存器和状态寄存器的内容合并为若干个 32 比特的 DWORD。每一个逻辑通道，寄存器的内容在其中占 16 个 DWORD，8 个相关器的部分相关结果在其中占 8 个 DWORD，相干累加的结果也与寄存器内容在一起存放在 State Buffer 中，占用 8 个 DWORD。因此每一个逻辑通道共需要 32 个 DWORD 的存储空间。如果系统支持 32 个逻辑通道，则需要 4kB 的存储空间。

State Buffer 中一个逻辑通道的内容排列如下图所示：

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	CARRIER_FREQ	CarrierFreq																															
1	CODE_FREQ	CodeFreq																															
2	CORR_CONFIG	Reserved					Coh Number					BitLength						Reserved				(7)		(6)		(5)	(4)	(3)	R	(2)		(1)		
3	NH_CONFIG	NH Length					R	NH Code																									
4	COH_CONFIG	Reserved																	DumpLength														
5	PRN_CONFIG	Buffer of Code Generator																															
6	PRN_STATE																																
7	PRN_COUNT																																
8	CARRIER_PHASE	CarrierPhase																															
9	CARRIER_COUNT	CarrierCount																															
10	CODE_PHASE	CodePhase																															
11	PRN_CODE	DumpCount																	JumpCount											PrnCode				C0
12	CORR_STATE	NH Count					Coherent Count					BitCount						PrnCode2				R		(11)		(10)	CurrCor			R		(9)		(8)
13	MS_DATA	DecodeData																															
14	PRN_CONFIG2	Buffer of Second Code Generator																															
15	PRN_STATE2																																
16		Partial Accumulator I of Correlator 0																	Partial Accumulator Q of Correlator 0															
17		Partial Accumulator I of Correlator 1																	Partial Accumulator Q of Correlator 1															
…		…																	…															
23		Partial Accumulator I of Correlator 7																	Partial Accumulator Q of Correlator 7															
24		Dumped Accumulator I of Correlator 0																	Dumped Accumulator Q of Correlator 0															
25		Dumped Accumulator I of Correlator 1																	Dumped Accumulator Q of Correlator 1															
…		…																	…															
31		Dumped Accumulator I of Correlator 7																	Dumped Accumulator Q of Correlator 7															

(1) Pre Shift (2) Post Shift (3) Data in Q Branch

(4) Enable Second Code Generator (5) Enable BOC (6) DecodeBit

(7) Narrow Factor (8) Coherent Done (9) Overwrite Protect

(10) Dumping (11) Code Subphase

Legend:

Configuration Field

Status Field

其中码生成器的字段排列及含义随不同的码生成器有所不同，具体的排列方式如下图所示：

General PRN Generator

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5/14	PRN_CONFIG	0	0	R		G2 InitState														G1 InitState													
6/15	PRN_STATE	R				G2 CurState														G1 CurState													
7	PRN_COUNT	GlobalCount																		G1 CurCount													

Weil Code

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5/14	PRN_CONFIG	1	0	(1)	R	Insertion Index														Weil Index													
6/15	PRN_STATE	Legendre Code 2																		Legendre Code 1													
7	PRN_COUNT	Reserved																		Current Count													

Memory Code

	Alias	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
5/14	PRN_CONFIG	1	1	Reserved																Start Index												Length	
6/15	PRN_STATE	Current Code																															
7	PRN_COUNT	Reserved																		Current Count													

(1) Weil Code Selection

Note: 4MSB of PRN_CONFIG2 is ignored, use 4MSB of PRN_CONFIG instead

Global polynomial registers

For parallel

Polynomial Set Reg0	0	R		G2 Polynomial															G1 Polynomial													
Polynomial Set Reg1	GlobalLength																		G1 Length													

For serialize

Polynomial Set Reg0	1	R		G2 Polynomial															G1 Polynomial													
Polynomial Set Reg1	GlobalLength																																

2.7 噪底计算单元

噪底计算单元用于计算输入信号的噪声功率。噪底计算单元计算得到的噪底有两个作用，第一个是用于 C/N0 计算中，作为用于分母的噪声功率（幅度），第二个是监测数字中频输入的平均信号功率是否正常。在正常情况下，数字中频输入的信号中大部分的功率为噪声，在 RF 前端正常完成 AGC 的控制的情况下，数字中频上的功率是基本稳定在某一个预期值上，当计算得到的噪声功率偏离预期值太远时，表示 AGC 出现问题，此时需要程序去干预 AGC。

为了尽量准确估计带内噪声功率，噪底计算单元计算的是±1MHz 带宽内的平均功率。具体的实现方法是将物理通道 0 第一轮运行时（也就是对应逻辑通道 0）剥离多普勒后得到的采样点与一个通过该通道的码时钟控制产生的长度为 1023 的 m 序列进行相关累加，在一个 m 序列循环周期内的累加结果作为基准噪声。此基准噪声的平均功率也就是归一化的 1ms 内的噪底（如果逻辑通道 0 的 PreShift 不为 0，需要进行相应的移位补偿）。

为了使计算更加简单，对噪底的处理以幅度值代替能量值进行计算。每次计算得到 1ms 的复数相关结果后，首先先计算其幅度值，然后通过公式$N_{n+1} = N_n + k(A - N_n)$更新平滑后的噪底幅度。其中$N_n$为上一次的平滑噪底幅度，$N_{n+1}$为更新后的平滑噪底幅度，$A$为当前计算得到的 1ms 相关结果的幅度，$k$为平滑系数。根据不同的情况，$k$可以取值 1/256 到 1/16384。平滑系数越小，得到的噪底就越稳定，但是相应的对噪底变化的响应就越慢。

之所以可以用幅度的平均值代替能量的平均值进行噪底计算，是因为假设了 1ms 的累加值的实部和虚部都是同样分布的高斯噪声。因此，其幅度符合瑞利分布，且数学期望值为$\sigma\sqrt{\frac{\pi}{2}} \approx 1.2533\sigma$。因此，只要对平滑后的幅度补偿相应的系数$\sqrt{\frac{\pi}{2}}$就可以得到原来的高斯噪声的平均功率。

2.8 数据解码

对于同时有数据通道和导频通道的信号，在配置上会用相关器 1~7 做导频通道的相关，用相关器 0 做数据通道的相关。此时用相关器 0 的相关结果可以进行数据通道的解码。但是如果相干积分时间超过了一个数据比特的长度（进入稳定跟踪后，相干积分时间往往会有 20 到 40 毫秒），那么相关器 0 的结果会包含多个数据比特，此时就无法完成数据解码了。在这种情况下，需要用硬件来完成数据的解码。

