Ettus UHD Research Project

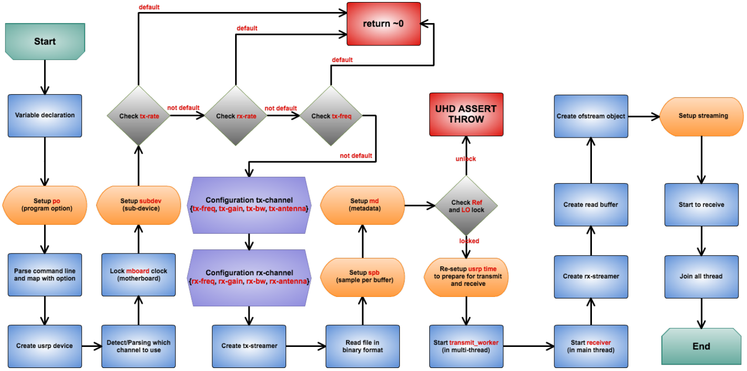
Extended from Ettus Research example code txrx_loopback_to_file

Cheng-en, Sung

Department of Computer Science Engineering
University of Yuan ze

26th June, 2017

Flow chart



Start
Variable declaration
Setup po (program option)
Parse command line and map with option
Create usrp device
Detect/Parsing which channel to use
Lock mboard clock (motherboard)
Setup subdev (sub-device)
Check tx-rate
Check rx-rate
Check tx-freq
default
default
default
not default
not default
not default
return ~0
Configuration tx-channel {tx-freq, tx-gain, tx-bw, tx-antenna}
Configuration rx-channel {rx-freq, rx-gain, rx-bw, rx-antenna}
Create tx-streamer
Read file in binary format
Setup spb (sample per buffer)
Setup md (metadata)
Check Ref and LO lock
unlock
locked
UHD ASSERT THROW
Re-setup usrp time to prepare for transmit and receive
Start transmit_worker (in multi-thread)
Start receiver (in main thread)
Create rx-streamer
Create read buffer
Create ofstream object
Setup streaming
Start to receive
Join all thread
End

Variable declare

```
//transmit variables to be set by po
std::string tx_args, tx_ant, tx_subdev, ref, otw, tx_channels, tx_data;
double tx_rate, tx_freq, tx_gain, tx_bw;
float ampl;

//receive variables to be set by po
std::string rx_args, file, type, rx_ant, rx_subdev, rx_channels;
size_t total_num_samps, spb;
double rx_rate, rx_freq, rx_gain, rx_bw;
float settling;
```

Setup po (program options)

```
//setup the program options
po::options_description desc("Allowed options");
desc.add_options()
    ("help", "help message")
    ("tx-args", po::value<std::string>(&tx_args)->default_value(""), "
         uhd transmit device address args")
    ("rx-args", po::value<std::string>(&rx_args)->default_value(""), "
         uhd receive device address args")
    ("file", po::value<std::string>(&file)->default_value("usrp_samples.
         dat"), "name of the file to write binary samples to")
    ("type", po::value<std::string>(&type)->default_value("short"), "
         sample type in file: double, float, or short")
    ...
    ...
    ...
    ("otw", po::value<std::string>(&otw)->default_value("sc16"), "
         specify the over-the-wire sample mode")
    ("tx-channels", po::value<std::string>(&tx_channels)->default_value(
         "0"), "which TX channel(s) to use (specify \"0\", \"1\",
         \"0,1\", etc)")
    ("rx-channels", po::value<std::string>(&rx_channels)->default_value(
         "0"), "which RX channel(s) to use (specify \"0\", \"1\",
         \"0,1\", etc)")
    ("tx-int-n", "tune USRP TX with integer-N tuning")
    ("rx-int-n", "tune USRP RX with integer-N tuning")
    ("tx-data", po::value<std::string>(&tx_data), "transmit data")
;
```

Parse command line and map with option

```
//parse command line and map with option
po::variables_map vm;
po::store(po::parse_command_line(argc, argv, desc), vm);
po::notify(vm);
```

