

jamesshao8



10

23

80

主题

帖子

积分

注册会员



积分

80

发消息



10

23

80

主题

帖子

积分

注册会员



积分

80

发消息

360

无线电安全研究院

件无线电 [LimeSDR] Made Simple 7 其它环境（下）

[LimeSDR] Made Simple 7 其它环境（下）

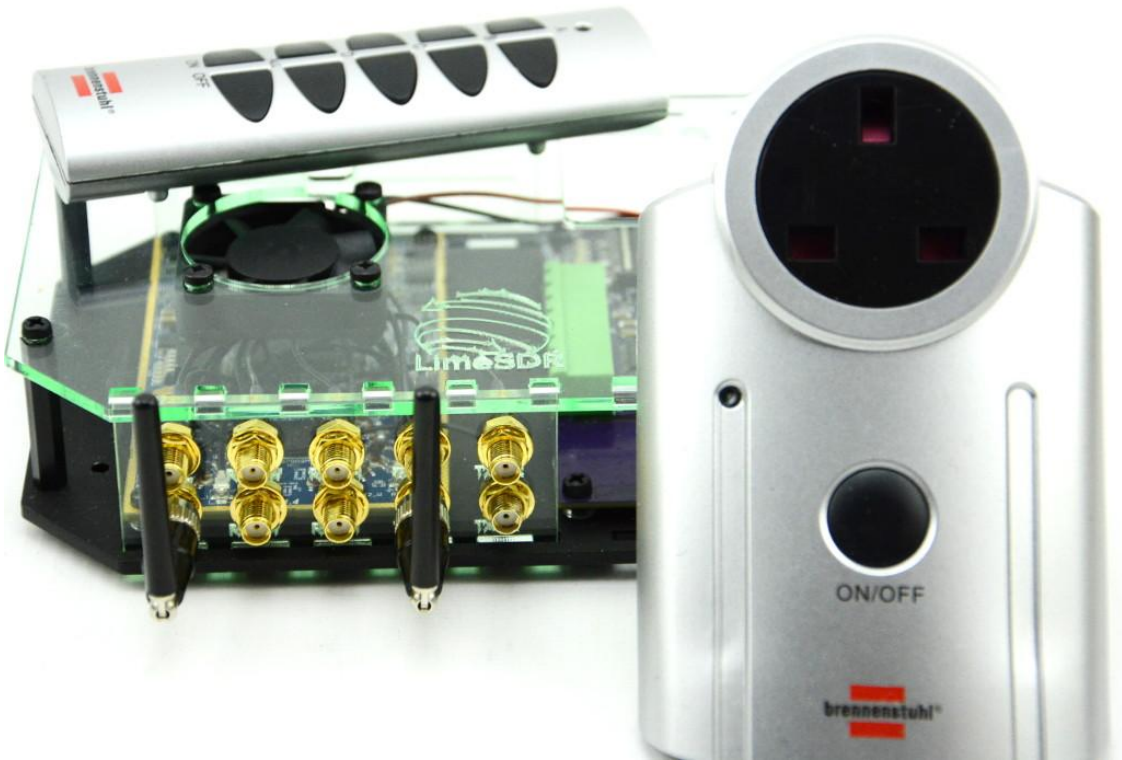
[复制链接]

 发表于 2018-12-22 11:41:06 | 只看该作者

楼主

电梯直达



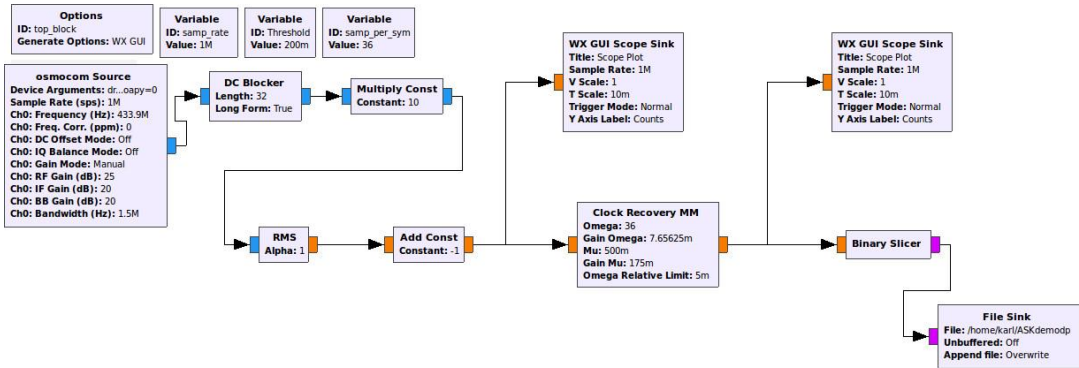


在GNU Radio中接收和发射ASK信号

这是第七篇教程。我们继续上一次的内容，我们会仿制出一个最简单的无线电设备。上一篇文章我们可以接收比特流，但是每个符号对应的比特数是错误的。这篇文章，我们计划改进ASK接收机，并且设计一个发射机，重放数据。

