一、GNSS-SDR 简介

1、软件概述

GNSS-SDR（**GNSS: G**lobal **N**avigation **S**atellite **S**ystems、**SDR: S**oftware **D**efined **R**eceiver）是一个用 C++ 实现的 GNSS 软件接收机开源项目。有了 GNSS-SDR，用户可以通过创建一个图来构建 GNSS 软件接收器，图中的节点是信号处理块，线条代表它们之间的数据流。该软件为不同的合适射频前端提供接口，并实现从接收器一直到 PVT 解算的所有功能。它的设计允许任何形式的定制，包括信号源、信号处理算法、与其他系统的互操作性、输出格式的互换，并为所有中间信号、参数和变量提供接口。

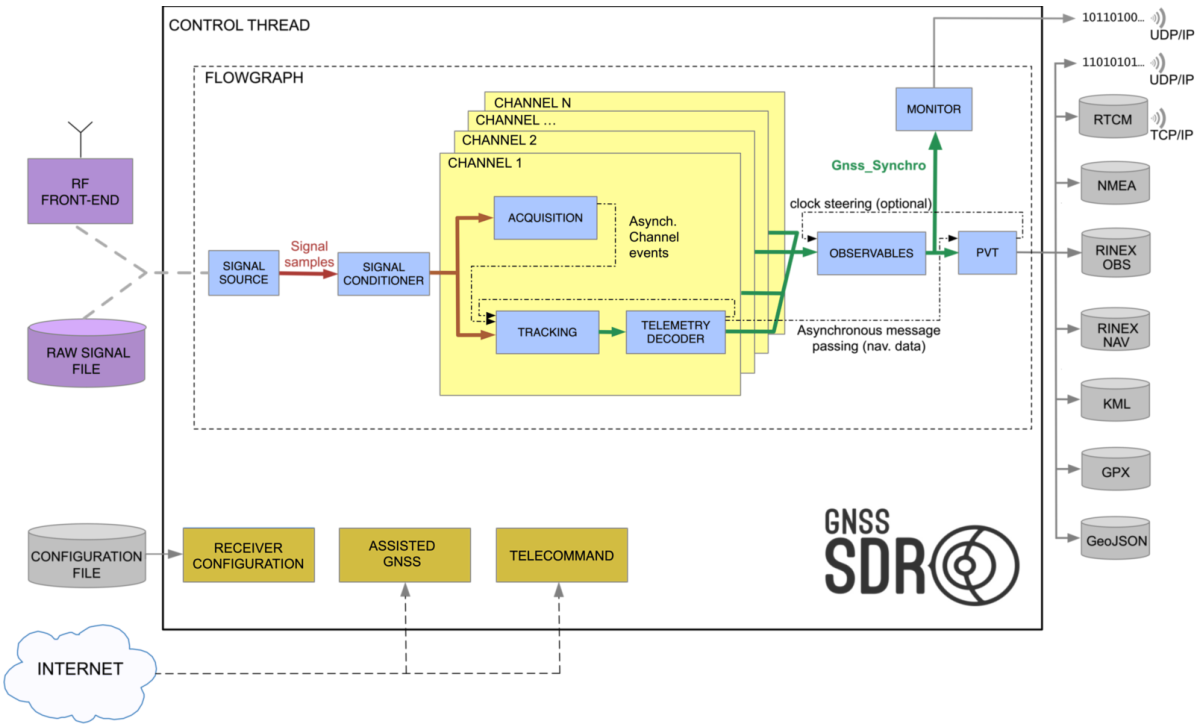
软件旨在促进新信号处理技术的发展，提供一种简便的方法来衡量这些技术对接收机整体性能的影响。通过对每个软件模块进行系统功能验证，以及使用真实和合成信号对整个接收机进行实验验证，对所有流程进行测试。

目前的技术仍无法以卫星发射频率（约 1.5 GHz）对信号进行数字处理，因此我们仍需要一个射频前端，将信号降频到较低频率，在此过程中进行一些滤波和放大，并以一定的速率进行采样，将量化的数字原始采样流传输到计算平台（通过 USB、以太网等）。

软件接收机可在普通的 PC 中运行，并通过 USB 和以太网总线为各种市售或定制的射频前端提供接口，使处理算法适应不同的采样频率、中间频率和采样分辨率。它还可以处理存储在文件中的原始数据样本。软件对可用的卫星信号进行信号采集和跟踪，对导航信息进行解码，并计算定位算法所需的观测值，最终实现完整导航解决方案。处理输出可存储在 RINEX 文件中，或通过 TCP/IP 服务器以 RTCM 3.2 消息形式实时传输。导航结果以 KML 和 GeoJSON 格式存储。

2、代码风格

3、软件框架



> 框架是软件库的一种特例。它们是由定义明确的应用程序接口（API）封装而成的可重用代码抽象，但又包含一些区别于普通库的关键特征：整个程序的控制流不是由调用者决定的，而是由框架决定的，而且用户通常可以通过有选择地覆盖或通过提供特定功能的用户代码进行专门化来扩展框架。软件框架的目的是促进软件开发，让设计人员和程序员将时间用于满足软件要求，而不是处理提供工作系统的更标准的底层细节，从而减少总体开发时间。GNSS-SDR 提出了一种基于 GNU 无线电框架的软件架构，以实现 GNSS 软件定义接收机。

1. 任务并行化

GNSS 基带信号处理需要很高的计算负荷，如果没有适当的软件架构来充分利用处理器，很难实现实时处理。因此，需要利用执行软件接收接收机的处理平台的基本并行性，以满足实时要求。

共享内存并行计算机是架构并行性的一个基本模式，它可以同时处理多个任务，方法很简单，**既可以将任务分配给不同的处理器，也可以在单个处理器中交错执行多个指令流**（这种方法被称为同步多线程，或 SMT），还可以将这两种策略结合起来。SMT 平台、多核机器和共享内存并行计算机都为执行多个独立指令流或线程提供系统支持。这种方法被称为**任务并行**，主要的编程语言、编译器和操作系统都支持这种方法。为了有效提高潜在的性能，平台上运行的软件在编写时必须能够在多个执行内核上分散工作量。为支持这一功能而编写的应用程序和操作系统被称为多线程。如果程序设计得当，执行速度几乎可以与处理内核的数量成线性关系。

任务并行化的重点是在不同的并行计算节点（处理器）上分配执行进程（线程），每个节点在相同或不同的数据上执行不同的线程（或进程）。将处理任务分散到不同的线程中必须经过精心设计，以避免出现瓶颈（无论是在处理过程中还是在内存访问中），从而阻塞整个处理链，使其无法实现实时运行。

2. 理论基础： 卡恩进程网络（Kahn's process networks）

卡恩进程描述了一种计算模型，在该模型中，进程通过通信通道连接起来，形成一个网络。进程产生数据元素或令牌，并沿着通信通道发送，等待的目标进程会使用这些数据元素或令牌。通信通道是进程交换信息的唯一方法。卡恩要求，当进程试图从一个空的输入通道获取数据时，必须暂停进程的执行。例如，进程不能测试输入是否存在数据。在任何给定的时间点，进程都只能启用或阻塞，等待一个输入通道的数据，而不能等待多个通道的数据。符合卡恩数学模型的系统是确定的：通信通道上产生的令牌历史不取决于执行顺序。有了适当的调度策略，就有可能实现软件定义的无线电进程网络，并使其具备两个关键特性：

- **Non-termination**：非终止，可理解为无限运行的流程图进程没有死锁情况。
- **Strictly bounded**：严格受限，对于所有可能的执行命令，通信通道上缓冲的数据元素数量都是受限的。

