YunSDR-Y5x0

Matlab 开发指南

Rev. 2.0

北京威视锐科技有限公司

威视锐®
V3 Technology,.Ltd.

修订记录

版本	修订日期	修订内容
1.0	2018 年 08 月 12 日	初始版本
1.1	2019 年 01 月 16 日	更新到 C API
1.2	2020 年 1 月	更新产品图片
2.0	2020 年 6 月	更新 4 通道 Y5x0 平台

关于威视锐科技

北京威视锐科技有限公司专注于软件定义（SDx）系列的研发与生产，面向无线通信、视频视觉和测试测量领域提供完整的解决方案，可应用于科研教学与产品研发。威视锐与微软研究院联合开发的 SoraSDR 软件无线电平台、YunSDR 软件无线电平台已经成为世界上知名大学和科研机构开展无线通信研究的首选平台，也是学术研究领域全球唯一的基于 x86 和 Windows 系统的宽带软件无线电平台，目前已经有超过 20 多个国家的 300 多个用户在 Sora 平台上开发无线局域网、移动通信、大规模 MIMO 等相关领域的应用。作为全球最大的可编程器件公司 Xilinx 的全球认证合作伙伴、授权培训合作伙伴和大学计划合作伙伴，威视锐科技提供基于 Xilinx FPGA/SoC 全方位解决方案。威视锐同时也是全球领先的高性能模拟器件厂家 ANALOG DEVICES 公司的第三方和大学计划合作伙伴，提供基于 ADI 的高性能射频收发器，转换器和传感器开发套件。特别是无线通信、物联网、视觉图像处理和数字信号处理的创新型实验室建设，威视锐可以提供完整的解决方案和技术支持服务。

多年以来，威视锐坚持“Innovation for Research”的发展理念，与国内众多知名高校建立合作关系，帮助专家、学者和研发工程师创新的理念变成现实和产品。对于产业界客户，威视锐提供严格验证的核心模块、智能便携的测量仪器以及定制化的设计服务来加快产品研发周期。

目 录

一、文档概述

本文档适用于使用加速卡和 YunSDR-Y5x0 4 通道 SDR 硬件平台。使用 Matlab 软件进 IQ 数据收发。

二、软件环境配置

在使用相应设备之前，需要对 PC 的环境进行配置。为保证顺利使用，请按照下列要求进行配置。

- **PC 操作系统：**

  Windows7 x64

  Windows10 x64

  Windows Server x64

- **Matlab 软件：**2016a 及以上；

  C 编译库：TDM-GCC，或其他 matlab 所支持的 C 编译器

  详细的编译库支持请参见 https://ww2.mathworks.cn/support/compilers.html

  环境配置完毕之后，打开 Matlab 的 Command Window，输入 mex -setup 命令，显示如下：

```
>> mex -setup
MEX configured to use 'MinGW64 Compiler (C)' for C language compilation.
Warning: The MATLAB C and Fortran API has changed to support MATLAB
     variables with more than 2^32-1 elements. In the near future
     you will be required to update your code to utilize the
     new API. You can find more information about this at:
     http://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-b

To choose a different C compiler, select one from the following:
MinGW64 Compiler (C)  mex -setup:C:\Users\hans\AppData\Roaming\MathWorks\MATLAB\R2016a\me
Microsoft Visual C++ 2013 Professional (C)  mex -setup:'C:\Program Files\MATLAB\R2016a\bi

To choose a different language, select one from the following:
 mex -setup C++
 mex -setup FORTRAN
```

图 1　Matlab 下的 C++编译器安装

表明 Matlab 可以找到 C++编译器(MinGW64)，配置成功（若提示找不到编译器，请重启电脑或检查相关软件的安装路径），然后单击蓝色字体：mex -setup C++，进行编译器切换。详细安装流程参考 2.1 TDM-GCC 编译器安装与设置。

（一）TDM-GCC 编译器安装与设置

不同的 Matlab 推荐使用的 TDM-GCC 版本不同，用户可以根据安装的 Matlab 版本参考上面链接选取合适的 GCC 版本。例如 Matlab 2016a 配套的 GCC 编译器版本是 tdm64-gcc-4.9.2.exe。安装好后需要在环境变量中做如下设置：

