Introduction to SoapySDR

K Prasanna Kumar

Contents

***Abstract*—This module gives an introduction to the design of Software Defined Radio(SDR) driver for any radio system using SoapySDR architecture.**

1 Introduction

SoapySDR is an open source SDR Driver architecture. It is used with many software applications like GNURadio, PathosSDR MATLAB, OAI etc.. and also with open source Radio systems like BladeRF, HackRF etc.. It functional flowgraph is shown bellow

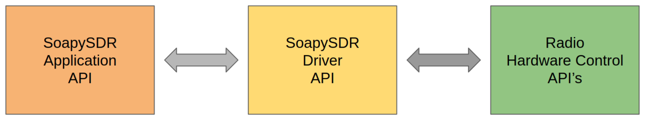

Fig. 1: SDR Driver Flow Graph

Similar flowgraph is present in PlutoSDR to interface libiio APIs with Software applications

2 Dependencies

```
sudo apt-get update
sudo apt-get install git cmake g++
    libpython-dev python-numpy swig
```

Author is from 5G Testbed, Dept. of ECE, IISc Bangalore, Email: kprasannakumar.iith@gmail.com, prasannakk@iisc.ac.in

3 Installation

```
git clone https://github.com/
    pothosware/SoapySDR.git
cd SoapySDR
git pull origin master
mkdir build
cd build
cmake ..
make -j4
sudo make install
sudo ldconfig
```

Verify the installation by running the following command

```
SoapySDRUtil --info
```

4 SDR Driver API

An example for the design of SDR driver using SoapySDR architecture is available in the following directory

```
cd ~/SoapySDR/ExampleDriver
```

Modify "CMakeLists.txt" & "MyDeviceSupport.cpp" present in the above directory as following

Problem 1. CMakeLists.txt

```
###############################
# Project setup -- only needed if
    device support is a stand-alone
    build
# We recommend that the support
    module be built in-tree with the
     driver.
###############################
cmake_minimum_required(VERSION
    2.6)
project(SoapySDRMyDevice CXX)
enable_testing()
```

```
#select the release build type by
   default to get optimization
   flags
if(NOT CMAKE_BUILD_TYPE)
    set(CMAKE_BUILD_TYPE "Release")
    message(STATUS "Build␣type␣not␣
       specified:␣defaulting␣to␣
       release.")
endif(NOT CMAKE_BUILD_TYPE)
set(CMAKE_BUILD_TYPE ${
   CMAKE_BUILD_TYPE} CACHE STRING "
   ")

##############################
# Header and library resources
   needed to communicate with the
   device.
# These may be found within the
   build tree or in an external
   project.
##############################
 #Path to .h and .hpp file
     directory of SoapySDR
include_directories(~/SoapySDR/
   include )

###############################
# build the module
###############################
find_package(SoapySDR CONFIG)

if (NOT SoapySDR_FOUND)
     message(WARNING "SoapySDR␣
        development␣files␣not␣found␣
        -␣skipping␣support")
     return()
endif ()


SOAPY_SDR_MODULE_UTIL(
     TARGET MyDevice
     SOURCES MyDeviceSupport.cpp
     LIBRARIES SoapySDR
)
```