Create usrp device

```
//create a usrp device
std::cout << std::endl;
std::cout << boost::format("Creating the transmit usrp device with: %s
    ...") % tx_args << std::endl;
uhd::usrp::multi_usrp::sptr tx_usrp = uhd::usrp::multi_usrp::make(
    tx_args);
std::cout << std::endl;
std::cout << boost::format("Creating the receive usrp device with: %s...
    ") % rx_args << std::endl;
uhd::usrp::multi_usrp::sptr rx_usrp = uhd::usrp::multi_usrp::make(
    rx_args);
```

Detect/Parsing which channel to use

```
//detect/parsing which channels to use
//default chnnel: 0, input channel foramt(0,1,2 or 0\1\2).
std::vector<std::string> tx_channel_strings;
std::vector<size_t> tx_channel_nums;
boost::split(tx_channel_strings, tx_channels, boost::is_any_of("\"',"));
for(size_t ch = 0; ch < tx_channel_strings.size(); ch++){
    size_t chan = boost::lexical_cast<int>(tx_channel_strings[ch]);
    if(chan >= tx_usrp->get_tx_num_channels()){
        throw std::runtime_error("Invalid TX channel(s) specified.");
    }
    else tx_channel_nums.push_back(boost::lexical_cast<int>(
         tx_channel_strings[ch]));
}
std::vector<std::string> rx_channel_strings;
std::vector<size_t> rx_channel_nums;
boost::split(rx_channel_strings, rx_channels, boost::is_any_of("\"',"));
for(size_t ch = 0; ch < rx_channel_strings.size(); ch++){
    size_t chan = boost::lexical_cast<int>(rx_channel_strings[ch]);
    if(chan >= rx_usrp->get_rx_num_channels()){
        throw std::runtime_error("Invalid RX channel(s) specified.");
    }
    else rx_channel_nums.push_back(boost::lexical_cast<int>(
         rx_channel_strings[ch]));
}
```

Lock mboard and Setup TX/RX subdev

```
//Lock mboard(mother-board) clocks
tx_usrp->set_clock_source(ref);
rx_usrp->set_clock_source(ref);

// setup tx/rx subdev if any of these parameter is given.
if (vm.count("tx-subdev")) tx_usrp->set_tx_subdev_spec(tx_subdev);
if (vm.count("rx-subdev")) rx_usrp->set_rx_subdev_spec(rx_subdev);
```

Check TX/RX rate, TX freq

```
//check the transmit sample rate
if (not vm.count("tx-rate")){
    std::cerr << "Please specify the transmit sample rate with --tx-rate
         " << std::endl;
    return ~0;
}
std::cout << boost::format("Setting TX Rate: %f Msps...") % (tx_rate/1e6
     ) << std::endl;
tx_usrp->set_tx_rate(tx_rate);
std::cout << boost::format("Actual TX Rate: %f Msps...") % (tx_usrp->
     get_tx_rate()/1e6) << std::endl << std::endl;

//check the receive sample rate
if (not vm.count("rx-rate")){
    std::cerr << "Please specify the sample rate with --rx-rate" << std
         ::endl;
    return ~0;
}
std::cout << boost::format("Setting RX Rate: %f Msps...") % (rx_rate/1e6
     ) << std::endl;
rx_usrp->set_rx_rate(rx_rate);
std::cout << boost::format("Actual RX Rate: %f Msps...") % (rx_usrp->
     get_rx_rate()/1e6) << std::endl << std::endl;

//check the transmit center frequency
if (not vm.count("tx-freq")){
    std::cerr << "Please specify the transmit center frequency with --tx
         -freq" << std::endl;
    return ~0;
}
```