找到“我的电脑”图标，右击点击“属性”，找到“高级系统设置”：

图 2 环境变量 1

点击“高级系统设置”，出现如下对话框，找到“环境变量(N)”选项，点击新建：

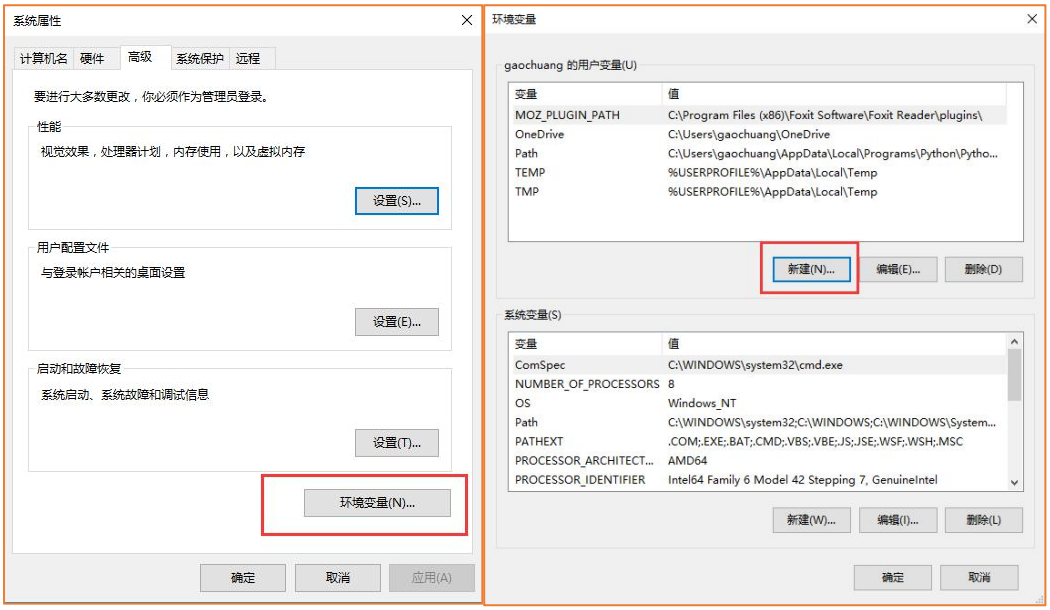

图 3 新建环境变量 1

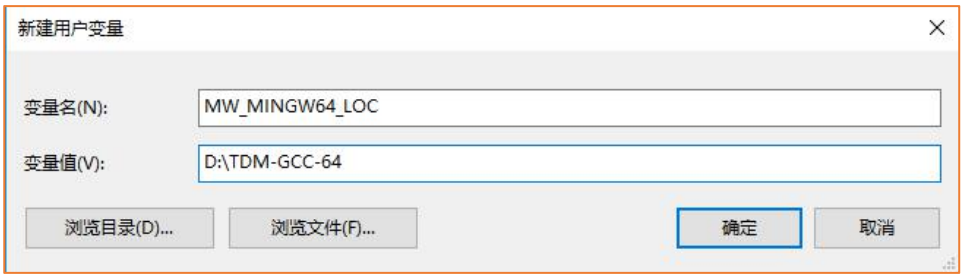

图 4 新建环境变量 2

注意：变量值填写 TDM-GCC 编译器的安装路径。

打开 MATLAB2016，输入以下命令：mex –setup，出现如下信息说明环境配置成功。

```
>> mex -setup
MEX configured to use 'MinGW64 Compiler (C)' for C language compilation.
Warning: The MATLAB C and Fortran API has changed to support MATLAB
     variables with more than 2^32-1 elements. In the near future
     you will be required to update your code to utilize the
     new API. You can find more information about this at:
     http://www.mathworks.com/help/matlab/matlab_external/upgrading-mex-files-to-use-64-b

To choose a different C compiler, select one from the following:
MinGW64 Compiler (C)  mex -setup:C:\Users\hans\AppData\Roaming\MathWorks\MATLAB\R2016a\me
Microsoft Visual C++ 2013 Professional (C)  mex -setup:'C:\Program Files\MATLAB\R2016a\bi

To choose a different language, select one from the following:
 mex -setup C++
 mex -setup FORTRAN
```

图 5　选择 mex

三、Matlab 开发流程及配套文件说明

本文档介绍的 Matlab 例程，通过 Matlab 的 API 控制 YunSDR 的射频工作参数、控制 YunSDR 发送和接收 IQ 数据。PC 通过高速总线与 YunSDR 相连，YunSDR 设备接收到 PC 发送到的一帧数据后，将数据缓存在 DDR 中，并可配置 YunSDR 的发送工作模式进入 txcyclic 循环模式（类似信号源的功能），TX 端口会反复发送这帧 IQ 数据；接收端收到数据后，YunSDR 将接收的数据搬移至 DDR，然后再通过通信总线发送至 PC，至此组成了一个无线模拟仿真系统。

由于 PC 和设备的接口通过通信总线实现，所以可在 PC 端通过 Matlab 等工具将数据按照用户自身需求进行调制，最终将调制后的数据通过通信总线下发至 YunSDR 设备；而接收端同样可以用 Matlab 将所接收的数据按照需求进行解调，基本流程参考下图。

