Lime Microsystems Limited

Surrey Tech Centre
Occam Road
The Surrey Research Park
Guildford GU2 7YG
Surrey
United Kingdom

Tel: +44 (0) 1483 685 063
Fax: +44 (0) 1428 656 662
e-mail: enquiries@limemicro.com

LimeSDR-USB v1.4

- MIMO channel alignment -

Chip version:	-
Chip revision:	-
Document version:	01
Last modified:	08-05-2018 10:31:29 AM

Contents

Revision History

Version 1.0r00
Released: 08 Apr, 2018
Initial version

1

Introduction

This document provides a brief description of known phase alignment issues on LimeSDR-USB and explains the steps that are used in attempt to correct these issues in LimeSuite.

2

Description of the issues

This chapter provides a quick overview of the known phase alignment issues on LimeSDR-USB. There are four known issues that cause inconsistencies in phase offset between two Rx channels:

1) 1 LML sample offset in RxTSP→LML
2) 1 TSP clock cycle offset in AFE→RxTSP
3) Multiple (0 to 3) CGEN clock cycles offset in AFE
4) 180 degrees phase offset in Quadrature generator.

2.1 One LML sample offset in RxTSP→LML

To synchronize RxTSP modules, RxTSP logic resets (0x20[9] and 0x20[11]) need to be performed at the same time.

Most of the time, offset between channels becomes 0 (for signals generated in RxTSP) after performing RxTSP logic resets.

Sometimes (chance seems to be less than 10%) offset becomes 1 LML sample and in this state it can be changed between 0 and 1 LML sample when toggling LML Rx FIFO reset (0x20[7]).

This issue can be observed using either RxTSP test signals or external signal.

2.2 One TSP clock offset in RxTSP→LML

When using external signal and after properly aligning RxTSP→LML, phase offset between channels may be one of the two values that differ by the duration equal to 1 TSP clock cycle. In any of the two states, the offset becomes 0 when switched to using RxTSP test signal. The issue can be observed by simply toggling RxTSP logic resets, however in such case resulting phase offsets is combination of this issue and issue described in 2.1.

This issue can only be observed when using ADC (external signal). It does not appear when using signals generated by RxTSP (TSG NCO or TSG DC with CMIX).

Increasing decimation does not have effect (offset is still equal to 1 TSP clock cycle).

2.3 Multiple CGEN clocks offset in AFE

Channel phase offset of 0, 1, 2 or 3 CGEN clocks is introduced after toggling:

- CGEN enable (0x86[0])
- or CGEN forward divider (0x86[4])
- or CGEN reset pulse (0x86[14])
- or ADC power down (0x82[3], 0x82[4])

When toggling these controls the resulting phase offset varies between total of 8 values that differ by steps equal to CGEN clock cycle. However, half of these values are obtained because of the issue described in 2.2, which causes jumping between two values separated by 1 TSP clock cycle (equal to 4 CGEN clock cycles). So when combined these two issues may produce 4 pairs of values.

This issue only appears when using ADC (external signal). It does not appear when using signals generated by RxTSP (TSG NCO or TSG DC with CMIX).

2.4 180 degree phase offset in Quadrature generator.

Channel phase offset of 0 or 180 degrees is introduced after toggling:

- Quadrature LO generator (0x10C[3])
- RFE module (0x10C[0])
- RFE path (0x10D[8:7])
- SXR VCO (0x11C[1])
- SXR module (0x11C[0])
- SXR forward divider (0x11C[4])

3

Channel alignment functions

Channel alignment is done by resetting various LMS7002m modules multiple times and checking the offset between Rx channels after each reset. The function that does channel alignment in LimeSuite is Streamer::AlignRxRF(). All channel alignment related functionality is currently located in 'LimeSuite/src/protocols/Streamer.cpp' file.