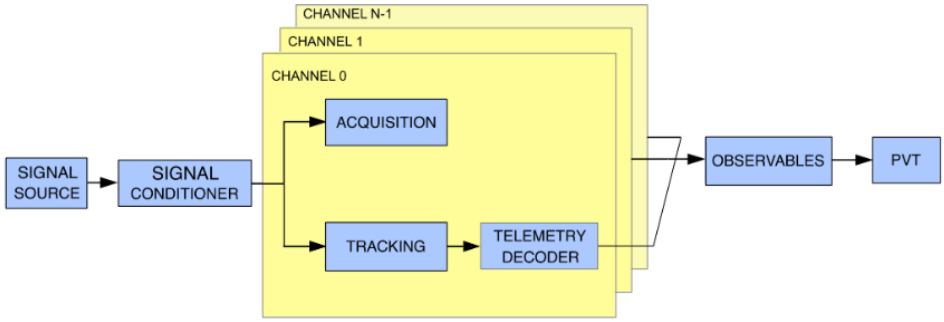
Parks 的博士论文对这种流程网络调度进行了分析。

> 软件定义无线电可表示为节点流图。每个节点代表一个信号处理模块，而节点之间的链接则代表数据流。流图的概念可视为一个非循环方向图（即无封闭循环），其中包含一个或多个源块（向流图中插入采样）、一个或多个汇块（终止或从流图中导出采样）以及介于两者之间的任何信号处理块。

一个最简单的流程图是这样的：

Example of a very simple flow graph.

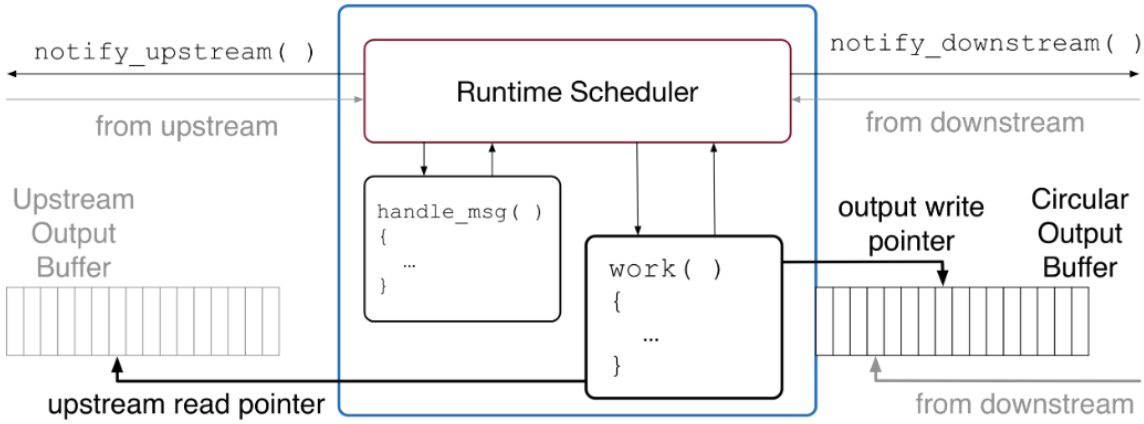
而另一个更复杂的例子大家应该已经很熟悉了，那就是：



Typical GNSS-SDR flow graph.

3. 实现方式：GNU Radio

GNU Radio 是一个用于软件定义无线电应用的免费开源框架，详细介绍见 PPT，它是这些概念的实际应用。除了提供种类繁多的信号处理模块（滤波器、同步元件、解调器、解码器等）外，GNU Radio 还提供了符合上述要求的运行时调度程序实现。这样，开发人员就可以专注于实际信号处理的实现，而不必担心如何将这些处理嵌入到高效的处理链中。通过采用 GNU Radio 的信号处理框架，GNSS-SDR 将其软件架构建立在成熟、高效的设计和经过广泛验证的实施基础之上。GNU Radio 框架实现的处理模块（即流程图中的某个节点）示意图如下：



每个程序块都有一个完全独立的调度程序，运行在自己的执行线程中，还有一个异步消息系统，用于与其他上下游程序块通信。实际的信号处理在 work() 方法中进行。

每个区块可以有任意数量的输入和输出端口，用于数据和与流程图中其他区块的异步信息传递。在所有基于 GNU Radio 框架的软件应用程序中，底层进程调度器将项目（即数据单元）从源传递到汇。对于每个区块来说，它在一次迭代中能处理的项目数量取决于其输出缓冲区的空间大小和输入缓冲区的可用项目数量。这个数字越大，效率就越高（因为大部分处理时间都用在处理样本上），但同时也会增加该程序块带来的延迟。相反，每次迭代的项目数越少，调度程序带来的开销就越大。

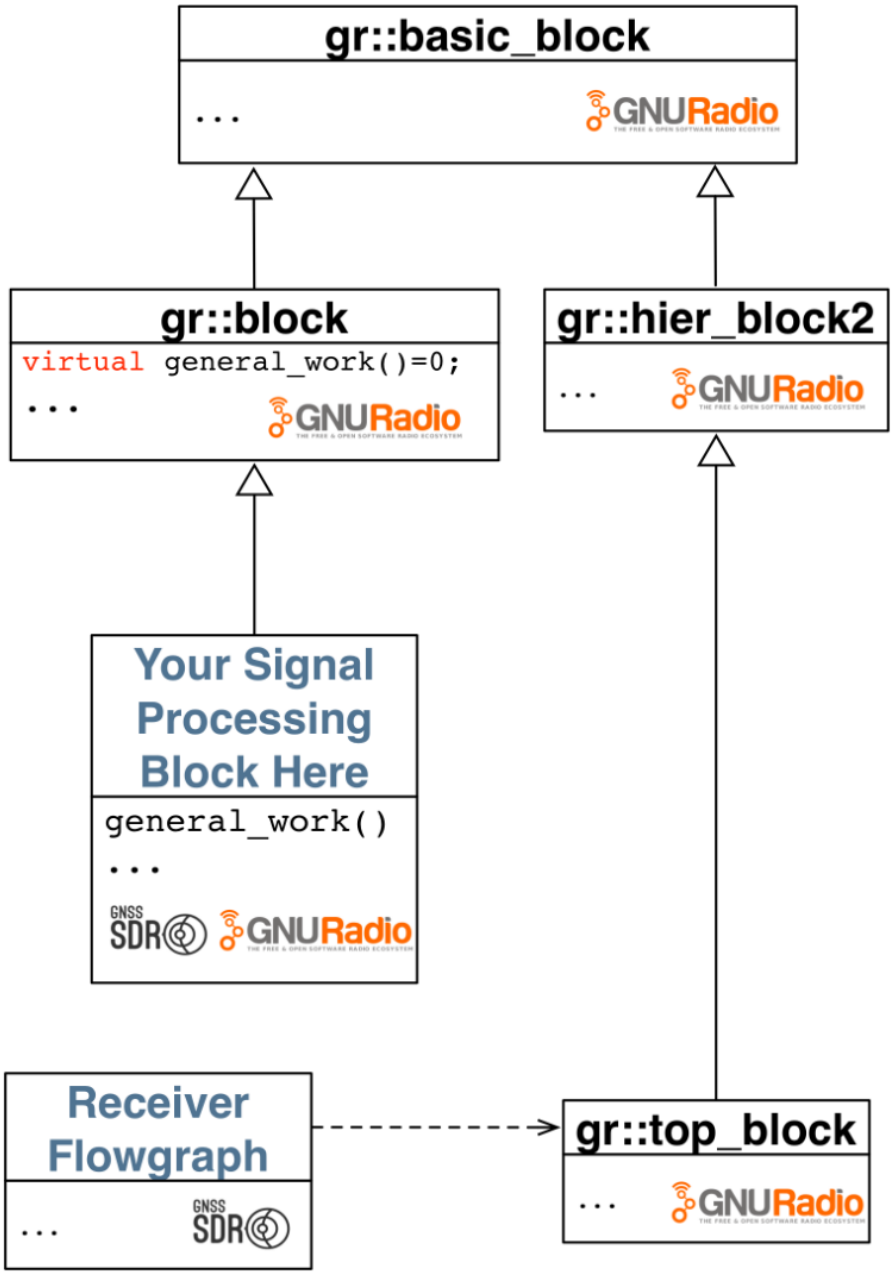
因此，在输入缓冲区的可用项数和输出缓冲区的可用空间方面存在一些限制和要求，以便使所有处理链高效。在 GNU Radio 中，每个区块都有一个运行时调度程序，它可以使用试图优化吞吐量的算法动态执行所有这些计算，实现满足 Parks 博士论文所述要求的进程网络调度。