对于数据通道和导频通道同时存在，并且对导频通道进行相位锁定的情况下，数据通道的数据可以通过解析积分结果的 I 路或者 Q 路（取决于数据和导频通道是否调制在正交项上）来确定数据通道上的调制数据。数据解码根据信道编码可以采用硬判决或者软判决。如果是硬判决的话，可以简单地取符号位，如果是软判决的话会比较复杂。通常来说，对于卷积码来说，采用 4 比特的软判决进行维特比译码基本上可以满足性能要求。对于 LDPC 编码来说，推荐使用 8 比特的符号进行 LDPC 的解码。因此数据通道的解码要考虑上述不同的需求。在使用硬件进行解码的时候，

需要在基本保证不溢出的情况下，尽量保留更多的有效比特位数，所以需要根据不同的情况在 16 比特的相干累加结果中选择不同的比特位置。

由于数据通道的的积分长度不会超过导频通道的积分长度，而数据通道的信号功率也是小于等于导频通道的信号功率，因此如果导频通道的相干积分值不溢出，数据通道的相干积分值也不会溢出，所以不需要考虑在数据比特长度的积分时间中积分值溢出的问题。

通常来说，即使考虑到天线增益，数据通道的等效载噪比也不会超过 46dBHz（如果超出了也会在解码逻辑中提供溢出保护），以此计算数据通道积分值的范围。在白噪声最优量化情况下，一毫秒积分的噪声方差值大约在 625（4.113MHz 采样率，单独 I 路或 Q 路），由于 46dBHz 情况下 1ms 的信噪比为 16dB，由$\mathrm{SNR} = 10 \cdot log_{10}\frac{A^2}{2\sigma^2}$，带入$\sigma = 625$和 SNR=16，则计算得到 A 大约为 5576。由此得到，未经过移位的积分值，B1C 或 L1C 的 10ms 累加值的绝对值大约在 55760（用 17 比特表示），而 E1B 的 4ms 累加值的绝对值大约在 22304（用 16 比特表示）。

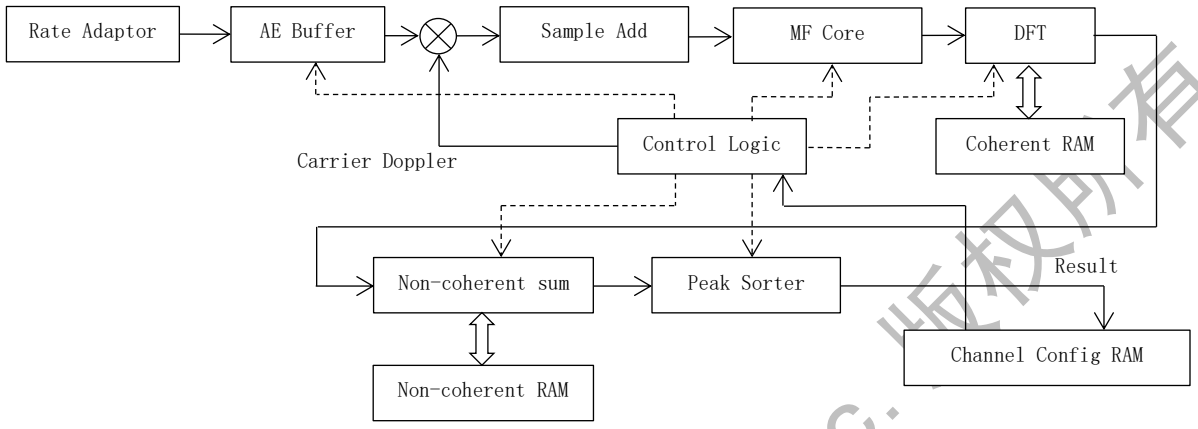
在有 pre-shift 和有 post-shift 的情况下，累加值会相应减小。因此首先计算比特选择因子 $S = B_{pre} + B_{post} - L$，这里 L 在 4ms 累加时为 0，在 10ms 累加时为 1，可以看到无论对于累加长度是 4ms 还是 10ms，在 S 等于 0 的时候，累加值最大范围不超过 16 比特；S 等于 1 的时候，累加值最大范围不超过 15 比特；以此类推。

所以，在 S 等于 0 的时候，从累加值的最高位（bit15）向后取 4 比特或 8 比特作为数据符号的取值；在 S 等于 1 的时候，从累加值的次高位（bit14）向后取 4 比特或 8 比特作为数据符号的取值；以此类推。如果更高位不全是符号位（截取数据时存在溢出），则进行相应的溢出保护。

通过这种方法，可以自动补偿软件设置不同参数带来的累加值的变化，尽量保留更多的比特位数以提高数据解码的成功率。

3. 捕获引擎设计

下图给出了捕获引擎的整体结构框图。它可以支持 GPS L1C/A、GPS L1C、Galileo E1、北斗 B1C 四个信号的捕获。



首先，将待捕获的卫星信号剥离掉标称中频后，降采样到两倍码速率，然后送进 AE buffer 进行捕获。GPS L1C/A，信号为 BPSK 调制，而其他三个信号是 BOC(1,1)调制，仅捕获一个旁瓣，也就是说载波剥离的时候要偏移 1.023MHz。

捕获引擎的核心功能模块是匹配滤波器。它把输入样点和 RPN 码进行相关后通过加法树进行累加得到当前码相位上的相关值。随着新的样点移入匹配滤波器并进行新的相关及累加，等效于 PRN 码在样点序列上滑动。

匹配滤波器的输出首先会与四个不同的旋转因子相乘并进行累加来得到 8 个不同频点的 DFT 结果。DFT 运算的中间结果会以 10bitI+10bitQ+4bitExp 的格式（后文称为 exp10 格式）存储在相干累加 RAM 中。在相干累加完成后，相干累加的结果会计算幅度，并将幅度进行非相干累加。非相干累加的中间结果以块浮点的形式存放在非相干累加 RAM 中。

