Synchronize two LimeSDR

b.shubenok 1 18 September 2017 09:52

Hi!

I've faced a challenge - synchronize two independent boards distributed in space.
Assume both PCs have clocks synced to 50us. I need to achieve 200us synchronization. USRP, AFAIK, has two methods for such case: **set_time_now** and **set_start_time**. After setting current time sample time of each sample can be extracted easily.
Does LimeSDR have something like that?

1 Like

Time Synchronized Multi LimeSDR

Creating 4 channel coherent receiver

Reading from multiple LimeSDR-Mini v2 in same app

LimeSDR-Mini v2.2 with external clock

FFY00 2 18 September 2017 10:18

You can use one as the external clock if the other. Just connect the clock-out of one to the other's clock-in and configure it as an a external clock input.

b.shubenok 3 18 September 2017 10:33

They are 50m away from each other, so this may be the problem.

FFY00 4 18 September 2017 10:48

You are synchronizing them for communication with each other or for receiving an external signal?

b.shubenok 5 18 September 2017 10:49

Receiving the same signal from multiple locations.

FFY00 6 18 September 2017 11:09

Do you control that signal or it’s 3rd party?

---

b.shubenok 7 18 September 2017 11:11

No, I have no control over that signal.

FFY00 8 18 September 2017 11:21

That’s more complicated. I can’t really help you with that, sorry.

---

b.shubenok 9 18 September 2017 11:25

Thank you anyway! The easiest way to do this, as I think, is to sample at a given time on both boards. But the question is does LimeSDR support hardware timestamping with absolute timestamps and if not, how hard it would be to implement it.

Zero 10 18 September 2017 12:32

Are you using both TX/RX channels? Because one option would be to use the second TX/RX set to send a sync back and forth between the two transceivers.

9a4db 11 18 September 2017 12:35

Sounds like TODA task 😉
you can recognize the pattern of received signal from both…

---

b.shubenok 12 18 September 2017 12:44

This is not an option, unfortunately - receivers may hear desired signal but not each other. Also, I'm not allowed to talk in this part of the spectrum.

Zero 13 18 September 2017 13:08

I've been poking around in the FPGA recently. It would not be a big deal to change the timestamp clock from being reset by the start of the RX stream (as at present) to an external input. Then your only problem would be to sync that external signal to 50 uS - which even a cheap GPS receiver PPS can manage.

Next time I have the source open I'll see if any of the FPGA pins are available on connectors. I recall there being a GPIO spigot, but don't know offhand if that's connected to the NIOS (CPU on the FPGA) or to the LMS7002.

---

b.shubenok 14 18 September 2017 13:16

Thanks for sharing this! Would wait for your reply and will go to see the source code too.

---

cph 16 18 September 2017 13:58

This would be a great addition, especially if a GPIO output bit from one SDR could feed the 'reset' input GPIO bit on itself and on other SDRs.

andrewback 17 18 September 2017 20:33

> Zero:
>
> I recall there being a GPIO spigot, but don't know offhand if that's connected to the NIOS (CPU on the FPGA) or to the LMS7002.

There is a GPIO header which is connected to the FPGA and pins can be toggled/read via the API.

Zero 18 18 September 2017 20:51

Awesome - that means one can be re-purposed as an external sync input.

I would offer to do this myself, but I'm in the middle of several other tasks just now. In a couple of weeks I'll be back inside the FPGA (I think I'm going to need to make extensive changes for one of my applications). If you haven't taken care of this by the end of the month, reach back out and I should be able to do something.

1 Like

---

cmichal 19 18 September 2017 21:01

I have some related issues: I'm planning to have multiple boards that I would like timestamps synchronized on, but I'd also like to be able to have GPIO outputs change so they are predictably timed with the RF. I'm a complete FPGA novice - so my look through the FGPA source hasn't helped much on even trying to evaluate how achievable this might be. But it looks to me as though the GPIO setting and the TX/RX streaming take very different paths through the FPGA.

Is there a way to set GPIO outputs at specific timestamps?

---

ccsh 20 18 September 2017 22:12

> Zero:
>
> Awesome - that means one can be re-purposed as an external sync input.