要实现这个目标我们需要同时有发射机（类似遥控器）和接收设备。我们手头正好有Brennenstuhl Primera-Line遥控插座。出于好奇，我们看了它的遥控器信号，它们也是ASK而且还是OOK，因此我们的流图几乎不需要大概就能适配。

我们上次做到这里



这是我们上次的流图，如果你没看过这篇文章最好看一下。在我们深入之前，最好先回忆一下这个流图做了什么，以及如何实现的。

osmocom source 对应于LimeSDR

DC blocker 用来对接收到的信号去除直流分量，这个模块使得示波器不会显示出大的直流分量

jamesshao8




10主题

23帖子

80积分

注册会员



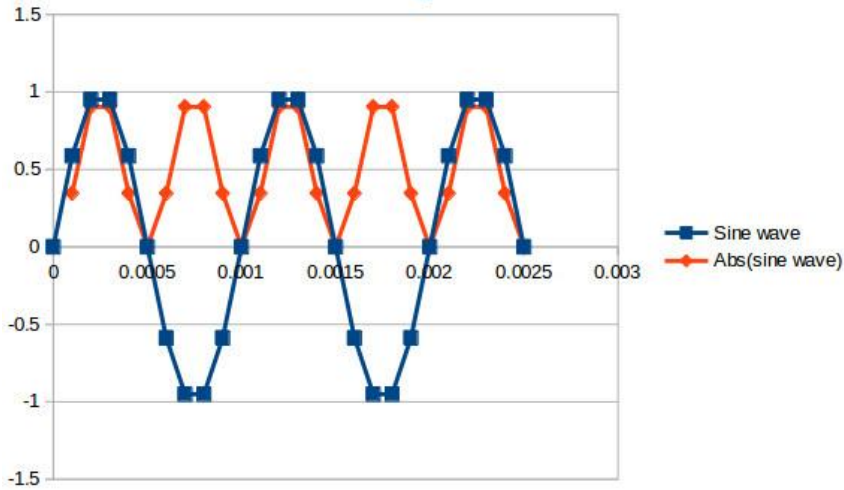
积分80

发消息

Multiply const 模块产生了一个固定的数字增益，大小是10  
RMS 模块把信号从正弦波的形式转换为 **社区** 形式  
Add const 增加了直流偏置，把信号移动下来，使其穿过零点，这是Clock recovery模块要求的  
Clock recovery模块和Binary slicer模块把数据恢复出来并记录到文件中

AM解调  
我们现在用RMS做AM解调，做得还不错，但是原理是什么？

如果我们看一下RMS模块内部，它很简单：Out = sqrt(Avg)，其中Avg = (1-Alpha)\*Avg + Alpha\*abs(in)^2，我们这里设置Alpha=1，那么Avg = abs(in)^2。



我们看一下正弦波的绝对值，我们会得到上述波形，然后把这个波形和原始波形以10个采样点取平均，我们会得到：  
正弦波=0  
Abs(正弦波) =0.5

现在你应该可以清楚知道RMS模块为什么能解调AM信号了，因为当载波出现时RMS模块输出一个大于0的值，没有载波时输出等于0。

Clock Recovery模块

上一篇文章中我们发现，用程序来解调比特比直接观察波形更难。因为我们必须根据比特的波特率在正确的时间点采样，并解码出比特流。我们使用了clock recovery模块，我们上次用几个字节来表示一个比特。我们上次说了这是因为我们没有选择一个正确的Omega值。Omega值代表每个比特对应的采样点数量（等效于波特率）。

我们如何知道每个比特对应多少采样点？

首先我们需要知道每个比特对应的长度。有很多方法，最简单的是找datasheet。我们这次无法找到。或者用URH程序可以帮助我们做这个事情，但是只用GNU Radio也可以实现。