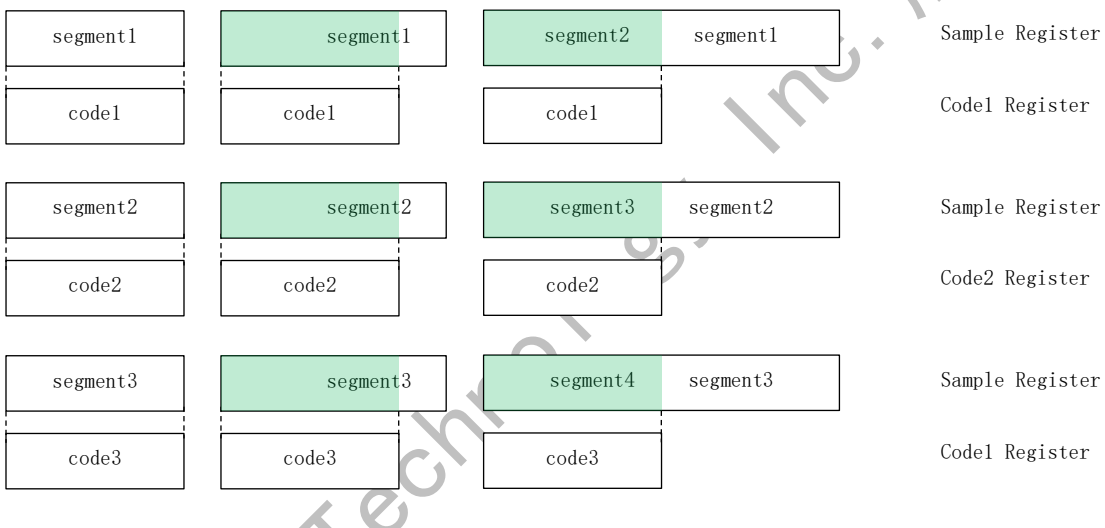
最终，非相干累加的结果经过一个峰值比较和检测单元找出最大的三个幅度峰值并判断捕获是否成功，以及捕获到的峰值所对应的多普勒和码相位。

3.1 捕获引擎运行时序

捕获引擎处理的输入样点的采样率通常是两倍码速率，与之对应的捕获的相关间隔是 1/2 码片。对于不同的系统，码长是不一样的。这也意味着捕获引擎要处理不同数目码相位的捕获。对于码长为 1023 的 GPS L1C/A 码来说，所需要搜索的码相位个数是 2046，Galileo 为 8184，而对于 L1C 和 B1C，需要完全进行捕获的码相位个数是 20460。但是上述所有信号的码速率都是 1.023MHz。

为了节省逻辑以及减少 RAM 的使用，匹配滤波器的深度定为 682，也就是 1023/3。这意味着如果需要完成 1ms 的相关，需要将三次匹配滤波器的结果进行相加。同时，一次可以进行搜索的码相位的个数也是 682，因此搜索完全部的码相位需要进行多轮循环才能完成。

匹配滤波器包含一套 682 个样点的移位寄存器（I/Q 各 6 比特）和两套 341 个比特的码移位寄存器。对于每一轮的 682 个码相位搜索中进行相干和非相干累加时，从 AE Buffer 读样点和匹配滤波器进行相关运算是连续进行的，下图给出了匹配滤波器运行的时序：在开始的时候，样点寄存器和第一套码寄存器都填入 1/3 毫秒的数据（也就是 682 个样点和 341 个码，称为一个 segment），也就是 Preload。随着相关运算的进行，新的样点逐渐进入样点寄存器，同时移出的样点被丢弃。当最后一个码相位的相关完成后，样点寄存器会填满下一个 1/3 毫秒的样点，因此，下一轮累加可以从这个状态开始。在进行相关运算的时候，同时将下一个 segment 的 341 个伪随机码填充另外一套码寄存器。当全部的 682 个码相位相关完成的时候，切换码寄存器的选择，然后匹配滤波器继续进行相关运算。如下图所示：



假设这些 1/3 毫秒的数据段可以标记为 s1、s2、s3 等等。而码也被分为 c1、c2、c3…几部分。首先 c1 被载入码寄存器，因此上图第一行表示了在第一次匹配滤波相关运算过程中样点移入的初始状态、中间状态以及结束状态。此时，码寄存器切换为使用 c2，随着样点的移入，第二行的初始状态逐渐变为第二行的最终状态。然后再切换到使用 c3 码，继续移入新的样点。可以看出，此时，累加的结果就是从 s1*c1+s2*c2+s3*c3…这样的码相位一直搜索到 s2*c1+s3*c2+s4*c3+…一共 682 个码相位。上述每一次匹配滤波器输出的结果，对同样的码相位进行累加或者是相干累加，或者是非相干累加。

当所有的相干累加和非相干累加完成后，就搜索了 682 个码相位。接下来，Preload 就从 s2 作为起始位置开始载入样点而不是从 s1 开始载入，也就是从偏移量 682 开始读取 AE Buffer。按照同样的步骤，可以从 s2*c1+s3*c2+s4*c3+…这样的码相位一直搜索到 s3*c1+s4*c2+s5*c3+…这 682 个码相位。重复以上过程，直至全部的码相位就搜索完成了。

对于捕获引擎的完整功能来说，是由多个嵌套的循环完成了。最内层的循环是将匹配滤波器通过移动样点输出 682 个不同的码相位的相关结果。第二层循环是把匹配滤波器加法树输出的同相位的结果进行三次累加得到 1 毫秒的相关值。第三层循环是把匹配滤波器累加后输出的 1 毫秒相关值在乘以 DFT 的旋转因子后累加起来完成相干累加。第四层循环是把相干累加得到的幅度进行非相干累加。第五层循环是码起始相位循环，通过多次以不同起始码相位进行 682 个码相位的搜索拼凑出完整的码相位搜索结果。由于捕获的带宽受限于相干累加长度，而且即便是最短的 1 毫秒相干累加，搜索的带宽也只有 500Hz，因此第六层循环是在频域上搜索不同的多普勒。第七层循环是搜索不同的 PRN 码。这一层循环通常是通过为不同的通道配置不同的参数实现的。捕获引擎典型的通道个数是 32 个。

捕获引擎在一次配置完启动后，可以不需要 CPU 的干预就完成全部可配置通道的搜索，硬件逻辑会在自动完成上述七层循环的捕获后再通过中断通知 CPU 任务已经完成，并可以让 CPU 查询捕获的结果。

3.2 码速率调整采样 Rate Adaptor

码速率调整重采样模块主要完成三个工作：第一个是将输入信号的标称中频部分进行剥离，变成零中频的信号；第二个是进行滤波后降采样，将采样率降低到两倍码速率，即 2.046MHz；第三个是对重采样后的样点进行重新量化，变成 I/Q 各 2 比特的 sign/mag 格式的信号。

剥离标称中频，采用的是和跟踪引擎中一样的 32 比特载波 NCO 和 6 比特索引的 sin/cos 查找表。通过一个单边带带宽为 1MHz 的低通滤波器滤波后，用一个 24 比特的 NCO 进行降采样，将采样率降低到 2.046MHz。由于设计的中频信号采样频率为稍大于 4MHz，因此可以认为滤波器带宽与采样率为一个固定比例关系，也就是滤波器的系数可以采用固定值。为简化设计，滤波器采用 6 阶 FIR 滤波器，滤波器系数为-5、8、20、20、8、-5。这样，在进行滤波器的时候，只需要加减法就可以实现。在对降采样后的信号进行重量化时，由于幅度只有一个比特，因此只需要一个量化门限。假设经过自动增益控制后（或者是射频前端芯片完成或者由前端预处理完成），信号的平均能量基本稳定，那么此时的量化门限不需要进行自适应调整，只需要采用软件的预设值即可。