在这种方案下，软件定义的信号处理模块读取其输入内存缓冲区中的可用样本，以最快的速度进行处理，并将处理结果放入相应的输出内存缓冲区，每个模块都在各自独立的线程中执行。这种策略的结果是，软件接收器总是试图以最大处理能力处理输入信号，因为流程图中的每个区块都是在处理器、数据流和缓冲空间允许的范围内以最快速度运行的，与其输入数据速率无关。要实现实时性，只需在足够强大的处理系统中执行接收器的整个处理链，以承受所需的处理负荷，但这并不妨碍以较慢的速度执行完全相同的处理过程，例如，在功能较弱的平台上从文件中读取样本。

> 每个处理块都在自己的线程中执行，并尝试以最快的速度处理来自其收入缓冲区的数据，而不管输入数据的速率如何。底层运行时调度程序负责管理数据流图中从源到汇的数据流。

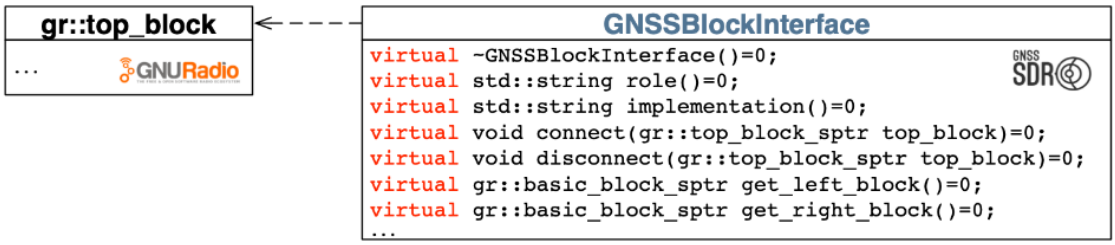
4. GNSS-SDR 的类型设计

面向对象软件设计的一个关键方面是类与类之间的关系。在 GNU Radio 框架中，`gr::basic_block` 是所有信号处理块的抽象基类，是一个具有名称和一组输入输出的实体的抽象概念。它从不被直接实例化；相反，它是 `gr::hier_block2` 和 `gr::block` 的抽象父类，`gr::hier_block2` 是一个递归容器，用于在内部图中添加或删除处理块或层次块，而 gr::block 则是所有处理块的抽象基类。信号处理流程是通过创建分层块树来构建的，分层块树在任何层级都可能包含实际执行信号处理功能的终端节点：

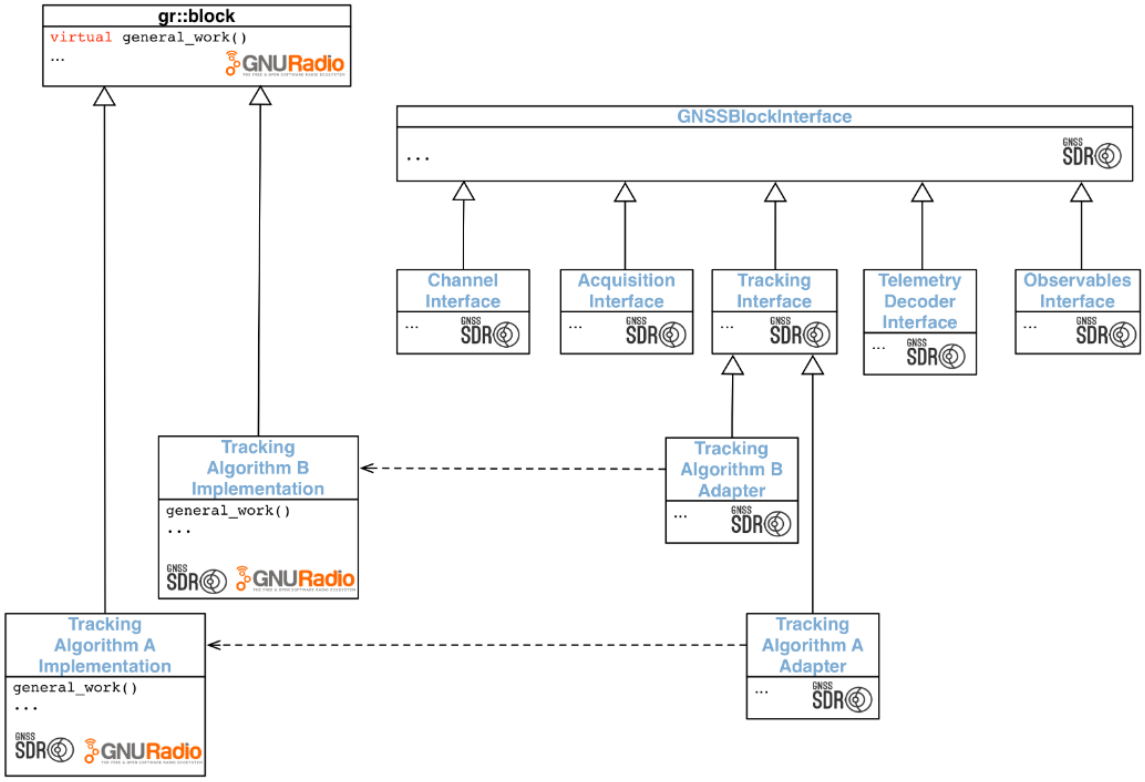


类 `gr::top_block` 是表示流程图的顶级层次块。它定义了程序执行过程中使用的 GNU Radio 运行时函数：`run()`、`start()`、`stop()`、`wait()` 等。它用于根据配置定义接收器流程图（即所有所需区块内的相互连接）。

名为 `GNSSBlockInterface` 的类是所有 GNSS-SDR 模块的通用接口。它定义了派生类必须实现的纯虚拟方法。纯虚拟方法的类被称为抽象类；它们不能直接实例化，只有在该类或父类实现了所有继承的纯虚拟方法后，才能直接实例化抽象类的子类。



通过对 `GNSSBlockInterface` 进行子类化，我们定义了接收机处理模块的接口。这种层次结构如下图所示，可为每个处理模块定义任意数量的算法和实现方法，这些算法和实现方法将根据配置进行实例化。这种策略定义了共享一个共同接口的多个实现，实现了接口与实现解耦的目标：它定义了一个算法族，封装了每个算法，并使它们可以互换。因此，我们可以让算法独立于使用它的程序而变化。