> I would offer to do this myself, but I'm in the middle of several other tasks just now. In a couple of weeks I'll be back inside the FPGA (I think I'm going to need to make extensive changes for one of my applications). If you haven't taken care of this by the end of the month, reach back out and I should be able to do something.

Did you just offered to add 1 PPS sync feature (that would work e.g. with GPS or rubid generators)? That would be really awesome, this is actually the only really important thing that LimeSDR is missing for now!

I'm not an expert, but if it would be possible to route 1 PPS signal directly from external SMA connector to that GPIO input, it would be even better, then our rear panel will look just like in USRP (10 MHz + 1 PPS)

1 Like

Zero 21 19 September 2017 00:20

Thinking about this, I can do the hardware reset, but that's the least of what needs to happen.

Just having the receive timestamp go to zero on a PPS doesn't mean the high level software can tell what sample the PPS occurred on. The software only gets the timestamp once per readStream - you wouldn't know the clock was zeroed until you noticed the *next* buffer's timestamp wasn't what you expected.

You could work it backward and figure out what sample the PPS occurred on, but that would be a pain. Also, I don't know what error checking occurs in the driver - will it spot a suddenly changed timestamp? Will it care?

I'm willing, but a real PPS sync capability is going to take some thought.

For now, here's a kludge that would probably work: run the PPS (yes, the pulse) into the second receive channel. Try through a 20 or 30 dB attenuator at first.

You will see something in the receiver on the rising edge of the PPS - how much, and at what frequency, depends on the risetime of the pulse. That will let you correlate PPS to sample number, which should do for the originally stated application.

In the meantime, should we perhaps start a PPS hack thread?

Zero 22 19 September 2017 13:39

Improved kludge: use the PPS to drive an RF switch. Run a carrier out the second TX, through the switch, and into the second RX. When the PPS occurs you see the tone in the receiver; you can find the leading

edge and correlate the PPS with buffer sample number.

---

mitch 23 23 October 2017 22:57

> Zero:
>
> drive an RF switch. Run a carrier out the second TX, through the switch, and into the second RX. When the PPS occurs you see the tone in the receiver; you can find the leading edge and

Hi @Zero did you guys implement the external 10MHz reference in the LimeSDR ? We could really use that. We are using OAI and need external clock synchronization.

If not, what would it take to make the Lime accept the external clock from OAI? I can make the change on the OAI side but have less knowledge of the LimeSDR code.

Thanks,
Mitch

Zero 24 23 October 2017 23:46

The external reference input already exists, if all you need is frequency coherence. What's missing is an external time sync input.

I didn't implement it, but don't think it would be such a big deal. My thinking is that time syncronized sample reception/transmission could be achieved using the real time tracking counters in the RX/TX burst control hardware. Set the software up to send at some time (say, 1e20 ns) into the future - much to long from the present to happen due to elapsed time. Then, the external signal occurs (brought in via one of the GPIO lines) and sets the real time counters to 1e20 - and there goes/comes your burst.

Alternatively you could just zero the counters with the external input, and transmit at some time after zero - but there's a race condition. The counter is zeroed when the stream is opened, and starts running - it's possible you would get to your transmit time before you received your sync input.

I'd be happy to hack something together, but I'm not in a position to test it - my burst-driven code still doesn't work, and I still don't know why. (You don't know of any documentation that explains the 'C' api doing burst scheduling, by any chance?)

mpb 25 5 November 2017 12:25

Hi all,

I'm currently refactoring the output logic of **ODR-DabMod**, a Digital Audio Broadcasting modulator, which has separate UHD and SoapySDR blocks. The intention is to make all features available to both USRPs and the LimeSDR. These include a digital predistortion prototype (which needs feedback after power amplifier, implemented with continuous TX and occasional RX bursts, with relative timestamps), and the ability to sync the transmission to a specific GPS-derived time (for single-frequency networks, where several units are transmitting from different remote sites). The latter also needs an external disciplined REFCLK, which I understood is supported.

Long story short: I'm also interested in the ability to set the HW time on the PPS rising edge.

My code assumes system time is synced using NTP, and the second change (in system time) is close to the PPS edge. To summarise: software waits until the second changes (+ margin, say 100ms), and tells the device "on the next PPS edge set your HW time to current_second+1".