3.3 中频采样存储 AE Buffer

AE Buffer 的主体是一块 32bit 宽度的 SRAM，用来存储用于捕获的样点。根据所需要的捕获灵敏度的不同，比较典型的中频采样存储空间的大小为 64kB~256kB。按照 2.046MHz 的采样率，每个采样点 I/Q 共 4 比特，则存储的采样数据的长度在 64ms 到 256ms。在本项目的工程实现中，采用的 128kB 的 SRAM。对于输入 AE Buffer 的采样数据，如果 Rate Adaptor 的中频配置是标称中频，则采样信号用于捕获 BPSK 调制的 L1 C/A 信号；如果 Rate Adaptor 的中频配置偏移了 1.023MHz，则采样信号用于捕获 BOC(1,1)调制的其他信号。

采样信号的采样率是两倍码速率，每个样点是 sign/mag 格式的 I/Q 各 2bit 共 4bit 构成。由于 SRAM 的宽度是 32bit，因此每次访问可以并行读或写 8 个样点。通过设计读写逻辑，可以通过单

个端口同时进行读写访问而不会引起访问冲突（这样允许在实际使用的时候可以在 AE buffer 没有填满的情况下开始捕获进程，尽管这样对软件控制启动的时刻需要精确控制）。当第一个采样点填充进入 AE Buffer 的时候，AE Buffer 会发出一个触发信号，这个触发信号可以用来锁存 TE FIFO 的指针或者 PPS 的计数器。这样在捕获完成得到信号的码相位后，可以根据同步锁存的计数值进行捕获到跟踪的转换。捕获引擎首先会在 AE Buffer 中填充样点，然后反复读取以进行信号捕获。读取时，AE buffer 支持从非 0 地址开始读取，以便完成不同的起始码相位的搜索。

3.4 PRN 码生成器

虽然捕获引擎的码生成模块采用了和跟踪引擎同样的码生成模块，但是为了简化设置，对不同系统的码生成模块进行了封装。外部通过 2 比特的系统选择和 6 比特的 SVID 选择，就可以生成所选择系统的 PRN 码。无论是生成 Weil 码，还是生成 Galileo 所用的 memory code 时，与跟踪引擎中的码生成模块一样，都需要访问外部的存储 Legendre 码或 memory code。为了节省实现逻辑，捕获引擎与跟踪引擎复用同样的存储空间。此时由于捕获引擎载入 PRN 码为可延迟的任务（不同于跟踪引擎需要连续运行，捕获引擎状态机会等待 PRN 码加载完成才会进入到下一步），因此捕获引擎读取数据的优先级是低于跟踪引擎的。

3.5 多普勒搜索补偿

由于匹配滤波器是对零中频的数据进行的相关运算，并且相干积分限制了频域搜索的宽度，因此在搜索更宽的带宽的时候，需要在频率上进行多次搜索，每次搜索都是将搜索的中心频率搬移到零中频再输入到匹配滤波器。每一次搜索的步长称作一个 stride。由于信号的多普勒较小的概率要大于多普勒较大的概率，因此频域搜索是从零多普勒，或者预测的信号多普勒开始向两侧进行搜索的，也就是说每次搜索的中心频率分别是将 0、1、-1、2、-2……乘以一个 stride。

频率搬移仍然使用 32 比特 NCO 控制的 64 个索引的 sin/cos 查找表，但是 sin/cos 的增益为 7。由于 AE Buffer 中采样点实部和虚部的取值为±1 或者±3，所以在与从查找表得到的复数载波相乘的时候，可以用加法代替乘法。

3.6 匹配滤波器

匹配滤波器是捕获引擎的核心模块，其实现的功能是将输入的样点序列与码序列做点积。输入样点和码分别进入到深度为 682，宽度分别是 12bit 和 1bit 的移位寄存器。由于搜索间隔为 1/2 码片，所以原理上码寄存器由码生成器产生的码分别重复一次。输入样点是 I/Q 各 6bit 的，码的取值为±1（0 表示 1，1 表示-1）。

在实际实现的时候，有若干实现技巧来简化逻辑的复杂度。

第一项优化技术，由于 PRN 码都被重复两次，因此，实际匹配滤波器核心中参与相关的数据不再是 682 对的样点和码，而是 341 对。当把采样点输入到匹配滤波器的样点寄存器的时候，首先会进行两两相加。也就是说，输入的样点不再是 s1、s2、s3、s4…，而是 s1+s2、s2+s3、s3+s4…。在进行相关的时候，码只会与奇数位置的样点进行相关并累加。所以，在第一个时刻，码 c1 会与

s1+s2 进行相关，码 c2 会与 s3+s4 进行相关，以此类推完成全部的 341 个码和 682 个样点的相关。到下一时刻，随着新的样点移入样点寄存器，s2+s3 替代了 s1+s2，s4+s5 替代了 s3+s4，直到最后。此时，相关累加的结果就是移动了半个码相位的结果。通过这一的方法，把相关累加的数量减半，但也相应要求匹配滤波器的深度是 2 的倍数。

第二项优化技术，是避免求补码相反数的运算。做相关时，如果相应的码是 0，则样点维持原有值，如果相应的码是 1，则样点变成原来的相反数。由于在对二进制补码求相反数时需要加 1 的运算，因此为了节省在相关运算中每一个求相反数都需要加 1 的逻辑，我们把加 1 放在全部累加的最后。对于一般的 PRN 码来说，其中有约一半是 1，因此可以认为在每 1023 个码中一共有 512 个 1。所以求相反数就用求反代替（其实是将码与和码相关的样点的每一比特分别异或），然后在最后加 512 补偿所有的+1。由于每 1023 个码（2046 个样点）对应的结果是分成 3 个 segment 完成的，所以为了基本平均地把需要加的 1 补偿在结果上，三个 segment 分别补偿 160、160、192。

第三项优化技术，是将加法树通过逐比特累加的方式进行。加法树通常是在整个捕获引擎中占用最大面积的逻辑。因此，优化这一部分的逻辑可以有效降低捕获引擎所占的面积。当我们考虑一个以二进制补码表示的 6 比特数 $b_5b_4b_3b_2b_1b_0$ 时，它可以表示为如下的值：$b_5x(-2^5)+b_4x2^4+b_3x2^3+b_2x2^2+b_1x2^1+b_0x2^0$。其中 b_5 到 b_0 的取值是 0 或者 1。所以，我们可以把所有样点的 b_0 加起来，并乘以一个 2^0 的比例因子，再把所有样点的 b_1 加起来，并乘以一个 2^1 的比例因子，直到把所有样点的 b_5 加起来，并乘以一个(-2^5)的比例因子。把所有的 341 个 1 比特数累加的逻辑对于所有的比特位来说是一样的。考虑到 I 和 Q 各 6 比特一个有 12 个比特需要累加，因此总共需要 12 个 1 比特的加法树。需要注意的是，由于最高位的比例因子是-2^5而不是 2^5，因此最高位的累加结果需要求相反数，也就是求反后再加 32。相应地，把这个值跟第二项优化中的加 1 补偿合并在一起，三个 segment 需要补偿的值变成了 192、192、224。