Config TX channel

```
//config each tx channel
for(size_t ch = 0; ch < tx_channel_nums.size(); ch++) {
    size_t channel = tx_channel_nums[ch];
    if (tx_channel_nums.size() > 1) {
        std::cout << "Configuring TX Channel " << channel << std::endl;
    }
    std::cout << boost::format("Setting TX Freq: %f MHz...") % (tx_freq
        /1e6) << std::endl;
    uhd::tune_request_t tx_tune_request(tx_freq);
    if(vm.count("tx-int-n")) tx_tune_request.args = uhd::device_addr_t("
        mode_n=integer");
    tx_usrp->set_tx_freq(tx_tune_request, channel);
    std::cout << boost::format("Actual TX Freq: %f MHz...") % (tx_usrp->
        get_tx_freq(channel)/1e6) << std::endl << std::endl;
    if (vm.count("tx-gain")){
        std::cout << boost::format("Setting TX Gain: %f dB...") %
            tx_gain << std::endl;
        tx_usrp->set_tx_gain(tx_gain, channel);
        std::cout << boost::format("Actual TX Gain: %f dB...") % tx_usrp
            ->get_tx_gain(channel) << std::endl << std::endl;
    }
    if (vm.count("tx-bw")){
        std::cout << boost::format("Setting TX Bandwidth: %f MHz...") %
            tx_bw << std::endl;
        tx_usrp->set_tx_bandwidth(tx_bw, channel);
        std::cout << boost::format("Actual TX Bandwidth: %f MHz...") %
            tx_usrp->get_tx_bandwidth(channel) << std::endl << std::
            endl;
    }
    if (vm.count("tx-ant")) tx_usrp->set_tx_antenna(tx_ant, channel);
}
```

Config RX channel

```
for(size_t ch = 0; ch < rx_channel_nums.size(); ch++) {
    size_t channel = rx_channel_nums[ch];
    if (rx_channel_nums.size() > 1) {
        std::cout << "Configuring RX Channel " << channel << std::endl;
    }

    //set the receive center frequency
    if (not vm.count("rx-freq")){
        std::cerr << "Please specify the center frequency with --rx-freq
             " << std::endl;
        return ~0;
    }
    std::cout << boost::format("Setting RX Freq: %f MHz...") % (rx_freq
         /1e6) << std::endl;
    uhd::tune_request_t rx_tune_request(rx_freq);
    if(vm.count("rx-int-n")) rx_tune_request.args = uhd::device_addr_t("
         mode_n=integer");
    rx_usrp->set_rx_freq(rx_tune_request, channel);
    std::cout << boost::format("Actual RX Freq: %f MHz...") % (rx_usrp->
         get_rx_freq(channel)/1e6) << std::endl << std::endl;

    //set the receive rf gain
    if (vm.count("rx-gain")){
        std::cout << boost::format("Setting RX Gain: %f dB...") %
             rx_gain << std::endl;
        rx_usrp->set_rx_gain(rx_gain, channel);
        std::cout << boost::format("Actual RX Gain: %f dB...") % rx_usrp
             ->get_rx_gain(channel) << std::endl << std::endl;
    }
```

Config RX channel (cont.)

```
    //set the receive analog frontend filter bandwidth
    if (vm.count("rx-bw")){
        std::cout << boost::format("Setting RX Bandwidth: %f MHz...") %
             (rx_bw/1e6) << std::endl;
        rx_usrp->set_rx_bandwidth(rx_bw, channel);
        std::cout << boost::format("Actual RX Bandwidth: %f MHz...") % (
             rx_usrp->get_rx_bandwidth(channel)/1e6) << std::endl << std
             ::endl;
    }
}
//set the receive antenna
if (vm.count("ant")) rx_usrp->set_rx_antenna(rx_ant);
```

Create TX streamer

```
//create a transmit streamer
/*
    otw stands for "over-the-wire sample mode"
    Setting the OTW ("over-the-wire") format is, in theory, transparent
         to the application, but changing this can have some side
         effects.
    Using less bits for example (e.g. when going from otw_format sc16 to
          sc8) will reduce the dynamic range, and increases quantization
          noise.
    On the other hand, it reduces the load on the data link and thus
         allows more bandwidth (a USRP N210 can work with 25 MHz
         bandwidth for 16-Bit complex samples, and 50 MHz for 8-Bit
         complex samples).
    The following otw format have been implemented:
     * sc16 (default using)
     * sc12
     * sc8

*/
uhd::stream_args_t stream_args("fc32", otw);
stream_args.channels = tx_channel_nums;
// Get the max number of samples per buffer per packet.
uhd::tx_streamer::sptr tx_stream = tx_usrp->get_tx_stream(stream_args);
```