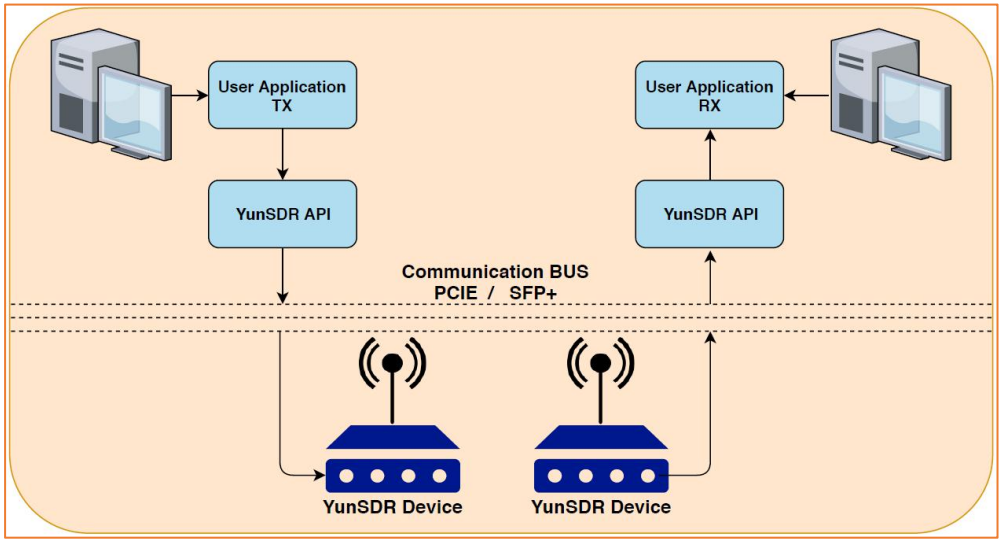

图 6　Matlab 软件功能流程

- 支持通过 Matlab 配置射频参数
- 发送端支持发送 Matlab 产生的 IQ 数据
- 通过 Matlab 可获取射频前端采集到的 IQ 数据
- 用户可定制自己的收发仿真系统

本文档所介绍的产品配套的 Matlab 程序文件包含用户交互文件及底层驱动文件，如下表所示：

文件	类型	说明
yunsdr.m	顶层文件	实现单板卡自发自收
yunsdr_tx.m	顶层文件	将 IQ 数据送给设备循环发送
yunsdr_rx.m	顶层文件	接收指定长度和通道数量的 IQ 数据
gen_rxbuf.m	底层函数	RX 接收数据驱动
gen_txbuf.m	底层函数	TX 发送数据驱动
gen_trxbuf.m	底层函数	回环应用中的收发驱动
tone_plot.m	底层函数	画图
yunsdr_api_ss.h		
riffa.dll	DLL 驱动文件	
libyunsdr_ss.dll	DLL 驱动文件	

表 1　文件列表

四、参考例程说明

本文档提供基于 Matlab 环境的单音信号收发测试，在此基础上用户可以根据单音信号的数据格式，二次开发行程自定义的仿真系统。

（一）发送模式

yunsdr_tx.m，用于验证设备发送。

1. 测试类型选择

通过以下命令设置测试模式类型，分别为 tone 单音、LTE FDD，source 变量定义测试类型，即要发送的 IQ 数据类型。

```
source='tone';% tone lte_new
```

Tone 信号单音周期 32 个采样点，单帧数据长度 3200 个样点。最后将单音信号复制 4 列，表示 4 个通道发送数据，txdata 矩阵 3200x8

```
for i=1:4
     txdata(:,i)=[repmat(c,1000,1)];%zeros(400,1);
end
```

LTE 信号，20MHz 带宽，发送数据长度 10ms。最后将 LTE 信号复制 4 列，表示 4 个通道发送数据，txdata 矩阵 307200x8。

2. 加载动态链接库

通过以下命令载入动态库文件。

```
if not(libisloaded('libyunsdr_ss'))
    [notfound,warnings] = loadlibrary('libyunsdr_ss','yunsdr_api_ss.h');
end
dptr = libpointer('yunsdr_device_descriptor');
devstring = libpointer('cstring');
devstring.Value = 'pciex:0,format:s16';
dptr = calllib('libyunsdr_ss', 'yunsdr_open_device', devstring);
if isNull(dptr)
    disp 'open yunsdr failed!';
    return;
end
```