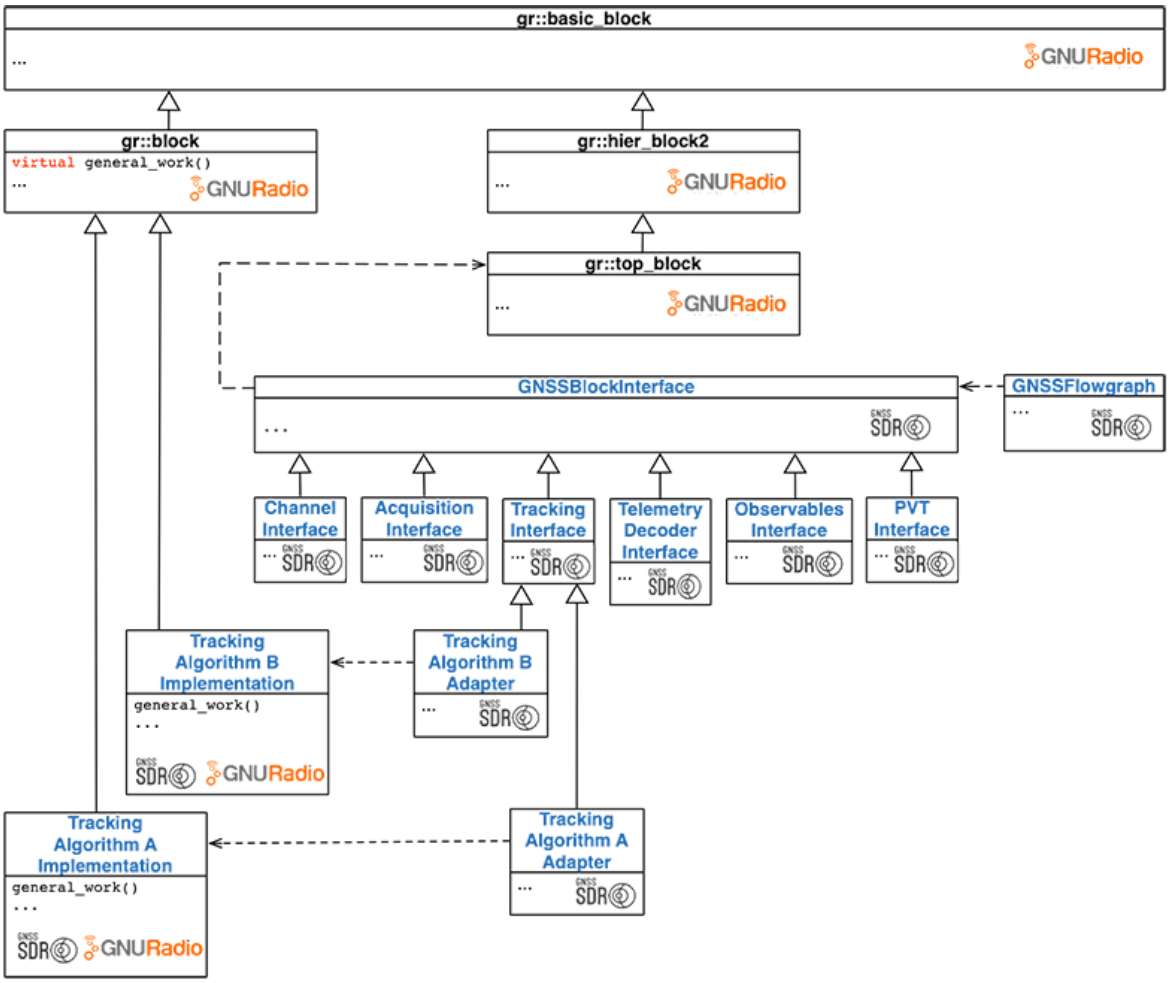


这种设计模式允许每个程序块有无限多的算法和实现方式。例如，定义一种新的信号采集算法需要一个适配器，确保它符合最基本的 `AcquisitionInterface`，并以 GNU 无线电处理模块的形式（即继承自 `gr::block`）进行实际实现。

例如： GPS_L1_CA_PCPS_Acquisition 是一个可用的采集块实现。与其他采集模块一样，它有一个继承自 AcquisitionInterface 的适配器，以及一个继承自 gr::block 并执行实际处理的相应 GNU Radio 模块。您可以查看源代码：

- Adapter interface: gnss-sdr/src/algorithms/acquisition/adapters/gps_l1_ca_pcps_acquisition.h
- Adapter implementation: gnss-sdr/src/algorithms/acquisition/adapters/gps_l1_ca_pcps_acquisition.cc
- Processing block interface: gnss-sdr/src/algorithms/acquisition/gnuradio_blocks/pcps_acquisition.h
- Processing block implementation: gnss-sdr/src/algorithms/acquisition/gnuradio_blocks/pcps_acquisition.cc

下图概述了 GNSS-SDR 的一般类层次结构及其与 GNU Radio 框架的关系：



到此为止，我们已经介绍了一种兼顾效率和可扩展性的软件设计。将全球导航卫星系统接收器建模为一个处理节点流图，其中有一个提供信号采样的源块、一个从其输入缓冲区读取数据并将输出写入其输出缓冲区的节点网络和一个汇块，这样就可以制定高效的流程调度策略。然后，我们提出了一个基于 GNU Radio 框架的软件架构，并为关键的 GNSS 处理模块定义了接口。此外，按照这种方法，我们可以为每个关键的 GNSS 信号处理模块定义数量不限的实现方法，所有这些方法都继承了底层设计，因此很容易重复使用。例如，我们可以为 GPS L1 C/A 信号、伽利略 E1B 等定义获取实现，然后像使用其他现有 GNU Radio 模块一样使用这些模块，从而受益于其内部运行时调度程序或异步消息传递系统等良好特性。不过，我们仍未说明这些模块是如何连接在一起的，整个系统是如何管理的，或者用户如何配置这些模块以定义一个完全自定义的软件定义的 GNSS 接收机。这些都是控制模块的工作。

4、控制模块

控制模块负责根据配置创建流程图，然后管理处理模块。它由四个主要部分组成：

- **configuration mechanism**：一种配置机制，具有足够的灵活性，可容纳数量不确定的算法、实现方式、相关参数和用例；
- **GNSS block factory**：一个全球导航卫星系统模块工厂，封装创建处理模块所涉及的复杂性，并通过一个通用接口引用新创建的对象，从而允许添加新的模块，而无需更改使用该模块的客户端软件的任何一行代码；
- **GNSS flow graph**：一个全球导航卫星系统流程图，管理根据配置创建的处理模块，并将它们连接到一个实现软件接收器的流程网络中；以及
- **Control Thread**：管理整个过程的控制线程。

1. configuration mechanism

配置允许用户通过指定流程图（信号源类型、通道数量、每个通道和每个模块使用的算法、卫星选择策略、输出格式类型等），轻松定义自己的定制接收器。由于难以预测未来模块实现所需的配置，我们采用了一种非常简单的方法，可以在不对代码产生重大影响的情况下进行扩展。只需将处理模块中的变量名称与配置中的参数名称进行映射即可。

属性在程序中通过 ConfigurationInterface 类进行传递。该接口有两种实现： FileConfiguration 和 InMemoryConfiguration。FileConfiguration 对象从文件中读取属性（属性名和值对）并将其存储在内部。相反，InMemoryConfiguration 并不从文件中读取属性；它在实例化后保持为空，而属性值和属性名是通过 set_property 方法设置的。FileConfiguration 用于实际的 GNSS-SDR 应用程序，而 InMemoryConfiguration 用于测试，以避免文件系统中的文件依赖性。

需要读取配置参数的类将接收 ConfigurationInterface 的实例，并从中获取参数值。例如，与 SignalSource 相关的参数应如下所示：

```
SignalSource.parameter1=value1
SignalSource.parameter2=value2
```

这些参数的名称除了一个保留字：implementation（实现）外，可以是任何其他名称。该参数的值表示工厂必须为该角色实例化的类的名称。例如，如果我们要为模块 SignalConditioner 使用 Pass_Through 实现，那么配置文件中相应的一行应为：