Read file in binary format

```
/*****************************************************************
 * read file function
 *****************************************************************/
vector< std::complex<double> read_file(string tx_data)
{
    std::vector< std::complex<double> > buff;

    std::ifstream is;
    is.open (tx_data, std::ios::binary | std::ios::in);
    is.seekg (0, std::ios::end);
    int rlen = is.tellg();
    is.seekg (0, std::ios::beg);

    char *rbuff = new char [rlen];
    is.read (rbuff,rlen);
    is.close();

    double* pbuff = (double*)rbuff;
    std::vector<double> tmp_buff(pbuff, pbuff + (rlen / sizeof(double)))
        ;
    double real = 0.0, imag = 0.0;
    for ( int i = 0; i < tmp_buff.size(); i++) {
        if (i % 2 == 0) {
            real = tmp_buff[i];
        } else {
            imag = tmp_buff[i];
        }
        std::complex<double> cp(real, imag);
        buff.push_back(cp);
    }
    retrun buff;
}
```

Setup spb (sample per buffer), md (metadata)

```
//setup sample per buffer
if (spb == 0) spb = tx_stream->get_max_num_samps()*10; // ?
int num_channels = tx_channel_nums.size();

//setup the metadata flags
uhd::tx_metadata_t md;
md.start_of_burst = true;
md.end_of_burst   = false;
md.has_time_spec  = true;
md.time_spec = uhd::time_spec_t(0.1); //give us 0.1 seconds to fill the
     tx buffers
```

Check Ref and LO lock

```
//Check Ref and LO Lock detect
std::vector<std::string> tx_sensor_names, rx_sensor_names;
tx_sensor_names = tx_usrp->get_tx_sensor_names(0);
if (std::find(tx_sensor_names.begin(), tx_sensor_names.end(), "lo_locked
    ") != tx_sensor_names.end()) {
    uhd::sensor_value_t lo_locked = tx_usrp->get_tx_sensor("lo_locked"
        ,0);
    std::cout << boost::format("Checking TX: %s ...") % lo_locked.
        to_pp_string() << std::endl;
    UHD_ASSERT_THROW(lo_locked.to_bool());
}
rx_sensor_names = rx_usrp->get_rx_sensor_names(0);
if (std::find(rx_sensor_names.begin(), rx_sensor_names.end(), "lo_locked
    ") != rx_sensor_names.end()) {
    uhd::sensor_value_t lo_locked = rx_usrp->get_rx_sensor("lo_locked"
        ,0);
    std::cout << boost::format("Checking RX: %s ...") % lo_locked.
        to_pp_string() << std::endl;
    UHD_ASSERT_THROW(lo_locked.to_bool());
}
tx_sensor_names = tx_usrp->get_mboard_sensor_names(0);

if ((ref == "mimo") and (std::find(tx_sensor_names.begin(),
    tx_sensor_names.end(), "mimo_locked") != tx_sensor_names.end())) {
    uhd::sensor_value_t mimo_locked = tx_usrp->get_mboard_sensor("
        mimo_locked",0);
    std::cout << boost::format("Checking TX: %s ...") % mimo_locked.
        to_pp_string() << std::endl;
    UHD_ASSERT_THROW(mimo_locked.to_bool());
}
```