3. 设置射频参数

可以根据需要设定射频的频点采样率等参数，详见 rf_init.m，可以选择需要配置的参数粘贴到 yunsdr_tx.m 的%% add rf config 下方

```
%% rf1 config
ret=calllib('libyunsdr_ss','yunsdr_set_tx_sampling_freq',dptr,0,uint32(122.88e6));
ret=calllib('libyunsdr_ss','yunsdr_set_tx_lo_freq',dptr,0,uint64(2500e6));
ret=calllib('libyunsdr_ss','yunsdr_set_rx_lo_freq',dptr,0,uint64(2500e6));
ret=calllib('libyunsdr_ss','yunsdr_set_tx1_attenuation',dptr,0,uint32(30e3));
ret=calllib('libyunsdr_ss','yunsdr_set_tx2_attenuation',dptr,0,uint32(30e3));
ret=calllib('libyunsdr_ss','yunsdr_set_rx1_rf_gain',dptr,0,uint32(5));
ret=calllib('libyunsdr_ss','yunsdr_set_rx2_rf_gain',dptr,0,uint32(5));
%% rf2 config
ret=calllib('libyunsdr_ss','yunsdr_set_tx_sampling_freq',dptr,1,uint32(122.88e6));
ret=calllib('libyunsdr_ss','yunsdr_set_tx_lo_freq',dptr,1,uint64(2500e6));
ret=calllib('libyunsdr_ss','yunsdr_set_rx_lo_freq',dptr,1,uint64(2500e6));
ret=calllib('libyunsdr_ss','yunsdr_set_tx1_attenuation',dptr,1,uint32(30e3));
ret=calllib('libyunsdr_ss','yunsdr_set_tx2_attenuation',dptr,1,uint32(30e3));
ret=calllib('libyunsdr_ss','yunsdr_set_rx1_rf_gain',dptr,1,uint32(5));
```

```
ret=calllib('libyunsdr_ss','yunsdr_set_rx2_rf_gain',dptr,1,uint32(5));
```

4. 获取采样率复位时间戳

Y5x0 默认采样率是 122.88MHz，采样率可以选择 122.88MHz 除以 4~128。时间戳是设备的定时机制，采用 64 位计数器以采样率为频率进行计数，接收和发送都是定位在这个时间戳的基础上。只有时间戳启动后设备才开始工作。

```
%% get samplerate
ret=calllib('libyunsdr_ss','yunsdr_get_tx_sampling_freq',dptr,0,value32);
samplerate=double(value32.Value);
%% set timpstamp start
ret=calllib('libyunsdr_ss','yunsdr_enable_timestamp',dptr,0,0);
ret=calllib('libyunsdr_ss','yunsdr_enable_timestamp',dptr,0,1);
```

5. 生成发送缓冲区

gen_txbuf.m 函数中对 txdata 数据进行量化，传输时将数据量化到 16bit 方便缓存与处理。量化后将 n 个通道的矩阵，按照 1 2 3 4.....的通道顺序排成一列，放入到生成的 int16ptr 缓冲区中。

```
function [tx_buf,length_tx,channel]=gen_txbuf(txdata)
length_tx=size(txdata,1);
channel=2^size(txdata,2)-1;
for i=1:size(txdata,2)
  c1=max(max([abs(real(txdata(:,i))),abs(imag(txdata(:,i)))]));
  if(c1>0)
    index=2000/c1;
  else
    index=0;
  end
  txdata1(:,i)=round(txdata(:,i).*index)*16;
end
txdata_s=txdata1(:);
txdatai=real(txdata_s);
txdataq=imag(txdata_s);
txdatam=zeros(length(txdatai)*2,1);
txdatam(1:2:end)=txdatai;
txdatam(2:2:end)=txdataq;
txdatamu=txdatam+(txdatam<0)*65536;
tx_buf = libpointer('voidPtrPtr');
tx_buf.Value = libpointer('int16Ptr', txdatamu);
end
```

6. 开始发送

Y5x0 支持将待发送的 IQ 数据缓存到自身的 ddr4 缓存中，循环发送，模拟作信号源的功能。基于时戳定时的原理是：

- 首先获取设备的时间戳
- 将所有通道开始发送 IQ 数据的时刻 ts
- 指定发送时刻 ts+1s(采样率)
- 调用发送函数将 IQ 数据送给设备
- 设备收到 IQ 数据由于还没有到发送时刻，所以将 IQ 数据缓存
- 等待到达发送时刻所有通道一起启动 TX
- 判断上位机配置了 tx'cyclic 指令，设备循环启动 TX 发送，循环播出 TX 序列，可以在频谱以上观测发送的信号。

```
%% send data in txcyclic mode
ret=calllib('libyunsdr_ss','yunsdr_tx_cyclic_enable',dptr,0,0);% reset txcyclic
ret=calllib('libyunsdr_ss','yunsdr_tx_cyclic_enable',dptr,0,1);
ret=calllib('libyunsdr_ss','yunsdr_read_timestamp',dptr,0,value64);
ts3=value64.Value+samplerate;
nwrite = calllib('libyunsdr_ss', 'yunsdr_write_samples_multiport_Matlab',  ...
  dptr, tx_buf, length_tx, channel, ts3, 0);
ret=calllib('libyunsdr_ss','yunsdr_tx_cyclic_enable',dptr,0,3);% start txcyclic
```

7. 结束循环发送

如果需要结束循环发送，需要复位发送函数即可

```
ret=calllib('libyunsdr_ss','yunsdr_tx_cyclic_enable',dptr,0,0);% reset txcyclic
```