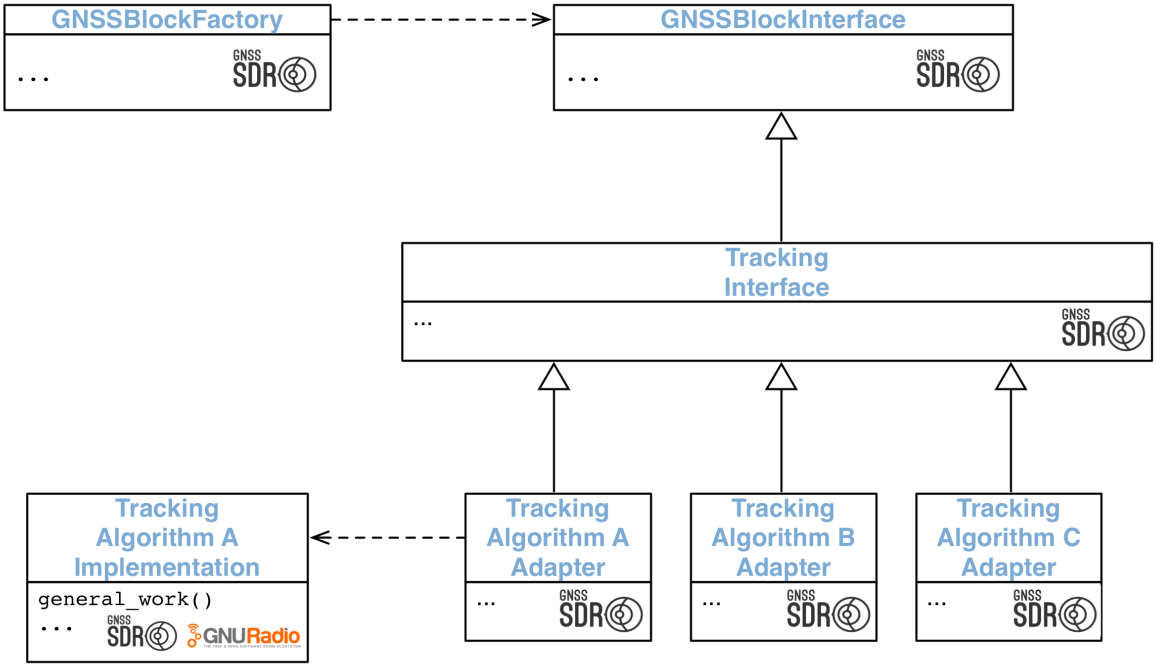
```
SignalConditioner.implementation=Pass_Through
```

由于配置只是一组属性名和值，没有任何意义或语法，因此系统的通用性很强，易于扩展。为系统添加新属性只需要修改使用这些属性的类即可。此外，配置文件不会根据任何严格的语法进行检查，因此它始终处于正确的状态（只要它包含 INI 格式的属性名和值对即可）。INI 文件是一个 -位文本文件，其中每个属性都有一个名称和一个值，格式为名称 = 值。属性不区分大小写，也不能包含间隔字符。分号（;）表示注释的开始，分号和行尾之间的内容将被忽略。

GNSS-SDR 允许用户通过单个配置文件（INI 格式的简单文本文件）定义定制的 GNSS 接收器，包括其结构（波段数、每个波段的信道数和目标信号）以及每个处理模块的特定算法和参数。因此，每个配置文件都定义了不同的 GNSS 接收机。gnss-sdr/conf中提供了一些此类文件的示例。

2. GNSS block factory

软件采用工厂模式，通过定义创建对象的接口来封装创建对象的过程，但让子类来决定实例化哪个类。具体来说，定义了一个简单的访问器类来获取配置对值，并将其传递给一个名为 GNSSBlockFactory 的工厂类。该工厂根据配置决定需要实例化哪个类，以及哪些参数应传递给构造函数。因此，工厂封装了块实例化的复杂性。有了这种方法，添加一个需要新参数的新程序块，就像添加程序块类和修改工厂使其能够实例化一样简单。程序块的实现与配置语法之间的这种松散耦合，可以在很大程度上扩展应用程序的能力。它还允许生产完全定制的接收器（例如，采集算法的测试平台），并在接收器链的任何一点放置观测器。



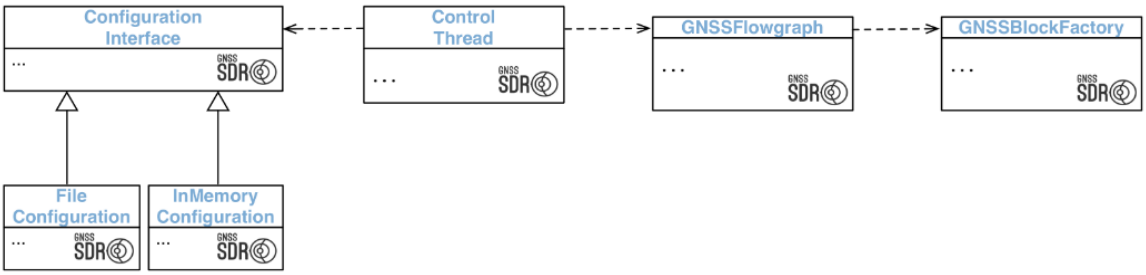
3. GNSS flow graph

GNSSFlowgraph 类负责根据配置准备块图、运行它、在运行期间修改它以及停止它。块由其角色标识。该类知道它必须实例化哪些角色，以及如何连接这些角色以配置下图所示的通用图。它依靠配置来获取所需的角色实例，然后应用 GNU Radio 块之间的连接，使图形随时可以启动。管理块和数据流的复杂性由 GNU Radio 的 gr::top_block 类处理。GNSSFlowgraph 封装了 gr::top_block 实例，因此我们可以利用 GNSS 块工厂、配置系统和处理块。该类还负责在运行时对流程图的配置进行更改，动态地重新配置通道：它选择选择卫星的策略。这可以是对所有卫星 ID 的顺序搜索，也可以是根据对接收器位置的粗略估计来确定哪些卫星最有可能在视图内的更智能的方法，以避免搜索地球另一侧的卫星。

该类在内部对要在图形上执行的操作进行编码。这些操作由简单整数标识。GNSSFlowgraph 提供了一种方法，用于接收编码动作的整数，该方法可触发整数所代表的动作。操作的范围从改变块的内部变量到通过添加/删除块来完全修改所构建的图。行动的数量和复杂程度仅受可用来进行编码的整数数量限制。这种方法封装了准备一个完整图形的复杂性，所有必要的块都已实例化并连接起来。它还充分利用了配置系统和 GNSS 模块工厂，使代码简洁易懂。此外，它还能轻松更新要对图形执行的操作集。

4. Control Thread

ControlThread 类负责实例化 GNSSFlowgraph 并传递所需的配置。一旦定义了流图并连接了图块，它就开始处理传入的数据流。然后，ControlThread 对象负责读取控制队列，并通过线程安全的消息队列处理处理模块发送的所有消息。



GNSS-SDR 的主方法实例化了一个 ControlThread 类对象，该对象由智能指针管理：

```
auto control_thread = std::make_unique<ControlThread>();
```

如上图所示，该对象的构造函数读取用户在执行接收器时提供的命令行标志，该标志指向包含配置的文本文件：

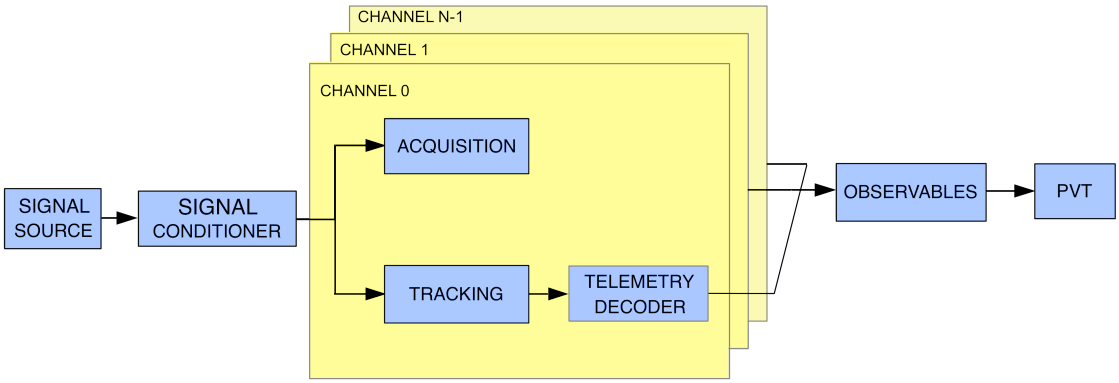
```
$ gnss-sdr --config_file=/path/to/my_receiver.conf
```