Check Ref and LO lock (cont.)

```cpp
if ((ref == "external") and (std::find(tx_sensor_names.begin(), tx_sensor_names.end(), "ref_locked") != tx_sensor_names.end())) {
    uhd::sensor_value_t ref_locked = tx_usrp->get_mboard_sensor("ref_locked",0);
    std::cout << boost::format("Checking TX: %s ...") % ref_locked.to_pp_string() << std::endl;
    UHD_ASSERT_THROW(ref_locked.to_bool());
}

rx_sensor_names = rx_usrp->get_mboard_sensor_names(0);
if ((ref == "mimo") and (std::find(rx_sensor_names.begin(), rx_sensor_names.end(), "mimo_locked") != rx_sensor_names.end())) {
    uhd::sensor_value_t mimo_locked = rx_usrp->get_mboard_sensor("mimo_locked",0);
    std::cout << boost::format("Checking RX: %s ...") % mimo_locked.to_pp_string() << std::endl;
    UHD_ASSERT_THROW(mimo_locked.to_bool());
}
if ((ref == "external") and (std::find(rx_sensor_names.begin(), rx_sensor_names.end(), "ref_locked") != rx_sensor_names.end())) {
    uhd::sensor_value_t ref_locked = rx_usrp->get_mboard_sensor("ref_locked",0);
    std::cout << boost::format("Checking RX: %s ...") % ref_locked.to_pp_string() << std::endl;
    UHD_ASSERT_THROW(ref_locked.to_bool());
}
```

Reset USRP time

```
//reset usrp time to prepare for transmit/receive
std::cout << boost::format("Setting device timestamp to 0...") << std::
    endl;
tx_usrp->set_time_now(uhd::time_spec_t(0.0));
```

Start transmit worker, recevier

```
//start transmit worker thread
boost::thread_group transmit_thread;
transmit_thread.create_thread(boost::bind(&transmit_worker, buff,
     tx_stream, md, num_channels));

//recv to file
if (type == "double") recv_to_file<std::complex<double> >(rx_usrp, "fc64
     ", otw, file, spb, total_num_samps, settling, rx_channel_nums);
else if (type == "float") recv_to_file<std::complex<float> >(rx_usrp, "
     fc32", otw, file, spb, total_num_samps, settling, rx_channel_nums);
else if (type == "short") recv_to_file<std::complex<short> >(rx_usrp, "
     sc16", otw, file, spb, total_num_samps, settling, rx_channel_nums);
else {
    //clean up transmit worker
    stop_signal_called = true;
    transmit_thread.join_all();
    throw std::runtime_error("Unknown type " + type);
}
```

Create RX streamer, read buffer, ofstream object

```cpp
//create a receive streamer
uhd::stream_args_t stream_args(cpu_format,wire_format);
stream_args.channels = rx_channel_nums;
uhd::rx_streamer::sptr rx_stream = usrp->get_rx_stream(stream_args);

// Prepare buffers for received samples and metadata
uhd::rx_metadata_t md;
std::vector <std::vector< samp_type > > buffs(
    rx_channel_nums.size(), std::vector< samp_type >(samps_per_buff)
);
//create a vector of pointers to point to each of the channel buffers
std::vector<samp_type *> buff_ptrs;
for (size_t i = 0; i < buffs.size(); i++) {
    buff_ptrs.push_back(&buffs[i].front());
}
```

Setup streaming

```
//setup streaming
uhd::stream_cmd_t stream_cmd((num_requested_samples == 0)?
    uhd::stream_cmd_t::STREAM_MODE_START_CONTINUOUS:
    uhd::stream_cmd_t::STREAM_MODE_NUM_SAMPS_AND_DONE
);
stream_cmd.num_samps = num_requested_samples;
stream_cmd.stream_now = false;
stream_cmd.time_spec = uhd::time_spec_t(settling_time);
rx_stream->issue_stream_cmd(stream_cmd);
```