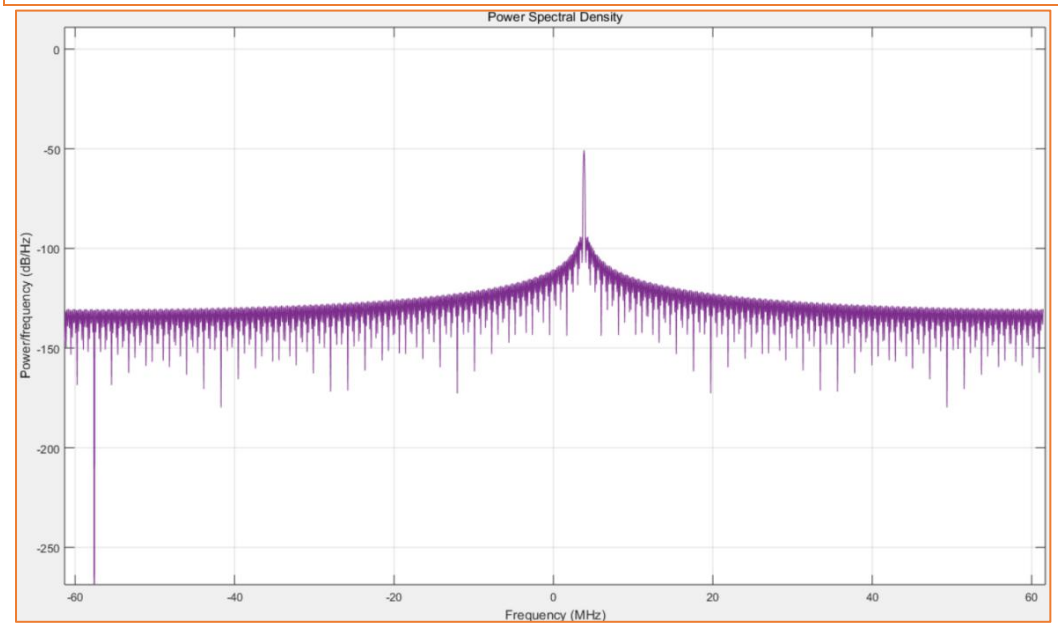

图 7　单音信号发送

（二）接收模式

yunsdr_rx.m，用于验证设备接收。接收与发送的流程基本一致，将采集到数据送给matlab。

首先使能加载动态库；

其次可以根据需要设定射频参数，和 4.1.2 节一样。

之后获取采样率复位时间戳，接收端和发送端共用采样率。接收端和发送端以同一个时间戳为基准。

1. 设定接收长度和接收通道

可以选择使能接收通道的编号，以 0~F 通道掩码表示 4 个通道，最低 bit 代表通道 0，最高 bit 代表通道 3。所有通道均使能是 F。

接收长度，可以设置 1ms~100ms 的数据量，步进 1ms。

```
rxch=hex2dec('f');% mask of 8 channel of each bit,ff is 8 chan
rxlength=samplerate/1000;% 1ms to 100ms
```

2. 启动接收函数

调用接收函数后，会自动在当前目录存储符合通道数据量的数据文件，如果使能 4 个通道会存储 4 个 IQ 数据文件。接收函数也会返回 n 列的 IQ 复数序列。

```
[rxdata,rx_time]=gen_rxbuf(dptr,rxlength,rxch,0);
```

3. 数据处理

接收端运行效果如下图所示：

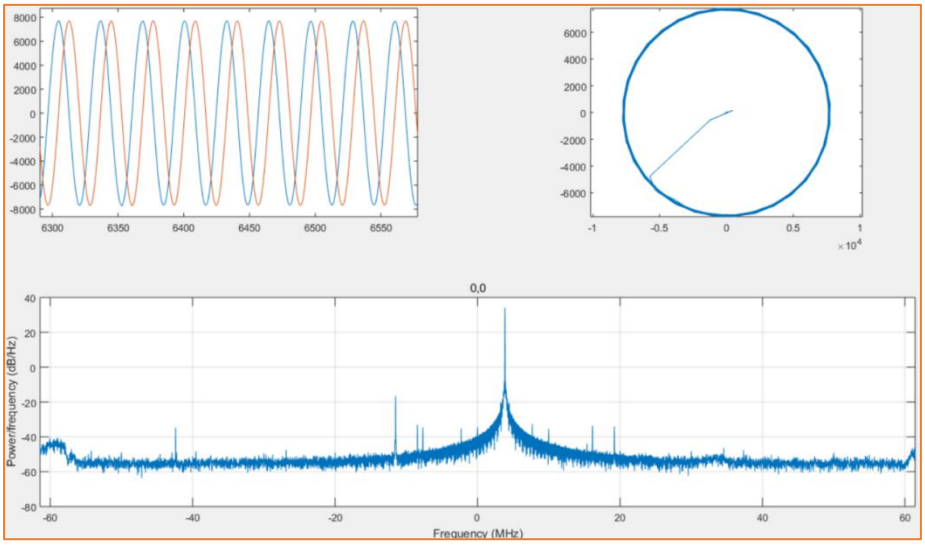

图 8 单音信号自收发运行效果

（三）收发回环

收发回环运行 yunsdr.m 文件，它集成了接收和发送的功能。可以在单独设备上验收接收和发送回环。此外设定的收发时戳是已知的，所以可以根据接收到的样点时刻计算收发回环的群延时。