If I understand the discussion correctly, the LimeSDR doesn't support anything like that for now. Is anybody working on this already? Could I help in any way (FPGA/SW dev)?

Cheers
mpb

---

andrewback 26 6 November 2017 11:35

So you need a PPS input on the LimeSDR that can be used to set O/S time on the host? There is a GPIO header and API to access this, but I'm not certain that this is what you want.

myriadrf/LimeSuite/blob/master/src/examples/gpio_example.cpp

```
/**
    @file   gpio_example.cpp
    @author Lime Microsystems (www.limemicro.com)
    @brief  GPIO example
 */
#include <lime/LimeSuite.h>
#include <iostream>
#include <chrono>
#include <thread>
```

```
//Device structure, should be initialize to NULL
lms_device_t* device = NULL;

int error()
{
    if (device != NULL)
        LMS_Close(device);
    exit(-1);
}
```

This file has been truncated. **show original**

Adding in @Zack as he is best positioned to comment.

1 Like

---

mpb 27 6 November 2017 15:20

No, when I say “HW time” I mean the time inside the LimeSDR.

My TX signal is timestamped in absolute time, each sample has a “time of release” associated with it. So I need the LimeSDR to have absolute time. My OS knows the time (thanks to NTP), but I cannot just tell the LimeSDR “initialise time to X” because the latency of that command over USB is not predictable. [1]

That’s why I need the PPS input. I hope my previous question makes more sense now.

Cheers
mpb

[1]
Maybe that assumption is wrong, in which case I’d be curious to know how to set the HW time without a PPS signal.

---

mpb 28 28 November 2017 04:14

Hi again,

I've looked at the LimeSuite sources, and see that neither `ILimeSDRStreaming::Streamer::SetHardwareTimestamp` nor `ILimeSDRStreaming::Streamer::GetHardwareTimestamp` do any interaction with the device.

In the HDL, I cannot find any mention of a hardware time counter, neither do I understand what happens on the FPGA with the timestamp given in the `counter` field of `lime::FPGA_DataPacket`.

Before I continue reading: is there a hardware time unit in the FPGA? Where is it?

IgnasJ 29 28 November 2017 07:37

Hello,

> mpb:
>
> I've looked at the LimeSuite sources, and see that neither ILimeSDRStreaming::Streamer::SetHardwareTimestamp nor ILimeSDRStreaming::Streamer::GetHardwareTimestamp do any interaction with the device.

SetHardwareTimestamp - sets timestamp offset in software. It does not actually change timestamp in HW.
GetHardwareTimestamp - returns last received HW timestamp + offset (if set).

> mpb:
>
> neither do I understand what happens on the FPGA with the timestamp given in the counter field of lime::FPGA_DataPacket.

In Rx packet, it is the value of HW timestamp at the time when the first sample in the packet was received.
In Tx packet, samples in packet are sent to LMS chip only when specified timestamp value is reached. (synchronization to timestamp has to be enabled).

---

mpb 30 28 November 2017 09:23

Thanks for your answer!

> IgnasJ:
>
> In Rx packet, it is the value of HW timestamp at the time when the first sample in the packet was received.

> In Tx packet, samples in packet are sent to LMS chip only when specified timestamp value is reached. (synchronization to timestamp has to be enabled).

Where is this HW timestamp counter in the HDL design?
Can I reset this counter or set it to a given value at the exact instant a GPIO goes high?

IgnasJ 31 28 November 2017 15:04

> mpb:
>
> Where is this HW timestamp counter in the HDL design?

myriadrf/LimeSDR-USB_GW/blob/master/src/rx_packets/rx_pct_data_v2.vhd#L520

```
510  port map(
511        clk             => clk,
512        reset_n         => reset_n,
513        data_in         => diq0,
514        data_in_valid   => infifo_rd_sig,
515        sample_width    => sample_width_reg1,
516        data_out        => cmprsd_data,
517        data_out_valid  => cmprsd_data_valid
518        );
519
520        cnt_inst : LPM_COUNTER
521    GENERIC MAP (
522        lpm_direction       => "UP",
523        lpm_port_updown     => "PORT_UNUSED",
524        lpm_type            => "LPM_COUNTER",
525        lpm_width           => 64
526    )
527    PORT MAP (
528        aclr        => clr_smpl_nr,
529        clock   => clk,
530        cnt_en  => lpmcnt_cnten,
```

> mpb:
>
> Can I reset this counter or set it to a given value at the exact instant a GPIO goes high?