然后，当调用 control_thread 对象的 run() 方法时，GNSSFlowgraph 类的一个成员会连接流程图，启动从源到汇的数据流，并不断处理来自控制队列的信息，直到接收器停止。

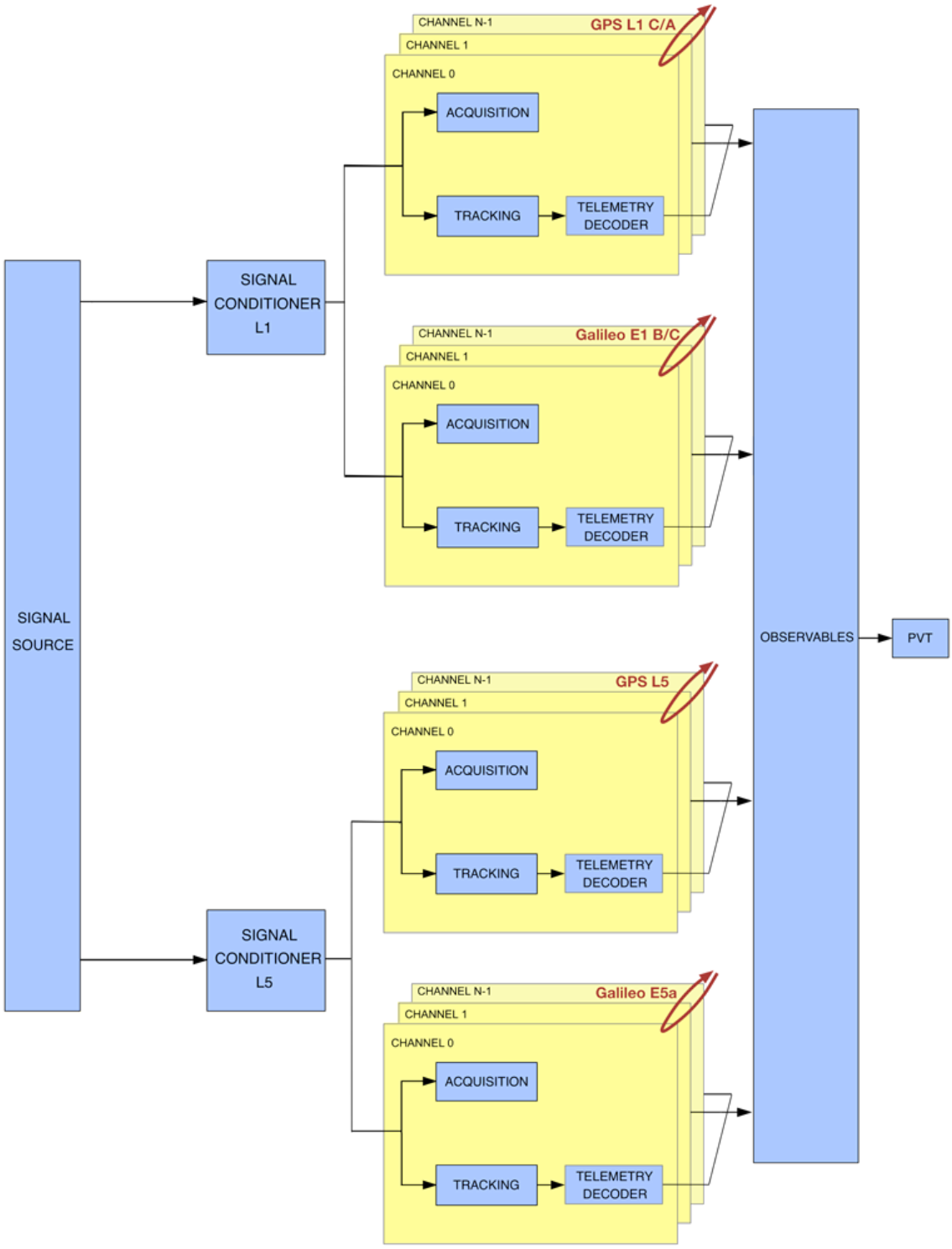
其实际实现摘录如下，其中 flowgraph_ 是 GNSSFlowgraph 类的一个对象：

```
int ControlThread::run()
{
    // Connect the flowgraph
    flowgraph_->connect();

    // Start the flowgraph
    flowgraph_->start();

    // Launch the GNSS assistance process
    assist_GNSS();

    // Main loop to read and process the control messages
    while (flowgraph_->running() && !stop_)
        {
            read_control_messages();
            if (control_messages_ != 0) process_control_messages();
        }
    std::cout << "Stopping GNSS-SDR, please wait!\n";
    flowgraph_->stop();
    flowgraph_->disconnect();
    return 0;
  }
```

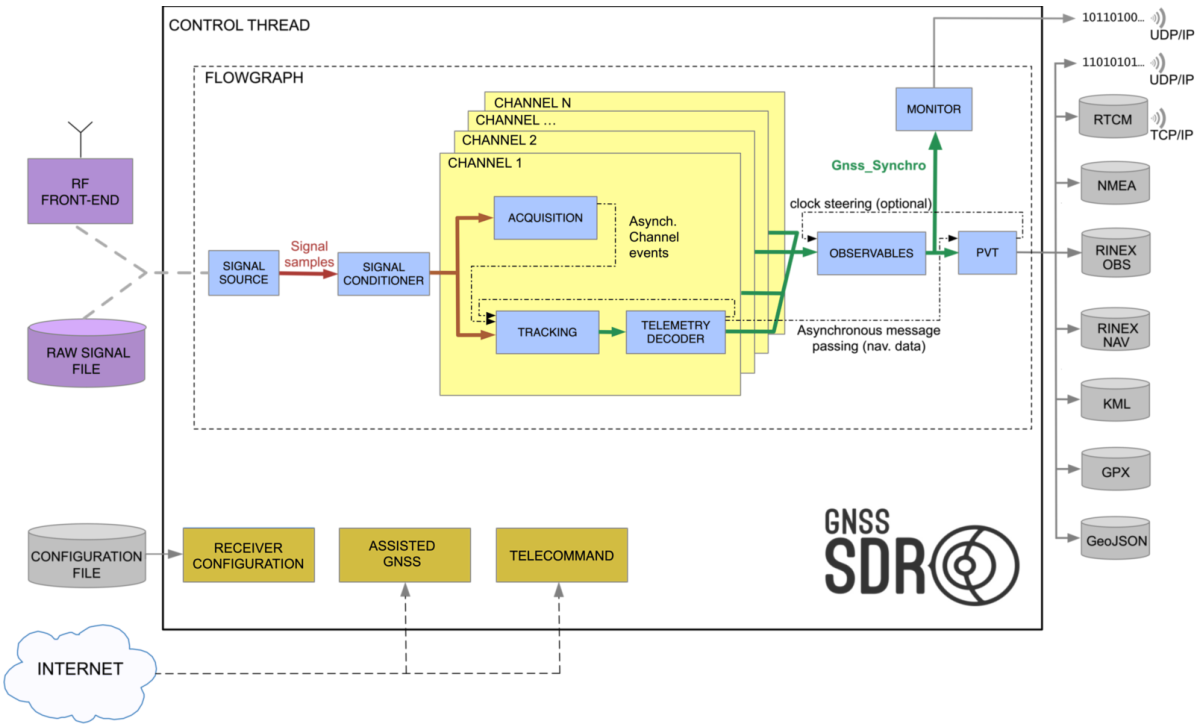
GNSSFlowgraph 类对象将解析配置文件，并要求块工厂提供相应的信号源、信号调节器、通道（每个通道都有自己的采集、跟踪和遥测解码器）、一个观测块（收集所有通道的处理结果）和一个 PVT 块（作为信号汇）：