收发回环的时戳和频点设置方法与 4.1 4.2 没有区别，主要的区别在于 gen_trxbuf 这个函数。它的原理采用了经典的 SDR 软件时隙控制的方式：

- 接收 1ms
- 接收 1ms，取得接收数据中的时间戳 TS2
- 指定发送时刻 TS2+10ms 发送 1ms 数据
- 继续连续接收 18ms 的数据

处理数据的方法，一共接收了 20ms 的数据，分为 20 个时隙 slot，在第 12 个 slot 发送数据。所以应该在地 12 个 slot 上可以接收端接收到发送的 IQ 数据。

理想的情况应该在地 12 个 slot 的第一个点就可以收到发送的 IQ 数据，但是由于链路和 FPGA 内部的滤波器造成了延时，这样可以从接收的数据中找到相对于第 12 个 slot 的起始点 t1，则 t1 就是收发的链路延时。

附录：动态库函数列表

DLLEXPORT YUNSDR_DESCRIPTOR *yunsdr_open_device(const char *url);

打开设备

DLLEXPORT int32_t yunsdr_close_device(YUNSDR_DESCRIPTOR *yunsdr);

关闭设备

DLLEXPORT int32_t yunsdr_get_sampling_freq_range(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *sampling_freq_hz_max, uint32_t *sampling_freq_hz_min);

获取支持的采样率范围，Y5x0 默认 122880000Hz，可配置范围 122880000 除以 1,4~128。rfid=0

DLLEXPORT int32_t yunsdr_get_rx_gain_range(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *gain_db_max, uint32_t *gain_db_min);

获取支持的接收增益设置范围，Y5x0 接收增益的调整范围 0~60。rfid=0

DLLEXPORT int32_t yunsdr_get_tx_gain_range(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *gain_db_max, uint32_t *gain_db_min);

获取支持的发送增益设置范围，Y5x0 发送衰减调整范围 0~41950mdB。rfid=0

DLLEXPORT int32_t yunsdr_get_rx_freq_range(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t *lo_freq_hz_max, uint64_t *lo_freq_hz_min);

获取支持的接收频点设置范围，Y5x0 75e6Hz ~ 5900e6Hz。rfid=0

DLLEXPORT int32_t yunsdr_get_tx_freq_range(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t *lo_freq_hz_max, uint64_t *lo_freq_hz_min);

获取支持的发送频点设置范围，Y5x0 75e6Hz ~ 5900e6Hz。rfid=0

DLLEXPORT int32_t yunsdr_get_tx_lo_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t *lo_freq_hz);

获取当前发送频点，rfid0~1 代表 01 23 四组通道的频点

DLLEXPORT int32_t yunsdr_get_tx_sampling_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *sampling_freq_hz);

获取当前发送采样率，rfid=0。

DLLEXPORT int32_t yunsdr_get_tx_rf_bandwidth (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *bandwidth_hz);

获取发送带宽，Y5x0 固定返回 100000000Hz

DLLEXPORT int32_t yunsdr_get_tx1_attenuation (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *attenuation_mdb);

获取发送衰减,Y5x0 固定返回 0。rfid0~1 表示 01 23 四组射频通道，tx1 表示每个射频通道的第一路

DLLEXPORT int32_t yunsdr_get_tx2_attenuation (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *attenuation_mdb);

获取发送衰减,Y5x0 固定返回 0。rfid0~1 表示 01 23 四组射频通道，tx2 表示每个射频通道的第二路

DLLEXPORT int32_t yunsdr_get_rx_lo_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t *lo_freq_hz);

获取当前接收频点，rfid 0~1 代表 01 23 四组通道的频点

DLLEXPORT int32_t yunsdr_get_rx_rf_bandwidth (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t *bandwidth_hz);

获取接收带宽,Y5x0 默认 122880000Hz,可配置范围 122880000 除以 1,4~128。rfid=0

DLLEXPORT int32_t yunsdr_get_rx1_gain_control_mode (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,RF_GAIN_CTRL_MODE *gc_mode);

获取接收增益模式,Y5x0 固定返回 0。rfid0~1 表示四组射频通道，rx1 表示每个射频通道的第一路

0=MGC 手动模式

1=FAST AGC 突发跟踪

2=SLOW AGC 包络跟踪

DLLEXPORT int32_t yunsdr_get_rx2_gain_control_mode (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,RF_GAIN_CTRL_MODE *gc_mode);