3.1 AlignRxRF()

AlignRxRF() is the main channel alignment function. At first, it saves current LMS7002m configuration and configures the chip to use internal RF loopback. The Rx frequency is set to 450 MHz and TxTSP modules are configured to generated DC test signals. Tx channels share the same LO and the test signal DC values are the same, so signals that are transmitted to both Rx channels via loopbacks are also the same.

```
void Streamer::AlignRxRF(bool restoreValues)
{
    auto regBackup = lms->BackupRegisterMap();
    lms->SPI_write(0x20, 0xFFFF);
    lms->SetDefaults(LMS7002M::RFE);
    lms->SetDefaults(LMS7002M::RBB);
    lms->SetDefaults(LMS7002M::TBB);
    lms->SetDefaults(LMS7002M::TRF);
    lms->SPI_write(0x10C, 0x88C5);
    lms->SPI_write(0x10D, 0x0117);
    lms->SPI_write(0x113, 0x024A);
    lms->SPI_write(0x118, 0x418C);
    lms->SPI_write(0x100, 0x4039);
    lms->SPI_write(0x101, 0x7801);
    lms->SPI_write(0x103, 0x0612);
    lms->SPI_write(0x108, 0x318C);
    lms->SPI_write(0x082, 0x8001);
    lms->SPI_write(0x200, 0x008D);
    lms->SPI_write(0x208, 0x01FB);
    lms->SPI_write(0x400, 0x8081);
    lms->SPI_write(0x40C, 0x01FF);
    lms->SPI_write(0x404, 0x0006);
    lms->LoadDC_REG_IQ(true, 0x3FFF, 0x3FFF);
    double srate = lms->GetSampleRate(false, LMS7002M::ChA);
     lms->SetFrequencySX(false,450e6);
```

After loading the test configuration, multiple resets and checks are performed in a loop until Rx channels seem to be aligned well enough. The following steps are performed in the loop:

- CGEN divider is toggled on and off to bring clock offsets to unknown (random) state.
- RxTSP→ LML alignment is performed by calling AlignRxTSP() (detailed in 3.3).
- Phase offsets between channels are obtained at two different signal frequencies. GetPhaseOffset() function (detailed in 3.2) is used to obtain phase offset between Rx channels.
- Phase offsets at different frequencies are compared, If the difference between them does not exceed defined tolerance values, channels are considered aligned and loop exits. Offsets at two frequencies are compared because phase offset between channels changes linearly with frequency when there is a clock offset but when there is no clock offset the change is usually very small.

```
    int dec = lms->Get_SPI_Reg_bits(LMS7_HBD_OVR_RXTSP);
    if (dec > 4) dec = 0;

    double offsets[] = {1.15/60.0, 1.1/40.0, 0.55/20.0, 0.2/10.0, 0.18/5.0};
    double tolerance[] = {0.9, 0.45, 0.25, 0.14, 0.06};
    double offset = offsets[dec]*srate/1e6;

    dataPort->WriteRegister(0xFFFF, 1 << chipId);
    fpga->StopStreaming();
    dataPort->WriteRegister(0x0008, 0x0100);
    dataPort->WriteRegister(0x0007, 3);
    bool found = false;
    for (int i = 0; i < 200; i++){
        lms->Modify_SPI_Reg_bits(LMS7_PD_FDIV_O_CGEN, 1);
        lms->Modify_SPI_Reg_bits(LMS7_PD_FDIV_O_CGEN, 0);
        AlignRxTSP();

        lms->SetFrequencySX(true, 450e6+srate/16.0);
        double offset1 = GetPhaseOffset(32);
        if (offset1 < -360)
            break;
        lms->SetFrequencySX(true, 450e6+srate/8.0);
        double offset2 = GetPhaseOffset(64);
        if (offset2 < -360)
            break;
        double diff = offset1-offset2;
        if (abs(diff-offset) < tolerance[dec])
        {
            found = true;
            break;
        }
     }
```

Finally, the chip settings are restored and AlignQuadrature() is called in order to correct 180 degrees phase offset between quadrature generators, because the previous procedure does not account for possible 180 degree phase flips in Tx and Rx quadrature generators

```
    if (restoreValues)
        lms->RestoreRegisterMap(regBackup);
    if (found)
        AlignQuadrature(restoreValues);
    else
        lime::warning("Channel alignment failed");
}
```