No, there is no such functionality, Currently, timestamp can be reset to 0 only from software (it is done before starting streaming)

---

mpb 32 1 December 2017 03:09

Thanks for the pointers! This counter lives in the RX path, I don't understand how it's influencing the TX path yet.

`lpmcnt_q` is only going to `pct_smplnr`, which only goes to `outfifo_data`, then straight through the `rx_path` to the `lms7_trx_top.bdf`, which I cannot parse easily without installing Quartus.

Am I missing a path from the counter to the TX path?

VytautasB 33 1 December 2017 10:59

> mpb:
>
> Thanks for the pointers! This counter lives in the RX path, I don't understand how it's influencing the TX path yet.

Actually there are two counters.

The first one counter that you have already found is included in RX transfer packets which are sent to PC trough USB.

You can track down wrrxfifo_wr signal which comes from rx_path and you will found similar counter in TX path (called - sample_nr_cnt_mimo). This counter is incremented when RX sample is captured and used to synchronize TX samples with RX samples. TX sample synchronization is done in tx_pct_data_mimo_v3 module.

---

mpb 34 25 December 2017 09:35

Thank you @VytautasB for the pointers. Why was this design with two counters used? Is this needed for some application? I was expected to find only one counter that represents time, I'm probably not seeing the whole picture.

For my application, I need the following two behaviours:

- The TX counter can be synchronised to an external event (1PPS rising edge) and counts all the time, even in case the TX stream is interrupted. The TX stream starts when the TX timestamp matches the counter.
- There is a continuous TX stream with timestamps, and from time to time an RX burst is done, and the RX packets can be aligned to the TX stream thanks to its timestamps.

Both are easy to achieve if there is a single always-running counter used for the timestamps.

What would be the easiest way to achieve these two behaviours?

---

ogun 35 27 December 2017 11:51

Is it possible to connect two limeSDRs to same SBC and use one of them to extend rx bandwidth and use one for transmit signals on gnuradio osmocom source by connecting their clocks together only?

---

cmichal 36 12 March 2018 23:40

Not sure how lively this thread is, but I was wanting something along the lines of the PPS reset of the timestamps discussed above, and did make a small change to the FPGA to do it. On the top level of the FPGA source (lms7_trx_top.bdf) I disconnected lte_clr_smpl_nr from the rx_path_top block. Then added an OR gate. The output of the OR goes to the clr_smpl_nr input of rx_path_top, the inputs of the OR come from lte_clr_smpl_nr and from FPGA_GPIO[0]. So now I can reset the timestamps with a pulse on GPIO0. If you don't want to use it, you can set that GPIO as an output (low), but if you do want to use it, set that GPIO as an input. Seems to work great.

In trying to get the rbf onto the board though I noticed that function in LimeSuite doesn't seem to work. LimeUtil does though.

Code-wise, its true you need to look for places where a timestamp is less than a previous timestamp, but that's actually way easier than the rf switch solution mentioned above (which I did also implement and made work - but what a pain - since gain changes with frequency quite badly in the range I'm using, and then you have to look at the magnitude of every sample to figure out where the pulse starts or stops).

In my application there is no tx happening when the timestamp is reset, though I start transmitting shortly afterwards and things seem to work correctly. I'm not sure how nice things would be if you were in the middle of transmitting when the timestamps suddenly reset.

Time Synchronized Multi LimeSDR

KarlL 37 22 March 2018 16:35

Does it work well? And how does it work exactly?

The 2 limes are receiving, then something (maybe one of the Limes) that is plugged to GPIO 0 of the 2 limes resets the counter on both at basically the same time (the same signal would be sent to both through a T), and so the next group of samples that is digitized will have a timestamp close to zero (probably not zero else it would mean the counter was reset at the beginning of a block of samples). From now on you can easily compare the samples from both Limes thanks to their shared timestamp.