GNSS-SDR 的配置机制非常灵活，足以允许其他更复杂的流图。例如，您可以用八个通道来接收一个给定信号（如 GPS L1 C/A），并定义另外八个通道来接收伽利略 E1 B/C 信号，从而定义一个多系统接收器。或者也可以将该结构扩展到另一个频段，定义一个多系统、双频 GNSS 接收机的流程图：


SIGNAL SOURCE
SIGNAL CONDITIONER L1
SIGNAL CONDITIONER L5
CHANNEL N-1
CHANNEL 1
CHANNEL 0
GPS L1 C/A
Galileo E1 B/C
GPS L5
Galileo E5a
ACQUISITION
TRACKING
TELEMETRY DECODER
OBSERVABLES
PVT

5、信号处理块



- Global receiver parameters 全局接收机参数
- Signal Source 信号源：信号源是向处理流程图注入连续的 GNSS 信号原始样本流的模块。它是一个抽象概念，涵盖了各种信号源，从文件中存储的样本（各种格式）到射频前端实时传输的多个样本流。
- Signal Conditioner 信号调节器：信号调节器模块负责将采样比特深度调整为可在运行软件接收器的主机上使用的数据类型，还可进行中频到基带的转换、重采样和滤波。
- Data Type Adapter 数据类型适配器：数据类型适配器的作用是对采样流中的数据类型进行转换。
- Input Filter 输入滤波器：输入滤波器模块的作用是从输入信号中过滤噪音和可能的干扰。
- Resampler 重采样：重采样器负责对信号进行重采样，并将其传送到并行处理通道。
- Channels 通道：每个通道都封装了用于信号采集、跟踪和解调单颗卫星导航信息的模块。这些抽象接口可采用不同的算法来处理任何合适的 GNSS 信号。用户可以定义软件接收器实例化的并行通道数量，GNU Radio 采用的每块线程调度器可自动管理现代多核处理器的多任务处理能力。
- Acquisition 捕获：是检测是否存在来自特定 GNSS 卫星的信号，如果检测到信号，可以提码相位和多普勒频移的粗略估计。码相位和多普勒频移但必须足够精确，以便初始化延迟和相位跟踪环路。
- Tracking 跟踪： 跟踪模块的作用是跟踪信号同步参数的变化：码相位、多普勒频移、载波相位。
- Telemetry Decoder 遥测解码器：遥测解码器模块的作用是从全球导航卫星系统卫星广播的导航信息中获取数据比特，获取导航电文数据。
- Observables 生成量测数据：观测数据块的作用是收集来自所有处理通道的同步数据，并从中计算出全球导航卫星系统的基本测量值：伪距、载波相位（或其相位差版本）和多普勒频移（或其伪距率版本）。接下来是对所获得测量结果的数学模型进行描述，并给出物理解释。这些模型将用于计算位置-速度-时间解决方案。所有处理过程都由 Hybrid_Observables 实现管理。
- PVT 定位授时：PVT 模块的作用是计算定位授时，并以适当的格式提供信息，供进一步处理或数据表示。
- Monitor 监控器：监控器模块提供了一个接口，通过 UDP 将接收器的内部数据流传输到本地或远程客户端，从而实时监控接收器的内部状态。

6、依赖的库

- GNU Radio：一个免费开源的软件无线电工具包。反过来，GNU Radio 也需要一些软件依赖，其中一些也被 GNSS-SDR 使用，特别是 Boost、FFTW 、VOLK
- Armadillo：是一个 C++ 线性代数库。它是系统中所有相关库（ LAPACK, BLAS, OpenBlas, ATLAS, others.）的封装器。
- glog：是 Google 日志模块的 C++ 实现。
- gflags：是一个实现命令行标志处理的 C++ 库。
- matio：是一个 MATLAB MAT 文件 I/O 库。
- PugiXML：是一款轻量级、简单而快速的 C++ XML 解析器。
- Protocol Buffers：谷歌语言中立、平台中立、可扩展的结构化数据序列化机制。
- 实现一些基本 SSL 功能的库，如 OpenSSL、GnuTLS、LibreSSL.

二、VScode + WSL2+ Docker desktop 编译调试

直接安装的过程看起来好繁琐、要配置的库太多了。好在作者提供了 Docker。

三、软件使用

本章指导您使用 GNSS-SDR 从零开始，以最简单的配置之一获得定位。信号源将是一个包含原始信号采样的文件（可在互联网上免费获取），因此本程序不需要射频前端或执行软件接收器的强大计算机。唯一的要求是在计算机上安装 GNSS-SDR，并连接互联网下载包含原始信号样本的文件。

1、检查安装情况

打开终端，输入：

```
gnss-sdr --version
```

如果你看见类似于下面，说明安装成功：

```
gnss-sdr version 0.0.18
```

为了利用处理器中的 SIMD 指令集，您需要运行 VOLK 和 VOLK_GNSSSDR 库的剖析器工具（这些操作只需进行一次，但可能需要一段时间）：

```
volk_profile
volk_gnsssdr_profile
```

2、下载原始信号采样文件

可以直接从终端下载：

```
mkdir work
cd work
wget https://sourceforge.net/projects/gnss-
sdr/files/data/2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.tar.gz
tar -zxvf 2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.tar.gz
```

或在浏览器中打开此链接，下载文件并解压缩。这将为您提供 2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat 文件，其中包含 100 秒的原始 GNSS 信号采样，这些采样由一个射频前端采集，其中心频率为 1575.42 兆赫的射频前端采集的原始全球导航卫星系统信号采样。4 MS/秒，采用交错 I&Q 16 位整数格式。

3、配置 GNSS-SDR

复制下图所示的 GNSS-SDR 配置，并将其粘贴到您最喜欢的纯文本编辑器中，注意检查参数 SignalSource.filename 是否实际指向原始数据文件的名称和路径，将文件保存为 my-first-GNSS-SDR-receiver.conf（或其他任意名称）。

```
[GNSS-SDR]

;######### GLOBAL OPTIONS ##################
GNSS-SDR.internal_fs_sps=2000000

;######### SIGNAL_SOURCE CONFIG ############
SignalSource.implementation=File_Signal_Source
SignalSource.filename=/home/your-
username/work/data/2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat
SignalSource.item_type=ishort
SignalSource.sampling_frequency=4000000
SignalSource.samples=0

;######### SIGNAL_CONDITIONER CONFIG ############
SignalConditioner.implementation=Signal_Conditioner
DataTypeAdapter.implementation=Ishort_To_Complex
InputFilter.implementation=Pass_Through
InputFilter.item_type=gr_complex
Resampler.implementation=Direct_Resampler
Resampler.sample_freq_in=4000000
Resampler.sample_freq_out=2000000
Resampler.item_type=gr_complex

;######### CHANNELS GLOBAL CONFIG ############
Channels_1C.count=8
Channels.in_acquisition=8
Channel.signal=1C

;######### ACQUISITION GLOBAL CONFIG ############
Acquisition_1C.implementation=GPS_L1_CA_PCPS_Acquisition
Acquisition_1C.item_type=gr_complex
Acquisition_1C.pfa=0.01
Acquisition_1C.doppler_max=10000
```

```
Acquisition_1C.doppler_step=250
Acquisition_1C.blocking=true

;######### TRACKING GLOBAL CONFIG ############
Tracking_1C.implementation=GPS_L1_CA_DLL_PLL_Tracking
Tracking_1C.item_type=gr_complex
Tracking_1C.pll_bw_hz=40.0;
Tracking_1C.dll_bw_hz=4.0;

;######### TELEMETRY DECODER GPS CONFIG ############
TelemetryDecoder_1C.implementation=GPS_L1_CA_Telemetry_Decoder

;######### OBSERVABLES CONFIG ############
Observables.implementation=Hybrid_Observables

;######### PVT CONFIG ############
PVT.implementation=RTKLIB_PVT
PVT.positioning_mode=Single
PVT.output_rate_ms=100
PVT.display_rate_ms=500
PVT.iono_model=Broadcast
PVT.trop_model=Saastamoinen
PVT.flag_rtcm_server=true
PVT.flag_rtcm_tty_port=false
PVT.rtcm_dump_devname=/dev/pts/1
PVT.rtcm_tcp_port=2101
PVT.rtcm_MT1019_rate_ms=5000
PVT.rtcm_MT1077_rate_ms=1000
PVT.rinex_version=2
```