获取接收增益模式,Y5x0 固定返回 0。rfid0~1 表示四组射频通道，rx2 表示每个射频通道的第二路

0=MGC 手动模式

1=FAST AGC 突发跟踪

2=SLOW AGC 包络跟踪

DLLEXPORT int32_t yunsdr_get_rx1_rf_gain (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,int32_t *gain_db);

获取接收增益,Y5x0 固定返回 0。rfid0~1 表示 01 23 四组射频通道，rx1 表示每个射频通道的第一路

DLLEXPORT int32_t yunsdr_get_rx2_rf_gain (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,int32_t *gain_db);

获取接收增益,Y5x0 固定返回 0。rfid0~1 表示四组射频通道，rx2 表示每个射频通道的第二路

DLLEXPORT int32_t yunsdr_set_rx_lo_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t lo_freq_hz);

配置接收频点，rfid 0~1 代表 01 23 四组通道的频点。范围 75e6Hz ~ 5900e6Hz

DLLEXPORT int32_t yunsdr_set_rx_rf_bandwidth (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t bandwidth_hz);

配置接收带宽，Y5x0 无效。rfid0~1 表示 01 23 四组射频通道

DLLEXPORT int32_t yunsdr_set_rx_sampling_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t sampling_freq_hz);

配置采样率，rfid=0 同时作用收发采样率，Y5x0 默认 122880000Hz，可配置范围 122880000 除以 1，4~128

DLLEXPORT int32_t yunsdr_set_rx1_gain_control_mode (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,RF_GAIN_CTRL_MODE gc_mode);

配置接收增益模式,Y5x0 默认 0 手动模式。rfid0~1 表示 01 23 四组射频通道，rx1 表示每个射频通道的第一路

0=MGC 手动模式

1=FAST AGC 突发跟踪

2=SLOW AGC 包络跟踪

DLLEXPORT int32_t yunsdr_set_rx2_gain_control_mode (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,RF_GAIN_CTRL_MODE gc_mode);

配置接收增益模式,Y5x0 默认 0 手动模式。rfid0~1 表示 01 23 四组射频通道，rx2 表示每个射频通道的第二路

0=MGC 手动模式

1=FAST AGC 突发跟踪

2=SLOW AGC 包络跟踪

DLLEXPORT int32_t yunsdr_set_rx1_rf_gain (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,int32_t gain_db);

配置接收增益,Y5x0 设置范围 0~60。rfid0~1 表示 01 23 四组射频通道，rx1 表示每个射频通道的第一路

DLLEXPORT int32_t yunsdr_set_rx2_rf_gain (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,int32_t gain_db);

配置接收增益,Y5x0 设置范围 0~60。rfid0~1 表示 01 23 四组射频通道，rx2 表示每个射频通道的第二路

DLLEXPORT int32_t yunsdr_set_rx_fir_en_dis (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

配置接收滤波器使能,Y5x0 无效固定 0。rfid0~1 表示 01 23 四组射频通道

DLLEXPORT int32_t yunsdr_set_tx_lo_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t lo_freq_hz);

配置发送频点，rfid 0~1 代表 01 23 四组通道的频点。范围 75e6Hz ~ 5900e6Hz

DLLEXPORT int32_t yunsdr_set_tx_rf_bandwidth (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t bandwidth_hz);

配置接收带宽，rfid0~1 表示 01 23 四组射频通道。Y5x0 无效固定 100000000Hz

DLLEXPORT int32_t yunsdr_set_tx_sampling_freq (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t sampling_freq_hz);

配置采样率，rfid=0 同时作用收发采样率，Y5x0 默认 122880000Hz，可配置范围 122880000 除以 1,4~128。rfid=0

DLLEXPORT int32_t yunsdr_set_tx1_attenuation (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t attenuation_mdb);

配置发送衰减,Y5x0 配置范围 0~41950mdB。rfid0~1 表示 01 23 四组射频通道，tx1 表示每个射频通道的第一路

DLLEXPORT int32_t yunsdr_set_tx2_attenuation (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint32_t attenuation_mdb);

配置发送衰减,Y5x0 配置范围 0~41950mdB。rfid0~1 表示 01 23 四组射频通道，tx2 表示每个射频通道的第二路

DLLEXPORT int32_t yunsdr_set_tx_fir_en_dis (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t status);

配置发送滤波器使能,Y5x0 无效固定 0。rfid0~1 表示 01 23 四组射频通道

DLLEXPORT int32_t yunsdr_get_rfchip_reg (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, uint32_t reg,uint32_t *value);

读取指定寄存器。rfid0~1 表示 01 23 四组射频通道

DLLEXPORT int32_t yunsdr_set_rfchip_reg(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, uint32_t reg,uint32_t value);