Problem 2. MyDeviceSupport.cpp

```
#include <SoapySDR/Device.hpp>
#include <SoapySDR/Registry.hpp>
#include <iostream>
```

```
class MyDevice : public SoapySDR::
   Device
{
public:

     MyDevice(void)
     {
          return;
     }

     std::string getDriverKey(void)
          const
     {
          return "MyRF";
     }

     std::string getHardwareKey(
        void) const
     {
          return "MYRF007";
     }

          std::string getVendorInfo(
             void) const
     {
          return "Prasanna␣kumar";
     }

     SoapySDR::Kwargs
        getHardwareInfo(void) const
     {

          SoapySDR::Kwargs args;

          args["Developer"] = "
             Prasanna␣Kumar";
          std::cout<<"I␣am␣in␣
             getHardwareInfo␣"<<std::
             endl;

          return args;
     }

//##### Channel Functions ######
//#### Frequency Functions ######
//###### Gain Functions ########

// ######## Stream API's are as
   following
```

```
SoapySDR::Stream *setupStream(
   const int direction,
   const std::string &format,
   const std::vector<size_t> &
      channels,
   const SoapySDR::Kwargs &args)
   {
   std::cout<<"I am in Setup 
      Stream "<<std::endl;
   return (SoapySDR::Stream *)
      this;
         }

int activateStream(
        SoapySDR::Stream *stream,
        const int flags = 0,
        const long long timeNs =
           0,
        const size_t numElems = 0)
        {
        std::cout<<"I am activate 
           Stream"<<std::endl;
        return 10;
        }

int readStream(
   SoapySDR::Stream *stream,
   void * const *buffs,
   const size_t numElems,
   int &flags,
   long long &timeNs,
   const long timeoutUs )
   {
        std::cout<<"I am in 
           readStream"<<std::endl;
        return 0;
    }

int writeStream(
   SoapySDR::Stream *stream,
   const void * const *buffs,
   const size_t numElems,
   int &flags,
   const long long timeNs,
   const long timeoutUs )
        {
        std::cout<<buffs<<std::
           endl;
        std::cout<<"I am in 
           writeStream "<<std::endl
           ;
    return 0;
        }

int deactivateStream(
    SoapySDR::Stream *stream,
    const int flags = 0,
    const long long timeNs = 0)
    {
        std::cout<<"I am in 
           Deactivae Stream"<<std::
           endl;
        return 0;
        }


void closeStream(SoapySDR::Stream
   *stream)
        {
        std::cout<<"I am in close 
           Stream"<<std::endl;
        return;
        }

};


SoapySDR::KwargsList SoapySDR::
   Device::enumerate(const Kwargs&)
    {
        SoapySDR::KwargsList
           enum_results;
        SoapySDR::Kwargs enumArgs;

        enumArgs["driver"] = "
           MyDevice";
        enumArgs["type"] = "
           MyDevice";


        enum_results.push_back(
           enumArgs);
        return enum_results;

    }

SoapySDR::KwargsList findMyDevice(
   const SoapySDR::Kwargs &args)
{
    SoapySDR::KwargsList results;

    if (args.count("type") == 0)
        return results;
    if (args.at("type") != "
        MyDevice") return results;

    SoapySDR::Kwargs MyArgs;
    MyArgs["type"] = "MyDevice";

    results.push_back(MyArgs);

    return results;
}

SoapySDR::Device *makeMyDevice(
    const SoapySDR::Kwargs &)
{
    return new MyDevice();
}

static SoapySDR::Registry
    registerMyDevice("MyDevice", &
    findMyDevice, &makeMyDevice,
    SOAPY_SDR_ABI_VERSION);
```

Install the user defined example driver by the following commands

```
cd ~/SoapySDR/ExampleDriver
cmake .
make
sudo make install
```

This driver is designed using inheritance concept by overloading the functions in the class SoapySDR::Device which is present in " /SoapySDR/lib/Device.cpp". Develop the remaining driver using [3]

5 SDR Application API

5.1 C++ API

Copy the code from [4], modify it as follows for Rx & Tx Streams respectively.