有些设备的波特率会变化，这样难度比较高。比如之前在eBay买的遥控钥匙就是这样，那么就会增加难度。所以现在我们换成了另一个遥控电源开关。

Brennenstuhl电源开关就没有这个问题。我们只需要在RMS模块后接一个示波器，然后计算最小的脉冲的长度，就可以得到准确的Omega值。我们算过了上升沿和下降沿的时间间隔大约是0.51ms。我们的采样率是1MSps/s，因此每1x10^-6秒就有一个采样点。那么我们可以得到0.51ms对应于510个采样点，这个510就是我们的新的Omega值。

在我们把510输入到clock recovery模块中前，我们要知道，把这么大的数字作为Omega不太好。最好先做降采样，然后再做clock recovery。我们增加一个Rational Resampler，然后降采样率设置为10，这样每个符号对应的采样点数量就降为了51，这个数字设置为Omega更好。

比特流是关键

现在我们的解码器就可以正常解调UHF电源开关的信号了，我们可以解调开关按钮各自的信号，我们就得到下面两个比特流：

每次传输（包含重复传输）都会有这个前导码: 11111100000000000000

关闭按钮的载荷 payload:110100110110110110100110100110110110100110110100100110100110100100111111

打开按钮的载荷 payload: 110100110110110110110110100100110110110100100110100110100110100110100100111111

开关按钮的区别很小。根据这种样式，很有可能这个比特流使用了某种形式的反码算法，来实现编码增益并对抗干扰。

构建发射流图

本主题由 mobier 于 2018-12-25 13:53 设置高亮



jamesshao8

10

23

80

主题

帖子

积分

注册会员

积分

80

发消息

回复

社区

沙发

楼主 | 发表于 2018-12-22 11:42:21 | 只看该作者

了解怎样的数据在开启和关闭我们的Brennunstuhl很不错，但是如果无法发射这样的信号那也美多少用处。我们现在创建一个433MHz的OOK发射机，并且重复我们前面观察到的序列。

流图中首先应该有我们观测到的数据。就像接收和解码数据时一样，有各种方法来存储这个数据。为了方便，我们这里使用了Vector Srouce，而不是文件存储或者message box。把我们的数据编码为vector，它看上去是这样的：(1,1,0,1...中间省略...,0,0,1)。我们还加入了15个0作为间隔，这样每次重复的数据间会有个暂停。

这个vector source会不停循环播放比特流，但是播放的速率是流图设置的采样率，在我们这里是1MSps/s，这样它的速度会比我们需要的快510倍。我们可以用一个超大的vector代替，每个比特都重复510次，比如1替换为1111...中间502个...111。但是这样的实现方式不优雅。更好的方式是使用Repeat Interpolation模块。这样它会根据差值长度来重复每个比特（在我们这里就是重复510个采样点），这样的效果实际上是一样的。接下来，我们可以使用Throttle模块来设置比特率为1MSps/s。这个模块会对流过的采样点的数量进行限制，把我们的输入速度降下来，达到目标波特率。这样我们就有了一个方波，波特率符合我们的要求。然而这是一个数字信号，没有载波，无法传输。

首先我们需要一个载波，使用一个433.91MHz的正弦波应该够了。我们用Signal Source来创建。OOK信号的好处是，它们要不就是开启（值是1），要不就是关闭（值是0），那么我们只需要把载波乘上去就完成编码了。

最好增加一个门，我们可以用GUI Chooser来选择0和1，这样用这个来控制另一个乘法模块，来控制信号是否发射，我们还可以在其中增加数字增益。最终我们可以用一个按钮来实时控制。

最后一级是Osmocom Sink，我们设置为正确的参数后就好了：

- Device Arguments: "driver=lime,soapy=0"
- Frequency: 433.91e6
- Bandwidth: 5.1e6 我们必须这样设置，否则GNU Radio会自动填写一个错误的值，造成流图无法生成。

在按下运行按钮前，必须注意尽管433MHz是免费频段，但是你还是限制发射功率，把增益降低或者用一个小天线就可以满足这个要求。如果你不确定，可以先用一个较低增益试试。

经常有人会觉得更多增益会更好。不是所有情况下都这样的，比如一个接收机正好在发射机旁边时，太大的增益会使接收机饱和，接收效果反而不好。

最后

我们希望这篇文章展现了GNU Radio和LimeSDR的强大之处，我们自己实现了一个遥控器，用来控制我们的电源开关，只需要这么一个简单的流图就能实现。这个对于所有OOK设备都是如此，包括你邻居家的门铃，所以使用这个流图的时候需要你自己负责。对于433MHz频段，我们暂时就讲到这里了，但是我们将来可能还会回来讲FSK和PSK，我们可能会尝试一些别的设备。

发新帖

返回列表