写入指定寄存器。rfid0~1 表示 01 23 四组射频通道

DLLEXPORT int32_t yunsdr_set_tx_lo_int_ext (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

配置外部发送本振,rfid=0 Y5x0 无效固定内部本振

DLLEXPORT int32_t yunsdr_set_rx_lo_int_ext (YUNSDR_DESCRIPTOR *yunsdr,uint8_t rf_id,uint8_t enable);

配置外部接收本振,rfid=0 Y5x0 无效固定内部本振

DLLEXPORT int32_t yunsdr_set_ext_lo_freq(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint64_t lo_freq_hz);

配置外部本振频率,rfid=0 Y5x0 无效固定

DLLEXPORT int32_t yunsdr_do_mcs(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

配置手动同步，rfid=0 Y5x0 无效

DLLEXPORT int32_t yunsdr_set_rx_ant_enable(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

配置接收天线口使能，rfid=0 。1=接收 RX 天线口有效，0=接收 RX 天线口关闭，收发全部通过 TRX 天线口

DLLEXPORT int32_t yunsdr_set_ref_clock(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,REF_SELECT source);

配置内外参考使能，rfid=0 。1=外部参考，0=内部参考。参考频率 30.72MHz

DLLEXPORT int32_t yunsdr_set_vco_select (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,VCO_CAL_SELECT vco);

配置选择压控振荡器，rfid=0 Y5x0 无效

DLLEXPORT int32_t yunsdr_set_auxdac1 (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, uint32_t vol_mV);

配置晶振频偏调节电压，rfid=0。Y550S 设定 300

DLLEXPORT int32_t yunsdr_set_duplex_select(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,DUPLEX_SELECT duplex);

配置 TDD 或者 FDD 模式，rfid=0。0=TDD 模式，1=FDD 模式。当处于 TDD 模式时可以通过关闭 RX 接收天线口设置收发共用一个天线

DLLEXPORT int32_t yunsdr_tx_cyclic_enable(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

配置循环发送模式，rf_id=0

Enable=0，关闭

Enable=1，准备

Enable=3，开始输出

DLLEXPORT int32_t yunsdr_set_trxsw_fpga_enable(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

配置 fpga 切换天线使能，rfid=0。配合协议栈使用

DLLEXPORT int32_t yunsdr_set_hwbuf_depth(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, uint32_t depth);

配置接收内存深度，rfid=0 单位byte，默认是16*1024*1024，最大可以设置到1024*1024*1024*2-1024*1024*100

DLLEXPORT int32_t yunsdr_get_hwbuf_depth(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, uint32_t *depth);

获取接收内存深度 rfid=0

DLLEXPORT int32_t yunsdr_get_firmware_version(YUNSDR_DESCRIPTOR *yunsdr, uint32_t *version);

获取版本号 rfid=0

DLLEXPORT int32_t yunsdr_get_model_version(YUNSDR_DESCRIPTOR *yunsdr, uint32_t *version);

获取硬件设备号，rfid=0 Y5x0是550

DLLEXPORT int32_t yunsdr_set_pps_select (YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, PPSModeEnum pps);

配置时戳模式，rfid=0 Y5x0默认是0

0：内部产生

1：内部GPS产生

2：外部PPS输入

DLLEXPORT int32_t yunsdr_set_rxchannel_coef(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, RF_RX_CHANNEL channel, int16_t coef1, int16_t coef2);

DLLEXPORT int32_t yunsdr_enable_rxchannel_corr(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, RF_RX_CHANNEL channel, uint8_t enable);

DLLEXPORT int32_t yunsdr_set_txchannel_coef(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, RF_TX_CHANNEL channel, int16_t coef1, int16_t coef2);

DLLEXPORT int32_t yunsdr_enable_txchannel_corr(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, RF_TX_CHANNEL channel, uint8_t enable);

以上是多通道相位校准函数，Y5x0 通过上位机校准并计算

DLLEXPORT int32_t yunsdr_enable_timestamp(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id,uint8_t enable);

使能时间戳

DLLEXPORT int32_t yunsdr_read_timestamp(YUNSDR_DESCRIPTOR *yunsdr, uint8_t rf_id, uint64_t *timestamp);

获取时间戳

DLLEXPORT int32_t yunsdr_read_samples_multiport_Matlab (YUNSDR_DESCRIPTOR *yunsdr,void *buffer, uint32_t count, uint8_t channel_mask, uint64_t *timestamp);

RX 端收数据，具体用法请见 gen_trxbuf 和 gen_rxbuf 函数

DLLEXPORT int32_t yunsdr_write_samples_multiport_Matlab (YUNSDR_DESCRIPTOR *yunsdr,const void *buffer, uint32_t count, uint8_t channel_mask, uint64_t timestamp, uint32_t flags);

TX 端发数据，具体用法请见 gen_txbuf 函数