最终的累加结果是 16 比特的 I 和 16 比特的 Q，并送给相干累加单元做进一步处理。

3.7 DFT 和相干累加

通过相干累加可以有效地增加捕获的灵敏度，但是同样的，捕获的带宽被限制在$\pm 1/3T$ 的范围内，其中 T 是相干累加时间。为了增加捕获的带宽，提高捕获效率，在进行相干累加的时候，采用了 DFT 的方法进一步对相关结果进行处理。每一毫秒的相干结果都与 4 对不同的旋转因子相乘（每一对旋转因子包含一对共轭的复数）并分别进行相干累加，从而获得 8 个不同频点上的相干累加结果。DFT 的公式如下所示：

$$X(k) = \sum_{n=0}^{N-1} x(i)e^{-j2\pi kn\Delta\varphi}, k = \pm 1,3,5,7$$

这里 N 表示 DFT 的长度（也就是相干累加的长度）。在上式中，每一对共轭的旋转因子都可以一起进行计算，也就是：

$$x(i)(\cos\theta \pm i \cdot \sin\theta) = (I + i \cdot Q)(\cos\theta \pm i \cdot \sin\theta)$$

$= (\mathrm{I}\cdot\cos\theta \mp \mathrm{Q}\cdot\sin\theta) + \mathrm{i}\cdot(\mathrm{Q}\cdot\cos\theta \pm \mathrm{I}\cdot\sin\theta)$

因此，采用 4 次实数乘法，并将结果进行不同的加减，就可以得到与一对共轭旋转因子相乘的两个结果。由于需要平均分配各个频点的间隔，DFT 的结果的频点相对于中心频点分别有±Δf、±3Δf、±5Δf 和±7Δf 的偏移量，此时每两个频点之间的距离是 2Δf。

用于计算旋转因子的 sin/cos 查找表是一个 256 个索引的，10 比特数值的查找表，查找表的索引采用一个 14bit 的 NCO 的高 8bit，每个条目的 sin/cos 值在-511 到 511 之间。

DFT 频率控制字与 DFT 频率间隔选择Δf的对应关系如下：

当旋转因子的频率是Δf时，每两个 1 毫秒相关值之间需要乘的旋转因子的相位变化值是$\Delta\varphi = \Delta f\cdot \mathrm{T} = 0.001\Delta f$，因此可以得到$\Delta f$与 DFT 旋转因子的索引增量Δi（即 DFT 的频率控制字）的关系式通过如下方式计算：

$$\frac{2\pi \mathrm{i}}{2^{14}} = 2\pi \mathrm{kn}\Delta\varphi = \frac{2\pi \mathrm{kn}\Delta f}{1000}$$

$$\mathrm{i} = \frac{2^{14}\mathrm{kn}\Delta f}{1000}$$

$$\Delta \mathrm{i} = \frac{\mathrm{i}}{\mathrm{kn}} = \frac{2^{14}\Delta f}{1000}$$

比如，一次搜索的搜索范围为 500Hz，其中 8 个 DFT 的频点均匀分布在 500Hz 之内，每两个频点的间隔 2Δf =500Hz/8=62.5Hz，于是Δf =31.25Hz。由此，计算得到对于 k=1 频点旋转因子索引的增量Δi为 512，对于 k=3 频点旋转因子索引的增量3Δi为 3x512 以此类推。或者将索引每次增加 512，索引值分别乘以 1、3、5、7 得到四个频点旋转因子的索引。此时 DFT 的频率控制字为 512。

相干累加长度会影响搜索带宽。一般相干累加长度越长，stride 越小。因此需要根据相干累加长度调整 stride 的大小。而通常由于 8 个 DFT 频点会均匀分布在一个 stride 的搜索范围内，所有 DFT 频率控制字也需要相应调整。此时，由于 2Δf为一个 stride 间隔ΔF（或 stride 的搜索范围）的 1/8，因此有$\Delta f = \frac{\Delta F}{16}$，也由此得到 DFT 频率控制字的值为$\Delta \mathrm{i} = \frac{2^{10}\Delta F}{1000}$。

匹配滤波器的输出依次为 682 个不同的码相位的结果，接下来输出的是 682 个码相位下一毫秒的输出结果，于是，需要一个容纳 682 个中间结果的相干累加 RAM。由于每一个结果采用 exp10 的格式存储，占用 24bit（3byte），所以总共 682 个码相位乘以 8 个频点的中间结果总共需要 3x682x8=16kB 的空间。

当相干累加完成后，所有的 682 个码相位和 8 个频点的相干累加结果就存放在相干累加 RAM 中。在最后一次相干累加完成的时候，其中最大的指数值会被统计出来，在输出给非相干累加模块的时候一起提供用来对所有的相干累加结果进行归一化。

由于相干累加 RAM 中的数据会被频繁地读写，所以读写的时序需要仔细地规划和设计。为增加读写数据率，相干累加 RAM 的宽度可以设计成 24x8=192bit，做到一次访问可以读写一个码相位的全部 8 个频点的结果。

需要注意的是，最后一次相干累加的结果并不会在计算完成后直接输出给非相干累加模块。由于需要统计本次累加值中最大的指数以进行归一化，因此最后的相干累加结果依然会写回到相干累加 RAM 中，并在完成本次相干累加后再读出来输出给非相干累加单元。也就是，对于第一轮相干累加来说，由于相干累加 RAM 里面有上一次累加的结果数据，所以仍然需要有读操作把数据读出来。对于最后一轮相干累加来说，累加的结果需要写回到相干累加 RAM 中去。也就是说每一个匹配滤波器输出的相关结果都会对应于相干累加 RAM 的一次读和一次写。

3.8 非相干累加

由于存在若干限制使相干累加长度无法无限延长，所以在相干累加次数达到一定值后，需要再进行非相干累加。对于延长相干累加时间最大的限制就是搜索带宽变窄和调制数据引起的累加损失。因此，尽管非相干累加并不能提高信噪比（进行非相干累加后，峰值增加的比率和噪底增加的比率是相同的），但存在和不存在信号两种情况下的概率密度函数的区别还是会随着非相干累加次数的增多而更加明显。因而，非相干累加可以在同样虚警概率的情况下获得更高的捕获概率，也就是等效地提高了捕获的灵敏度。而且，增加非相干累加的时间并不会带来上述增加相干累加时长所造成的损失。