4、运行 GNSS-SDR

好了，我们来回顾一下。我们有：

- 安装好的 GNSS-SDR。
- 信号源： 一个名为 2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat 的文件，包含 100 秒的 GPS 原始信号样本，由射频前端采集。
- 将 2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat 文件作为信号源的 GPS L1 C/A 接收机的配置文件。

这样，我们就可以运行软件定义的 GPS 接收机了。在终端中输入：

```
gnss-sdr --config_file= 【conf文件路径】
```

您应该看到类似的内容：

```
$ gnss-sdr --config_file=./my-first-GNSS-SDR-receiver.conf
Initializing GNSS-SDR v0.0.18 ... Please wait.
Logging will be done at "/tmp"
Use gnss-sdr --log_dir=/path/to/log to change that.
Processing file /home/your-
username/work/2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat, which contains 1600000000
[bytes]
GNSS signal recorded time to be processed: 99.999 [s]
Starting a TCP/IP server of RTCM messages on port 2101
The TCP/IP server of RTCM messages is up and running. Accepting connections ...
...
```

然后，经过几秒钟检测 GPS 信号并解码其导航信息的某些帧（至少是来自四颗卫星的子帧 1、2 和 3）...

```
...
Current receiver time: 14 s
New GPS NAV message received in channel 3: subframe 1 from satellite GPS PRN 20
(Block IIR)
New GPS NAV message received in channel 0: subframe 1 from satellite GPS PRN 01
(Block IIF)
New GPS NAV message received in channel 4: subframe 1 from satellite GPS PRN 32
(Block IIF)
New GPS NAV message received in channel 2: subframe 1 from satellite GPS PRN 17
(Block IIR-M)
Current receiver time: 15 s
Current receiver time: 16 s
Current receiver time: 17 s
Current receiver time: 18 s
Current receiver time: 19 s
Current receiver time: 20 s
New GPS NAV message received in channel 3: subframe 2 from satellite GPS PRN 02
(Block IIR)
New GPS NAV message received in channel 0: subframe 2 from satellite GPS PRN 01
(Block IIF)
New GPS NAV message received in channel 4: subframe 2 from satellite GPS PRN 32
(Block IIF)
New GPS NAV message received in channel 1: subframe 2 from satellite GPS PRN 11
(Block IIR)
New GPS NAV message received in channel 2: subframe 2 from satellite GPS PRN 17
(Block IIR-M)
Current receiver time: 21 s
Current receiver time: 22 s
Current receiver time: 23 s
Current receiver time: 24 s
Current receiver time: 25 s
Current receiver time: 26 s
New GPS NAV message received in channel 3: subframe 3 from satellite GPS PRN 20
(Block IIR)
New GPS NAV message received in channel 0: subframe 3 from satellite GPS PRN 01
(Block IIF)
New GPS NAV message received in channel 4: subframe 3 from satellite GPS PRN 32
(Block IIF)
New GPS NAV message received in channel 2: subframe 3 from satellite GPS PRN 17
(Block IIR-M)
```

```
New GPS NAV message received in channel 1: subframe 3 from satellite GPS PRN 11
(Block IIR)
First position fix at 2013-Apr-04 06:23:31.740000 UTC is Lat = 41.2749 [deg],
Long = 1.98754 [deg], Height= 100.795 [m]
Position at 2013-Apr-04 06:23:32.000000 UTC using 4 observations is Lat =
41.274888307 [deg], Long = 1.987581872 [deg], Height = 86.928 [m]
Position at 2013-Apr-04 06:23:32.500000 UTC using 4 observations is Lat =
41.274964746 [deg], Long = 1.987510141 [deg], Height = 90.557 [m]
Current receiver time: 27 s
Position at 2013-Apr-04 06:23:33.000000 UTC using 4 observations is Lat =
41.274921885 [deg], Long = 1.987605767 [deg], Height = 73.365 [m]
Position at 2013-Apr-04 06:23:33.500000 UTC using 4 observations is Lat =
41.274866502 [deg], Long = 1.987553835 [deg], Height = 83.313 [m]
Current receiver time: 28 s
Position at 2013-Apr-04 06:23:34.000000 UTC using 4 observations is Lat =
41.274904024 [deg], Long = 1.987612510 [deg], Height = 87.615 [m]
Position at 2013-Apr-04 06:23:34.500000 UTC using 4 observations is Lat =
41.274877911 [deg], Long = 1.987553312 [deg], Height = 81.405 [m]
Current receiver time: 29 s
Position at 2013-Apr-04 06:23:35.000000 UTC using 4 observations is Lat =
41.274916750 [deg], Long = 1.987576650 [deg], Height = 108.288 [m]
Position at 2013-Apr-04 06:23:35.500000 UTC using 4 observations is Lat =
41.274803167 [deg], Long = 1.987562527 [deg], Height = 90.144 [m]
Current receiver time: 30 s
Position at 2013-Apr-04 06:23:36.000000 UTC using 4 observations is Lat =
41.275044618 [deg], Long = 1.987619037 [deg], Height = 102.346 [m]
Position at 2013-Apr-04 06:23:36.500000 UTC using 4 observations is Lat =
41.274878468 [deg], Long = 1.987557377 [deg], Height = 102.764 [m]
Current receiver time: 31 s
Position at 2013-Apr-04 06:23:37.000000 UTC using 4 observations is Lat =
41.274995336 [deg], Long = 1.987554843 [deg], Height = 113.653 [m]
Position at 2013-Apr-04 06:23:37.500000 UTC using 4 observations is Lat =
41.274951615 [deg], Long = 1.987600581 [deg], Height = 92.064 [m]
Current receiver time: 32 s
...
```

如果您看到与此类似的内容……太好了！您正在使用开源软件定义的 GPS 接收机获取定位信息！

```
...
Current receiver time: 1 min 40 s
Position at 2013-Apr-04 06:24:45.500000 UTC using 5 observations is Lat =
41.274821613 [deg], Long = 1.987629659 [deg], Height = 69.292 [m]
Position at 2013-Apr-04 06:24:46.000000 UTC using 5 observations is Lat =
41.274817101 [deg], Long = 1.987576895 [deg], Height = 43.517 [m]
Position at 2013-Apr-04 06:24:46.500000 UTC using 5 observations is Lat =
41.274830209 [deg], Long = 1.987583859 [deg], Height = 54.475 [m]
Stopping GNSS-SDR, please wait!
Total GNSS-SDR run time: 37.106698 [seconds]
GNSS-SDR program ended.
Stopping TCP/IP server on port 2101
$
```

现在，您可以检查调用 GNSS-SDR 的文件夹中的处理输出结果：

- 一个 `.kml` 文件

- 一个 `.geojson` 文件
- 一个 `.gpx` 文件
- 一个 `.nmea` 文件
- OBS 和 NAV 文件

四、配置文件