3.2 GetPhaseOffset()

GetPhaseOffset() function resets sample streaming buffers to clear old data, reads samples from the chip and calculates a DFT bin corresponding to the frequency of interest for both Rx channels. The function returns phase difference between channels that is calculated from the DFT bins.

```
double Streamer::GetPhaseOffset(int bin)
{
    int16_t* buf = new int16_t[sizeof(FPGA_DataPacket)/sizeof(int16_t)];

    dataPort->ResetStreamBuffers();
    fpga->StartStreaming();
          if  (dataPort->ReceiveData((char*)buf,  sizeof(FPGA_DataPacket),  chipId,  50)!
=sizeof(FPGA_DataPacket))
    {
        lime::warning("Channel alignment failed");
        delete [] buf;
        return -1000;
    }
    fpga->StopStreaming();
    dataPort->AbortReading(chipId);
    //calculate DFT bin of interest and check channel phase difference
    const std::complex<double> iunit(0, 1);
    const double pi = std::acos(-1);
    const int N = 512;
    std::complex<double> xA(0,0);
    std::complex<double> xB(0, 0);
    for (int n = 0; n < N; n++) {
        const std::complex<double> xAn(buf[8+4*n], buf[9+4*n]);
        const std::complex<double> xBn(buf[10+4*n],buf[11+4*n]);
const std::complex<double> mult = std::exp(-2.0*iunit*pi*double(bin)* double(n)/double(N));
        xA += xAn * mult;
        xB += xBn * mult;
    }
    double phaseA = std::arg(xA) * 180.0 / pi;
    double phaseB = std::arg(xB) * 180.0 / pi;
    double phasediff = phaseB - phaseA;
    if (phasediff < -180.0) phasediff +=360.0;
    if (phasediff > 180.0) phasediff -=360.0;
    delete [] buf;
    return phasediff;
}
```

3.3 AlignRxTSP()

AlignRxTSP() corrects sample alignment between TSP and LML modules. At first it saves current chip configuration and configures both channels to generate NCO test signals from RxTSP modules.

```
void Streamer::AlignRxTSP()
{
    uint32_t reg20;
    uint32_t regsA[2];
    uint32_t regsB[2];

    {//backup values
      const std::vector<uint32_t> bakAddr={(uint32_t(0x0400)<<16), (uint32_t(0x040C)<<16)};
        uint32_t data = (uint32_t(0x0020) << 16);
```

```
        dataPort->ReadLMS7002MSPI(&data, &reg20, 1, chipId);
        data = (uint32_t(0x0020) << 16) | 0xFFFD;
        dataPort->WriteLMS7002MSPI(&data, 1, chipId);
        dataPort->ReadLMS7002MSPI(bakAddr.data(), regsA, bakAddr.size(), chipId);
        data = (uint32_t(0x0020) << 16) | 0xFFFE;
        dataPort->WriteLMS7002MSPI(&data, 1, chipId);
        dataPort->ReadLMS7002MSPI(bakAddr.data(), regsB, bakAddr.size(), chipId);
    }
    {//alignment search
        uint32_t dataWr[4];
        dataWr[0] = (1 << 31) | (uint32_t(0x0020) << 16) | 0xFFFF;
        dataWr[1] = (1 << 31) | (uint32_t(0x0400) << 16) | 0x8085;
        dataWr[2] = (1 << 31) | (uint32_t(0x040C) << 16) | 0x01FF;
        dataPort->WriteLMS7002MSPI(dataWr, 3, chipId);
        uint32_t* buf = new uint32_t[sizeof(FPGA_DataPacket) / sizeof(uint32_t)];
```

After that the function enters a loop where multiple RxTSP module resets and sample checks are performed. In each iteration of the loop the following actions are performed:

- RxTSP modules are reset.
- Streaming buffers are cleared to remove leftover data
- Samples are read from the chip
- Samples from first and second channel are compared. If they are equal loop stops as channels are aligned.

```
       fpga->StopStreaming();
        dataPort->WriteRegister(0xFFFF, 1 << chipId);
        dataPort->WriteRegister(0x0008, 0x0100);
        dataPort->WriteRegister(0x0007, 3);
        dataWr[0] = (1 << 31) | (uint32_t(0x0020) << 16) | 0x55FE;
        dataWr[1] = (1 << 31) | (uint32_t(0x0020) << 16) | 0xFFFD;

        for (int i = 0; i < 100; i++) {
            dataPort->WriteLMS7002MSPI(&dataWr[0], 2, chipId);
            dataPort->ResetStreamBuffers();
            fpga->StartStreaming();
            if (dataPort->ReceiveData((char*)buf, sizeof(FPGA_DataPacket), chipId, 50) !=
sizeof(FPGA_DataPacket))
            {
                lime::warning("Channel alignment failed");
                break;
            }
            fpga->StopStreaming();
            dataPort->AbortReading(chipId);
            if (buf[4] == buf[5])
                break;
        }
        delete[] buf;
    }
```

Finally, the original chip settings are restored and function exits.

```
    {//restore values
        uint32_t dataWr[7];
        dataWr[0] = (uint32_t(0x0020) << 16) | 0xFFFD;
        dataWr[1] = (uint32_t(0x0400) << 16) | regsA[0];
        dataWr[2] = (uint32_t(0x040C) << 16) | regsA[1];
        dataWr[3] = (uint32_t(0x0020) << 16) | 0xFFFE;
        dataWr[4] = (uint32_t(0x0400) << 16) | regsB[0];
        dataWr[5] = (uint32_t(0x040C) << 16) | regsB[1];
        dataWr[6] = (uint32_t(0x0020) << 16) | reg20;
        dataPort->WriteLMS7002MSPI(dataWr, 7, chipId);
    }
}
```

3.4 AlignQuadrature()

AlignQuadrature() function attempts to correct 180 degree flip between quadrature generators.

At first it saves current chip configuration and loads RF loopback configuration for testing. Tx A channels is configured to generate DC test signal from TxTSP while Tx B channel is disabled. RF loopbacks for both channels are configured, however only Tx A channel is active so Rx B channel depends on the leakage to receive the test signal (Rx channel gains are set differently because of this). The reason for using only one Tx channel is because there can also be 180 degree flip in Tx quadrature generators. However, because of reliance on leakage, the function may not work correctly with some frequency and Rx path settings.

```
void Streamer::AlignQuadrature(bool restoreValues)
{
    auto regBackup = lms->BackupRegisterMap();

    lms->SPI_write(0x20, 0xFFFF);
    lms->SetDefaults(LMS7002M::RBB);
    lms->SetDefaults(LMS7002M::TBB);
    lms->SetDefaults(LMS7002M::TRF);
    lms->SPI_write(0x113, 0x0046);
    lms->SPI_write(0x118, 0x418C);
    lms->SPI_write(0x100, 0x4039);
    lms->SPI_write(0x101, 0x7801);
    lms->SPI_write(0x108, 0x318C);
    lms->SPI_write(0x082, 0x8001);
    lms->SPI_write(0x200, 0x008D);
    lms->SPI_write(0x208, 0x01FB);
    lms->SPI_write(0x400, 0x8081);
    lms->SPI_write(0x40C, 0x01FF);
    lms->SPI_write(0x404, 0x0006);
    lms->LoadDC_REG_IQ(true, 0x3FFF, 0x3FFF);
    lms->SPI_write(0x20, 0xFFFE);
    lms->SPI_write(0x105, 0x0006);
    lms->SPI_write(0x100, 0x4038);
    lms->SPI_write(0x113, 0x007F);
    lms->SPI_write(0x119, 0x529B);
    auto val = lms->Get_SPI_Reg_bits(LMS7_SEL_PATH_RFE, true);
    lms->SPI_write(0x10D, val==3 ? 0x18F : val==2 ? 0x117 : 0x08F);
    lms->SPI_write(0x10C, val==2 ? 0x88C5 : 0x88A5);
    lms->SPI_write(0x20, 0xFFFD);
    lms->SPI_write(0x103, val==2 ? 0x612 : 0xA12);
    val = lms->Get_SPI_Reg_bits(LMS7_SEL_PATH_RFE, true);
    lms->SPI_write(0x10D, val==3 ? 0x18F : val==2 ? 0x117 : 0x08F);
    lms->SPI_write(0x10C, val==2 ? 0x88C5 : 0x88A5);
    lms->SPI_write(0x119, 0x5293);
    double srate = lms->GetSampleRate(false, LMS7002M::ChA);
    double freq = lms→GetFrequencySX(false);
```