为简化处理逻辑，非相干累加采用累加幅度而不是累加能量的方法。与相干累加一样，非相干累加也需要一个非相干累加 RAM 来存放中间结果，非相干累加 RAM 中依次存放所有码相位和所有 8 个频点的非相干累加数据。与相干累加不同的是，第一轮的非相干累加不需要读操作，幅度值直接写入非相干累加 RAM，而最后一轮非相干累加的结果也不需要再写回到非相干累加 RAM，而是直接输出给峰值搜索模块。全部的码相位和频率的相干累加的幅度值采用块浮点的方式存储，每个值占用 8bit，而所有的值共用一个指数。相干累加的输出会首先计算幅度，并归一化为 8bit 的值。在将相干累加模块的输出与当前非相干累加的中间结果进行相加的时候，指数会取两者中较大的一个并把另外一个进行归一化。在累加的过程中，两个 8bit 的结果最大有可能到 9bit，也就是在非相干累加的过程中可能会出现结果溢出，并需要将指数加 1 来将溢出的结果调整回 8bit，后续的结果都会采用新的指数。因此指数加 1 在一轮非相干累加过程中最多只会发生一次。所以，相干累加模块会记住新的指数值，并将调整发生的位置记下来。在下一轮进行非相干累加从非相干累加 RAM 读取中间结果的时候，凡是未到调整位置的结果，都需要右移一位以配合新的加过 1 的指数，而超过调整位置的结果就不需要调整了。如果在过程中没有发生溢出，调整位置就是 0，意味着从第一个位置起用的就是当前指数值。

与相干累加 RAM 一样，非相干累加 RAM 的数据宽度也是 8 个频点的数据，也就是 64bit。每一个非相干累加结果占用一个字节，非相干累加 RAM 的大小就是 682x8=5.3kB。

非相干累加模块还会在最后一轮累加的时候计算噪底。每次 8 个频点的幅度累加结果会求平均值，并把 682 个码相位的平均值相加得到一个 18 位的噪底。噪底在搜索完成的时候，可以用来跟峰值比较以进一步对判断信号捕获是否成功提供参考。需要注意的是，在设置了提前结束并且信号能量很高时，非相干累加会提前结束。此时有可能不会计算噪底。在这种情况下，一般来说噪底也不再需要。如果确实需要一个噪底参考值，那么其他通道的噪底可以起到同样的作用。

3.9 峰值比较器

峰值比较器会比较非相干累加的结果，并在全部的码相位和全部的搜索频段中选择全部的峰值中最大的三个。峰值比较器中存储的峰值包括信号幅度，对应的码相位和对应的多普勒频率。

对于每一个非相干累加模块输出的 8 个频率的幅度值，峰值比较器首先会从中挑出一个最大值进入峰值比较器。如果这个最大值大于峰值比较器中已有的任何一个峰值，则会将其中的一个进行替代。替代的策略取决于当前峰值是否和已有的峰值位置重合。如果位置重合的话则替代重合的峰值，否则作为新的峰值加入，原有峰值中的最小的一个将被丢弃。重合的判决是两个峰值的码相位和频点相同或临近（也就是说频率和相位同时满足差值不超过 1）。对于频域进行多个 stride 搜索和码相位进行多轮搜索的情况，频率 stride 值和码相位搜索的轮数需要和非相干累加模块当前的频点值和码相位的值进行组合。

由于上面所描述的峰值替代逻辑，当一个新的峰值插入峰值比较器以后，根据峰值大小比较的结果和峰值位置重合匹配的结果，有如下几种情况（其中原有的峰值分别表示为 1、2、3，新插入的峰值表示为 X）：

Case number	Value position
0	123
1	X12
2	1X2
3	12X
4	X23
5	X13
6	1X3

而采用哪种插入后的结果，是由如下的峰值大小比较和峰值位置匹配的组合决定的。

	No match	Match 1	Match 2	Match 3
X<3	0	0	0	0
X>1	1	4	5	1
1>X>2	2	0	6	2
2>X>3	3	0	0	3

从上表可以看出，根据组合的 16 种情况，会有 7 种不同的插入结果。而对于每一个峰值位置来说，在插入后的新值有三种选择（峰值 1 只有两种），分别是不变、取上一个更大的峰值、取新插入的值。因此，硬件设计的时候可以根据上述组合确定一个 2bit 的控制信号，用来控制新值的锁存。

如果允许捕获成功提前结束，那么每一轮非相干累加完成后，峰值比较器都会进行一次判断，如果最大的峰值超过第三大的峰值某一个倍数，则判定捕获成功，此后不会再进行本通道任何后

续的搜索，直接结束当前通道，开始下一个通道的搜索。这样的处理可以在强信号的条件下缩短捕获所需的时间。

3.10 通道配置及捕获流程

搜索 GNSS 信号的时候，需要有很多不同的配置。每一个通道都可以进行不同的配置。捕获引擎会根据每一个通道不同的配置执行不同的搜索。绝大部分的配置会合并成一个数组，并且在软件进行配置的时候存放在一块 SRAM 的空间中。捕获引擎在完成一个通道的搜索后，也会把捕获的结果写回到这块 SRAM 空间。当捕获引擎完成全部通道的搜索后，CPU 可以通过查看 SRAM 空间的内容得到捕获的结果。

下表列出来一个通道可以进行配置的参数以及相应的说明：

名称	范围	说明	备注
PrnSel	0~3	系统选择。00~11 分别表示 GPS L1C/A、Galileo E1C、BDS B1C 和 GPS L1C。	
Coherent Number	1~63	相干累加长度	
Noncoh. Number	1~127	非相干累加长度	
Stride Number	1~63	频率搜索 stride 的个数	
Read Address	0~31	AE buffer 读取样点的起始地址偏移，比例因子为 682	
Code Span	0~31	需要搜索的码相位个数，比例因子为 682	
Center frequency	-2^{19}~2^{19}-1	搜索的中心频点，比例因子为 2^{12}	等效 0.5125/Hz
Stride Interval	0~2^{22}-1	Stride 的间隔	等效 2099.202/Hz
SvId	1~63	所要搜索卫星的 SVID。从 1 开始表示 SV1，一直到 63 表示 SV63。不同系统取值范围不同。	
DFT Frequency	0~2^{11}-1	DFT 频率间隔	等效 16.384/Hz
Early Terminate	0/1	是否允许搜索提前结束	
Peak Ratio	0~7	捕获成功的阈值比例。0~7 分别表示判断捕获成功的条件是最大峰值超过第三大峰值的 1.125~2 倍，以 0.125 倍为间隔。	