Is this how it works?

@IgnasJ Any thoughts on this? Also if synchronization works well like this, and as it seems the changes at the FPGA level are minimum, could such changes be added to the official repo? Though thinking of it this would make 1 less available GPIO…

@cmichal Do you have a fork that others can use to test this? Or some detailed explanations for a complete FPGA novice on how to modify it?

A *LimeSDR Made Simple* on the FPGA, how it works, what it does, how it's linked to others things, would actually be really interesting.

---

cmichal 38 22 March 2018 18:03

What you describe is basically right -toggling the GPIO high then low will reset the timestamp counter, and you detect it by seeing that the most recent timestamp is lower than the one before. I think if you see a timestamp of 0 you shouldn't trust it, because it probably gets held at zero as long as the GPIO is high, so look at the timestamp of the next packet.

I think the only source file that's changed is the top level bdf. I've put mine at **http://www.phas.ubc.ca/~michal/lms7_trx_top.bdf**, and the binary at: **http://www.phas.ubc.ca/~michal/LimeSDR-USB_lms7_trx_HW_1.4.rbf**

If you open the .qsf, then browse the .bdf and search for the signal names I mentioned above, it should be easy to spot the or gate I added. The diff of the bdf seems very large, but I'm using a newer version of Quartus and it seems to have moved things around.

Time Synchronized Multi LimeSDR

KarlL 39 23 March 2018 00:32

Thanks a lot, I'll take a look at it.

---

mleech 40 24 March 2018 00:07

So, I didn't realize that LimeSDR supported timestamps AT ALL, let alone had any mechanisms that reset them. This is good news for people wanting to do coherence across multiple devices.

Quite apart from the external-trigger mechanism (1PPS, etc), there's also, in USB3, the ITP packet which is broadcast to all devices. The Isochronous Timestamp Packet includes a 27-bit sequence number.

So, one might be able to setup a scenario were the host says to all the attached Lime boards "hey guys, when you see ITP sequence number , start streaming".

---

andrewback 41 27 March 2018 08:46

That is an interesting idea. @Zack , curious as to your thoughts on how hard this might be to implement?

---

cmichal 42 19 April 2018 23:07

There seems to be some issue with my modification. I tested it fairly carefully looking at the synchronization of transmitted samples with respect to an external clock, and all looked good. I've just started looking at receiving now, and there seems to be a big issue. It looks like the latency between tx and rx varies each time you hit the GPIO timestamp reset button. The timestamp at which I receive an external event varies by ~1000 samples, seemingly independent of the sample rate (tested 10 MSPS and 25 MSPS).

It almost seems as though a packet in progress is discarded when the reset arrives? Any insight would be appreciated.

---

andrewback 43 20 April 2018 11:42

Tagging @Zack to see if he can perhaps advise.

---

cmichal 44 20 April 2018 17:02

That would be great. I spent some more time looking at the VHDL, and I think I'm starting to see why it doesn't work. There are two timers keeping track of the sample number - one appears to count the samples as they are received from the LMS7002M, the other appears to count them as they get loaded into packets. It looks to me like the former runs regularly, but the second is bursty. So they both get reset at the same time, but they didn't necessarily have the same values when they were reset.

I'm working on a new plan - instead of resetting the sample count, I'll try to report the time that the GPIO goes from L->H. When the GPIO is high, the time value that gets reported in the rx packet header will be the stored time from the timer that counts incoming samples. So that receiving software can tell the difference, I flip on the highest bit in the time word. At this point this strategy is functional, still need to test if it gives good synchronization.

I'm a *complete* novice at vhdl, so my interpretation of what's going on in here should certainly be read with appropriate caution.

LimeSDR USB GPIO input problem

---

cmichal 45 20 April 2018 20:06

So this new strategy seems to do the trick. Now my rx events start in the same place, give or take one or two samples, every time!