Problem 3. Rx Application API

```
#include <cstdio>          //
    stdandard output
#include <cstdlib>

#include <SoapySDR/Device.hpp>
#include <SoapySDR/Types.hpp>
#include <SoapySDR/Formats.hpp>

#include <string>          // std::
    string
#include <vector>          // std::
    vector<...>
#include <map>             // std::
    map< ... , ... >

#include <iostream>

int main()
{

        // 1. create device
            instance

        //      1.1 set arguments
        //              args can
            be user defined or from
            the enumeration result
        //              We use
            first results as args
            here:
        SoapySDR::Kwargs args;
        args["driver"] = "MyDevice
            ";

        //      1.2 make device
        SoapySDR::Device *sdr =
            SoapySDR::Device::make(
            args);

        if( sdr == NULL )
        {
                fprintf(stderr, "
                    SoapySDR::Device
                    ::make␣failed\n"
                    );
                return
                    EXIT_FAILURE;
        }

        // 2. query device info
        std::vector< std::string >
            str_list;    //string
            list
```

```
//        2.1 antennas
str_list = sdr->
   listAntennas(
   SOAPY_SDR_RX, 0);
printf("Rx␣antennas:␣");
for(int i = 0; i <
   str_list.size(); ++i)
         printf("%s,",
            str_list[i].
            c_str());
printf("\n");

//        2.2 gains
str_list = sdr->listGains(
    SOAPY_SDR_RX, 0);
printf("Rx␣Gains:␣");
for(int i = 0; i <
   str_list.size(); ++i)
         printf("%s,␣",
            str_list[i].
            c_str());
printf("\n");

//        2.3. ranges(
   frequency ranges)
SoapySDR::RangeList ranges
    = sdr->
   getFrequencyRange(
   SOAPY_SDR_RX, 0);
printf("Rx␣freq␣ranges:␣")
   ;
for(int i = 0; i < ranges.
   size(); ++i)
         printf("[%g␣Hz␣->␣
            %g␣Hz],␣",
            ranges[i].
            minimum(),
            ranges[i].
            maximum());
printf("\n");

// 3. apply settings
sdr->setSampleRate(
   SOAPY_SDR_RX, 0, 10e6);

sdr->setFrequency(
   SOAPY_SDR_RX, 0, 433e6);

// 4. setup a stream (
   complex floats)
SoapySDR::Stream *
   rx_stream = sdr->
   setupStream(
   SOAPY_SDR_RX,
   SOAPY_SDR_CF32);
if( rx_stream == NULL)
{
         fprintf( stderr, "
            Failed\n");
         SoapySDR::Device::
            unmake( sdr );
         return
            EXIT_FAILURE;
}
sdr->activateStream(
   rx_stream, 0, 0, 0);

// 5. create a re-usable
   buffer for rx samples
std::complex<float> buff
   [1024];

// 6. receive some samples
for( int i = 0; i < 10; ++
   i)
{
         void *buffs[] = {
            buff};
         int flags;
         long long time_ns;
         int ret = sdr->
            readStream(
            rx_stream, buffs
            , 1024, flags,
            time_ns, 1e5);
         printf("ret␣=␣%d,␣
            flags␣=␣%d,␣
            time_ns␣=␣%lld\n
            ", ret, flags,
            time_ns);
}

// 7. shutdown the stream
sdr->deactivateStream(
   rx_stream, 0, 0);
            //stop streaming
sdr->closeStream(
   rx_stream );

// 8. cleanup device
```

```
        handle
    SoapySDR::Device::unmake(
        sdr );
    printf("Done\n");

    return EXIT_SUCCESS;
}
```

Save the above code with the file name "soapy-rx-api.cpp" and compile it using following command

```
g++ soapy-rx-api.cpp -o soapy-rx-
    api -lSoapySDR
```