以上的参数和搜索结果一起，每个通道会合并为 8 个 32bit 的 DWORD。如果捕获引擎最多支持搜索 32 个通道的时候，SRAM 的大小为 32x8x32bit=1kB。

一般来说，需要同时搜索的卫星数不会超过 32 个。但是在某些情况下，需要配置超过 32 个通道。比如说，stride 的个数最多配置为 63 个，以每一个 stride 的为 500Hz 计算，总共的频率搜索范围是 32kHz，也就是±16kHz。这对于一般的多普勒来说已经足够，但是如果需要搜索更宽的范围，可以把搜索范围分成两部分，分两个通道进行搜索。

每一个通道的通道配置和结果存放在 8 个 32bit DWORD 中，若干个通道的配置在配置 SRAM 中连续排列。通道配置的字段排列如下图所示：

		31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0						②	Ratio				NCSNumber									COHNumber								StrideNumber						
1		PrnSel		SvId										CenterFreq>>12																			
2			DftFreq																		ReadAddress							CodeSpan					
3		StrideInterval																															
4		①				GlobalExp								NoiseFloor																			
5	LOBAL_PEAK0	Mag								FreqBin										Cor													
6	LOBAL_PEAK1	Mag								FreqBin										Cor													
7	LOBAL_PEAK2	Mag								FreqBin										Cor													

① Success
② EarlyTerminate

由于在进行捕获的时候，越靠近标称中频捕获卫星的概率越大，所以在进行频域搜索的时候，硬件会从中心频率开始，依次向两边进行搜索，也就是搜索频点相对于中心频率偏移的 stride 依次是 0、1、-1、2、-2、3…。

在使用捕获引擎进行捕获的时候，首先启动 AE Buffer 填充。AE Buffer 是否填充满的相应的状态指示位可以从状态寄存器查询获得。通道配置 SRAM 的内容可以根据所需进行配置。当 AE Buffer 填充满了以后，就可以启动 AE 捕获功能（理论上可以不需要等 AE buffer 填充满就可以启动 AE）。当 AE 捕获完成后，可以通过中断通知 CPU。CPU 此后可以重新配置通道参数进行下一轮捕获，或者填充新的中频数据。

由于不同卫星信号有不同的码长度，因此需要捕获的码相位的个数也不同。捕获的码相位的个数在字段 Code Span 中进行设置，该字段代表需要搜索 segment（也就是 682 个码相位）的个数。因此，如果需要搜索全部的码相位，GPS L1C/A 需要设置为 3、Galileo E1 需要设置为 12、L1C 和 B1C 需要设置为 30。字段 Read Address 表示了搜索起始位置在 AE Buffer 中的偏移量。由于 AE Buffer 中的样点为 2 倍码速率，因此偏移 1 个样点表示偏移半个码相位。该字段的值表示搜索起始位置偏移多少个 segment（也即 682 个样点）。在搜索全部码相位的时候，Read Address 可以设置为 0，而如果仅需搜索部分码相位，可以通过组合 Read Address 和 Code Span 来完成。

3.11 结果判断及相关峰位置计算

信号捕获的 3 个峰值会存在每一个通道配置字段的最后 3 个 DWORD 中，其中幅度占 8 个比特，频率占 9 个比特，码相位占 15 个比特。所有的峰值的幅度值和 Noise Floor 共用同一个指数值 GlobalExp。其中码相位表示的是对应 PRN 的第一个码的样点在 AE Buffer 中的地址。而多普勒用以下方式进行计算：频率值高 6 比特用 stride 的偏移量，低 3 比特用 DFT 对应的频率，因此频率值是一个有符号数。DFT 的频率间隔通常配置为 1/8 的 stride 频率间隔。所以，低 3 比特的 DFT 值从 0~7 分别表示相对于当前 stride 的中心频点-3.5~3.5 倍 DFT 间隔的频点位置。因此，计算频点位置首先应该减掉 3.5，然后乘以 DFT 频率间隔（即 1/8 的 stride 间隔）得到以赫兹为单位的多普勒。比如，对于 500Hz 的 stride 间隔，-7 的频率值表示多普勒频率在(-7-3.5)x(500Hz/8)=-656.25，23 的频率值表示多普勒频率在(23-3.5)x(500Hz/8)=1218.75Hz。

4. 辅助模块设计

辅助模块是在卫星导航接收机中起辅助作用的模块，最典型的是 PPS。

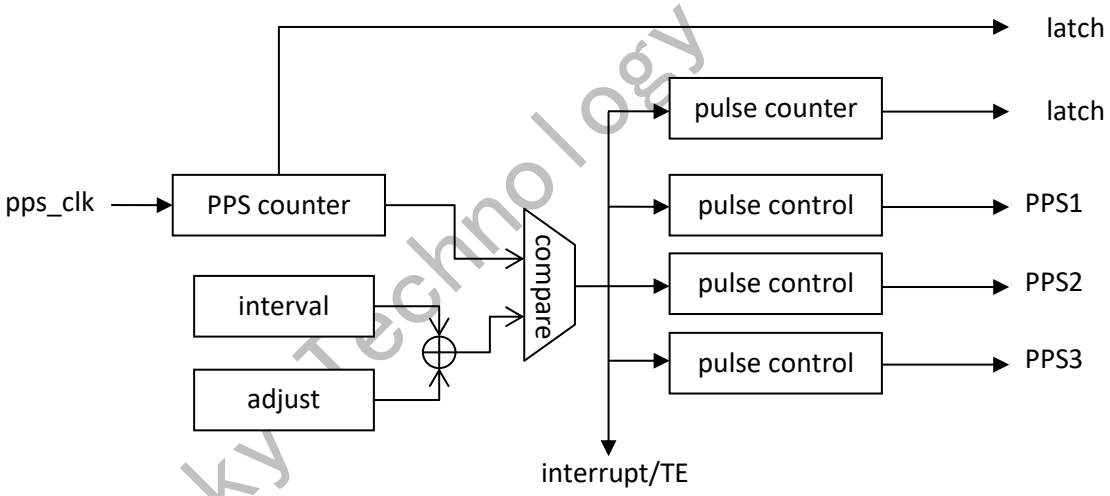
4.1 PPS 模块

PPS 模块的主要功能是提供秒脉冲输出，输出信号的上升沿或下降沿对齐指定系统的整秒时刻。由于秒脉冲沿的时间分辨率取决于秒脉冲的时钟频率，因此更快的时钟频率有助于提升秒脉冲的精确度。基于这种考虑，PPS 模块提供了一个可以和 GNSS 基带异步的 PPS 时钟输入，并且将主要的计数逻辑都放在了 PPS 时钟域里面。跨时钟域的设计将在后面进行介绍。