After loading the test configuration, the function enters a loop where search for channel alignment is performed. In every iteration of the loop the following steps are done:

- Phase offset between channels is obtained using GetPhaseOffset() (detailed in 3.2).
- If offset is between -90 and 90 degrees channels are considered aligned and the loop exits.
- If offset is too high, Rx quadrature generators are reset using RstRxIQGen() function (detailed in 3.5) and loop proceeds to the next iteration.

```
    dataPort->WriteRegister(0xFFFF, 1 << chipId);
    fpga->StopStreaming();
    dataPort->WriteRegister(0x0008, 0x0100);
    dataPort->WriteRegister(0x0007, 3);
    lms->SetFrequencySX(true, freq+srate/16.0);
    bool found = false;
    for (int i = 0; i < 100; i++){

        double offset = GetPhaseOffset(32);
        if (offset < -360)
            break;
        if (fabs(offset) <= 90.0)
        {
            found = true;
            break;
        }
        RstRxIQGen();
    }
```

When the alignment search loop finishes, the original chip settings are restored and the function exits.

```
    if (restoreValues)
        lms->RestoreRegisterMap(regBackup);
    if (!found)
        lime::warning("Channel alignment failed");
}
```

3.5 RstRxIQGen()

RstRxIQGen() function is used to reset Rx quadrature generators. It cuts off clock from RxPll LO to quadrature generators ad then toggles quadrature generator enable for both channels to reset them.

```
void Streamer::RstRxIQGen()
{
    uint32_t data[16];
    uint32_t reg20;
    uint32_t reg11C;
    uint32_t reg10C;
    data[0] = (uint32_t(0x0020) << 16);
    dataPort->ReadLMS7002MSPI(data, &reg20, 1, chipId);
    data[0] = (uint32_t(0x010C) << 16);
    dataPort->ReadLMS7002MSPI(data, &reg10C, 1, chipId);
    data[0] = (1 << 31) | (uint32_t(0x0020) << 16) | 0xFFFD;
    dataPort->WriteLMS7002MSPI(data, 1, chipId);
    data[0] = (uint32_t(0x011C) << 16);
    dataPort->ReadLMS7002MSPI(data, &reg11C, 1, chipId);
    data[0] = (1 << 31) | (uint32_t(0x0020) << 16) | 0xFFFD;             //SXR
    data[1] = (1 << 31) | (uint32_t(0x011C) << 16) | (reg11C | 0x10);    //PD_FDIV
    data[2] = (1 << 31) | (uint32_t(0x0020) << 16) | 0xFFFF;      // mac 3 - both channels
    data[3] = (1 << 31) | (uint32_t(0x0124) << 16) | 0x001F; //direct control of powerdowns
    data[4] = (1 << 31) | (uint32_t(0x010C) << 16) | (reg10C | 0x8);     // PD_QGEN_RFE
    data[5] = (1 << 31) | (uint32_t(0x010C) << 16) | reg10C;             //restore value
    data[6] = (1 << 31) | (uint32_t(0x0020) << 16) | 0xFFFD;             //SXR
    data[7] = (1 << 31) | (uint32_t(0x011C) << 16) | reg11C;             //restore value
    data[8] = (1 << 31) | (uint32_t(0x0020) << 16) | reg20;              //restore value
    dataPort->WriteLMS7002MSPI(data, 9, chipId);
}
```