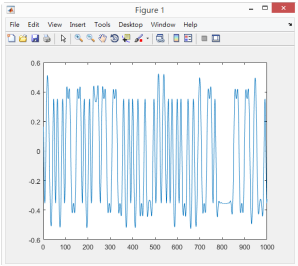
Start to receive

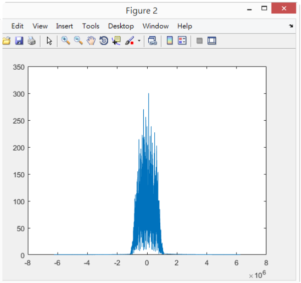
```
while(not stop_signal_called and (num_requested_samples >
      num_total_samps or num_requested_samples == 0)) {

    size_t num_rx_samps = rx_stream->recv(buff_ptrs, samps_per_buff, md,
          timeout);
    timeout = 0.1f; //small timeout for subsequent recv

    if (md.error_code == uhd::rx_metadata_t::ERROR_CODE_TIMEOUT) {
        ...
    }
    if (md.error_code == uhd::rx_metadata_t::ERROR_CODE_OVERFLOW){
        ...
    }
    if (md.error_code != uhd::rx_metadata_t::ERROR_CODE_NONE){
        ...
    }
    num_total_samps += num_rx_samps;

    for (size_t i = 0; i < outfiles.size(); i++) {
        outfiles[i]->write((const char*) buff_ptrs[i], num_rx_samps*
              sizeof(samp_type));
    }

}
```

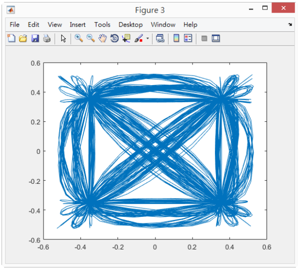
Join all thread

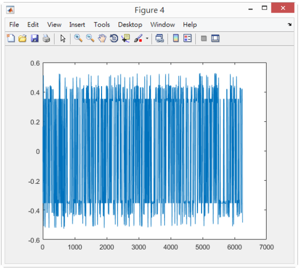
```
if (type == "double") recv_to_file<std::complex<double> >(rx_usrp, "fc64
     ", otw, file, spb, total_num_samps, settling, rx_channel_nums);
else if (type == "float") recv_to_file<std::complex<float> >(rx_usrp, "
     fc32", otw, file, spb, total_num_samps, settling, rx_channel_nums);
else if (type == "short") recv_to_file<std::complex<short> >(rx_usrp, "
     sc16", otw, file, spb, total_num_samps, settling, rx_channel_nums);
else {
    //clean up transmit worker
    stop_signal_called = true;
    transmit_thread.join_all();
    throw std::runtime_error("Unknown type " + type);
}
```

TX data result

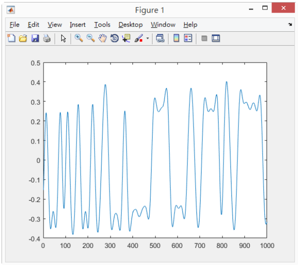


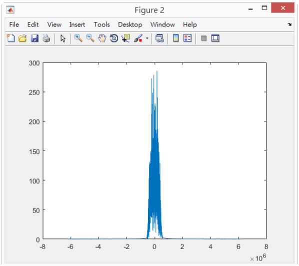


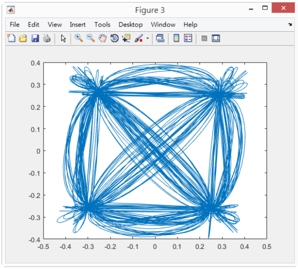


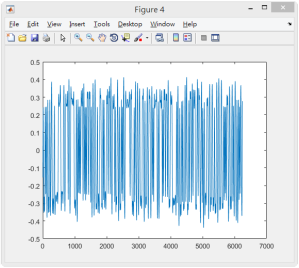


RX data result









Reference

- Getting Started with UHD and C++
  https://kb.ettus.com/Getting_Started_with_UHD_and_C++
- EttusResearch project
  https://github.com/EttusResearch/uhd/
- USRP Hardware Driver and USRP Manual
  https://files.ettus.com/manual/
- EttusResearch example code
  https://githubcom/EttusResearch/uhd/txrx_loopback_to_file.cpp
- Source code link
  http://codepad.org/iQoh4MCZ