考虑到多系统的需求，PPS 可以提供三个有不同时间差、不同边沿类型和不同脉冲宽度的 PPS 脉冲输出。由于秒脉冲的时间差基本固定，因此上述三个脉冲输出由同样的主计数器控制。三个 PPS 输出可以分别对齐三个不同星座的时间系统，或者一个 UTC 时间的秒脉冲加两个星座时间系统的秒脉冲输出。

4.1.1 PPS 模块的设计

PPS 的模块设计框图如下：



PPS 时钟驱动 PPS counter 计数，当计数值等于脉冲间隔的时候，会产生一个 PPS 事件。PPS 事件会分别将脉冲计数加 1、产生 PPS 中断、锁存 TE FIFO 的写指针，并且输出到脉冲控制逻辑。脉冲控制逻辑会在各自的 delay 时间过后，产生相应宽度的脉冲输出。为了调整 PPS 沿的位置，与 PPS counter 进行比较的值是脉冲间隔和间隔调整的和。因此，写脉冲调整寄存器可以将下一次 PPS 的脉冲沿提前或延后。脉冲调整寄存器在下一次 PPS 事件后会自动清零，因此只起一次作用。

在 CPU 发出指令，或者有外部事件发生时，PPS counter 时钟计数以及脉冲计数值会进行锁存。作为驱动时钟的 pps_clk 的最高频率设计为 1GHz，因此采用 30 位计数值。考虑到间隔最大值还需

要加上 30 位的调整量，因此计数寄存器为 31 比特。脉冲计数采用 8bit，可以保证锁存值在 256 秒内不会出现循环。

另外，PPS 模块输出的信号电平，在缺省状态下是无效电平，电平值取决于设置的有效输出边沿是上升沿还是下降沿。这就意味着当改变输出边沿极性的时候，输出的 PPS 信号会有翻转造成的边沿，这个边沿相对于所需要的配置来说是无效边沿。但是为了避免误触发，仍然建议在外部输出端设置高阻态进行保护。在 PPS 模块中不会做相应的实现，而是建议在 SoC 集成的时候在相应的引脚上以 I/O 引脚的形式实现。一方面是这样有利于将 PPS 引脚与其他 I/O 引脚进行复用，另一方面是 PPS 引脚往往需要更大的驱动电流，可以将其与三态的实现集成在一起。

4.1.2 PPS 模块的跨时钟域设计

由于 CPU 总线访问、与 GNSS 其他逻辑的交互信号处于系统时钟域中，而 PPS 本身的计数逻辑和控制逻辑处于 PPS 时钟域中，因此对于控制寄存器和状态寄存器的读写需要考虑跨时钟域的问题。

对于控制寄存器和状态寄存器的读操作，可以直接使用 PPS 时钟域中的寄存器值的信号。这是由于控制寄存器的值只会因为 CPU 的操作改变，因此寄存器的值是一个稳定的电平，经过不同的时钟域不会产生改变。状态寄存器的值是 CPU 和外部事件产生的计数器的锁存值。当锁存完成并且下一次锁存未发生时，寄存器的值和控制寄存器一样也是稳定电平，不会因为跨时钟域改变。

对于控制寄存器的写操作，采用如下的操作方式：当系统时钟域的总线上存在写信号的时候，在系统时钟域上首先会锁存写地址和写数据，同时会对一个系统时钟域中的写标志寄存器的值进行翻转。上述操作都在系统时钟域内，因此不会存在数据不完整的问题。在 PPS 时钟域内，会对写标志寄存器的值进行两级锁存，然后通过锁存的值判断是否写标志存在翻转。之所以加两级锁存，是为了形成一定的延时，确保系统时钟域上锁存的写地址和写数据稳定。由于 CPU 通过总线执行寄存器的写操作在程序执行的时候间隔至少会有数个系统时钟周期，因此从 PPS 时钟域的角度看，写标志信号、锁存的写地址和写数据在一定时间段内都是稳定不变的信号，这样在跨时钟域的时候就可以保证数据正确并有足够的保持时间。

4.1.3 PPS 的操作方式

PPS 的操作包括初始化 PPS 寄存器并实现 PPS 沿的初始对准，以及在运行过程中持续调整 PPS 沿以保持和整秒的对齐。其中前者按照以下步骤完成：

1. 当完成卫星信号的跟踪并且 PVT 完成定位后，在 PPS_CTRL 寄存器中使能 PPS（并可选地使能 PPS 中断）
2. 当 PPS 事件发生的时候，TE_FIFO 的 TE_FIFO_LWADDR_PPS 会锁存 PPS 事件发生时的 FIFO 写指针（使能 PPS 中断可以更好地帮助判断 PPS 事件什么时候发生）
3. 通过 PVT 的钟差结果，以 TE_FIFO 的读写指针计算得到 PPS 事件发生对应的某一个卫星系统的时间，将相应的时间与整秒的时间差折算成 PPS 时钟的个数，写入 PPS_PULSE_ADJUST 寄存器。可以额外将 PPS 事件提前一点以便为 PPS 脉冲延时留出一定裕量。

4. 如果 PPS 时钟与射频时钟来自同一个时钟源，可以通过 PVT 的钟漂计算得到 PPS_PULSE_INTERVAL 的值并进行设置。
5. 可选地将步骤 1~3 再执行一遍确保 PPS 事件发生的时刻在期望时间历元上。
6. 设置 PPS 脉冲控制寄存器和脉冲宽度寄存器使能相应的 PPS 脉冲输出。

在运行过程中，同样按照上述步骤 1~3 先计算 PPS 事件的时间与期望时间的时间差后，一种方法是将时间差经过一个 DLL 滤波器，计算出 PPS 事件间隔（相当于脉冲频率），然后通过控制时间间隔的方式调整 PPS 沿跟踪准确时间；另一种方法是根据时间差调整 PPS_PULSE_ADJUST 寄存器，同时根据钟飘或者几次调整 PPS_PULSE_ADJUST 的平均值独立调整 PPS_PULSE_INTERVAL 寄存器的值。

在对 PPS 进行操作的时候，需要注意以下几点：第一，PPS 事件要早于任何一个 PPS 脉冲沿，因此最好在设置的时候就留出几百纳秒到一微秒的裕量；第二，对所有寄存器的调整尽量在 PPS 事件发生后尽快完成；第三，由于所有时间系统的间隔都相对恒定，因此 PPS 的实现是用一个 PPS 事件进行不同的延时产生 PPS 脉冲，因此 PPS_PULSE_ADJUST 的调整会影响到所有的脉冲沿。

PPS 还会根据外部事件以及 CPU 锁存命令来锁存 PPS 时钟计数器以及脉冲计数器的值。由于上述事件同时还会锁存 TE FIFO 的写指针并以此进行时间同步，因此 PPS 锁存上述事件的功能为设计冗余，正常情况下不需要上述功能。