Problem 4. Tx Application API

```
#include <cstdio>        //
    stdandard output
#include <cstdlib>

#include <SoapySDR/Device.hpp>
#include <SoapySDR/Types.hpp>
#include <SoapySDR/Formats.hpp>

#include <string>        // std::
    string
#include <vector>        // std::
    vector<...>
#include <map>           // std::
    map< ... , ... >

#include <iostream>

int main()
{

    // 1. create device
        instance

    //      1.1 set arguments
    //              args can
        be user defined or from
        the enumeration result
    //              We use
        first results as args
        here:
    SoapySDR::Kwargs args;
    args["driver"] = "MyDevice
        ";

    //      1.2 make device
    SoapySDR::Device *sdr =
        SoapySDR::Device::make(
        args);

    if( sdr == NULL )
    {
            fprintf(stderr, "
                SoapySDR::Device
                ::make␣failed\n"
                );
            return
                EXIT_FAILURE;
    }

    // 2. query device info
    std::vector< std::string >
        str_list;    //string
        list

    //      2.1 antennas
    str_list = sdr->
        listAntennas(
        SOAPY_SDR_TX, 0);
    printf("Rx␣antennas:␣");
    for(int i = 0; i <
        str_list.size(); ++i)
            printf("%s,",
                str_list[i].
                c_str());
    printf("\n");

    //      2.2 gains
    str_list = sdr->listGains(
        SOAPY_SDR_TX, 0);
    printf("Rx␣Gains:␣");
    for(int i = 0; i <
        str_list.size(); ++i)
            printf("%s,␣",
                str_list[i].
                c_str());
    printf("\n");

    //      2.3. ranges(
        frequency ranges)
    SoapySDR::RangeList ranges
        = sdr->
        getFrequencyRange(
        SOAPY_SDR_RX, 0);
    printf("Rx␣freq␣ranges:␣")
        ;
for(int i = 0; i < ranges.
    size(); ++i)
        printf(”[%g␣Hz␣->␣
            %g␣Hz],␣”,
            ranges[i].
            minimum(),
            ranges[i].
            maximum());
printf(”\n”);

// 3. apply settings
sdr->setSampleRate(
    SOAPY_SDR_TX, 0, 10e6);

sdr->setFrequency(
    SOAPY_SDR_TX, 0, 433e6);

// 4. setup a stream (
    complex floats)
SoapySDR::Stream *
    tx_stream = sdr->
    setupStream(
    SOAPY_SDR_TX,
    SOAPY_SDR_CF32);
if( tx_stream == NULL)
{
        fprintf( stderr, ”
            Failed\n”);
        SoapySDR::Device::
            unmake( sdr );
        return
            EXIT_FAILURE;
}
sdr->activateStream(
    tx_stream, 0, 0, 0);

// 5. create a re-usable
    buffer for tx samples
std::complex<float> buff
    [1024];

// 6. receive some samples
for( int i = 0; i < 10; ++
    i)
{
        void *buffs[] = {
            buff};
        int flags;
        long long time_ns;
        int ret = sdr->
            writeStream(
            tx_stream, buffs
            , 1024, flags,
            time_ns, 1e5);
        printf(”ret␣=␣%d,␣
            flags␣=␣%d,␣
            time_ns␣=␣%lld\n
            ”, ret, flags,
            time_ns);
    }

    // 7. shutdown the stream
    sdr->deactivateStream(
        tx_stream, 0, 0);
            //stop streaming
    sdr->closeStream(
        tx_stream );

    // 8. cleanup device
        handle
    SoapySDR::Device::unmake(
        sdr );
    printf(”Done\n”);

    return EXIT_SUCCESS;
}
```

Save the above code with the file name ”soapy-tx-api.cpp” and compile it using following command

```
g++ soapy-tx-api.cpp -o soapy-tx-
    api -lSoapySDR
```

5.2 Python API

Python bindings for SoapySDR

```
sudo apt-get install python-dev
    swig
```

Get the python code from [5]

```
import SoapySDR
from SoapySDR import * #SOAPY_SDR_
    constants
import numpy #use numpy for
    buffers

#enumerate devices
results = SoapySDR.Device.
    enumerate()
```

```
for result in results: print(
    result)

#create device instance
#args can be user defined or from
    the enumeration result
args = dict(driver="rtlsdr")
sdr = SoapySDR.Device(args)

#query device info
print(sdr.listAntennas(
   SOAPY_SDR_RX, 0))
print(sdr.listGains(SOAPY_SDR_RX,
    0))
freqs = sdr.getFrequencyRange(
   SOAPY_SDR_RX, 0)
for freqRange in freqs: print(
    freqRange)

#apply settings
sdr.setSampleRate(SOAPY_SDR_RX, 0,
     1e6)
sdr.setFrequency(SOAPY_SDR_RX, 0,
    912.3e6)

#setup a stream (complex floats)
rxStream = sdr.setupStream(
   SOAPY_SDR_RX, SOAPY_SDR_CF32)
sdr.activateStream(rxStream) #
    start streaming

#create a re-usable buffer for rx
    samples
buff = numpy.array([0]*1024, numpy
    .complex64)

#receive some samples
for i in range(10):
      sr = sdr.readStream(rxStream,
          [buff], len(buff))
      print(sr.ret) #num samples or
          error code
      print(sr.flags) #flags set by
          receive operation
      print(sr.timeNs) #timestamp
          for receive buffer

#shutdown the stream
sdr.deactivateStream(rxStream) #
    stop streaming
sdr.closeStream(rxStream)
```

References

[1] SoapySDR Build Guide https://github.com/pothosware/SoapySDR/wiki/BuildGuide

[2] SoapySDR Project wiki https://github.com/pothosware/SoapySDR/wiki

[3] SoapySDR Docxygen https://pothosware.github.io/SoapySDR/doxygen/latestclassSoapySDR_1_1Device.html

[4] SoapySDR C++ Application https://github.com/pothosware/SoapySDR/wiki/Cpp_API_Example

[5] SoapySDR Python Application https://github.com/pothosware/SoapySDR/wiki/PythonSupport