Changed source files are:
http://www.phas.ubc.ca/~michal/lms7_trx_top2.bdf and
http://www.phas.ubc.ca/~michal/rx_path_top.vhd
The output binary is: **http://www.phas.ubc.ca/~michal/capture_timer.rbf**

To use it, you need to hold GPIO0 high for at least one full rx buffer, and then watch for rx buffers with the leading bit set. Remove the leading bit and what's left is the last timestamp before GPIO0 went high. That same number gets transmitted in the timestamp field with every packet until GPIO0 goes low. There is some bit shifting happening in the timer in some circumstances I didn't fully understand, so it might be safest to check for a bit set in either of the top 2.

@KarlL and @mleech , if you're using the older version, you might want to give this a try.

2 Likes

LimeSDR-USB as an Oscilloscope

KarlL 46 21 April 2018 16:17

Thanks, I actually installed the Quartus software a few days ago, did the modification and tried it. Strangely, without doing anything to the GPIO 0, when receiving samples the timestamps were incorrect. Instead of having multiples of 34133 like 0, 34133, 68267, 102400, … I was having multiples of 133 and it was going back to 0 after reaching 11200 like so : 0, 133, 267, 400, 533, 667, 800, 933, 1067, 1200, …, 10933, 11067, 11200, 0, 133, 267, …

I did use the latest master where I would regularly see huge jumps in timestamps.

So with your new strategy, setting GPIO 0 to high sets the highest bit of the timestamp to 1, and setting it to low sets the highest bit back to 0? I was actually interested in doing exactly the same, as it allows to not lose the timestamp, while still knowing when the GPIO 0 was changed. I'll try it when I have the chance.

Did you use the latest code from the master branch?

---

mleech 47 21 April 2018 16:18

Carl:

I don't actually have multiple LimeSDRs, but this will be useful for a project iin Kelowna that I'm peripherally involved with. Once you have it where you want it, and it can be generalized it maybe should be picked-up by the devs here.

My thought is that one GPIO pin is used to enable this feature, and the other acts as a trigger. For normal use, without the "enable this feature" pin, the device would act normally.

What is your use-case? I'm guessing NMR related, but maybe not?

-Marcus

---

andrewback 48 22 April 2018 09:52

> mleech:
>
> it maybe should be picked-up by the devs here.

Pinged @Zack to see if it's something we can roll-in to a future update.

Zack 49 23 April 2018 06:40

Thanks. Will pick it up and consider to roll-in to a feature update.

KarlL 50 23 April 2018 07:44

> cmichal:
>
> That same number gets transmitted in the timestamp field with every packet until GPIO0 goes low.

Does that mean the Timestamp stays constant until GPIO0 goes low? Or it continues to run as normally, only with the leading bit set?

> mleech:
>
> What is your use-case? I'm guessing NMR related, but maybe not?

I'm actually playing with a switch and would like to know when the switch happened.

> Zack:
>
> Thanks. Will pick it up and consider to roll-in to a feature update.

That would be really neat to have that feature in the official FPGA.

cmichal 51 23 April 2018 18:20

> KarlL:
>
> I did use the latest master where I would regularly see huge jumps in timestamps.
>
> So with your new strategy, setting GPIO 0 to high sets the highest bit of the timestamp to 1, and setting it to low sets the highest bit back to 0? I was actually interested in doing exactly the same, as it allows to not lose the timestamp, while still knowing when the GPIO 0 was changed. I'll try it when I have the chance.
>
> Did you use the latest code from the master branch?

This is actually based on a snapshot of the FPGA repo from Feb 6, last commit is:
ab2a7a4a668cee5f47c6519bb4f013cf716533d2

I've had the best luck making things behave if I keep the buffer size to be an integral multiple of the native packet size. For data format I12, that's 1360.

The GPIO usage is what you said. Keep GPIO 0 for normal timstamps. When you set GPIO 1 high, the last timestamp seen before it went high (with the leading bit flipped on) gets transmitted for as long as GPIO 1 is high. When you set it low again, everything goes back to normal.

Did you try my image?

> mleech:
>
> My thought is that one GPIO pin is used to enable this feature, and the other acts as a trigger.
> For normal use, without the "enable this feature" pin, the device would act normally.
>
> What is your use-case? I'm guessing NMR related, but maybe not?

Yes, multi-channel NMR spectrometer. So the rf from multiple channels needs to be nicely synchronized, and also synchronized with logic events controlled by another device. Things are working reasonably well at this point, the start-up to start-up jitter is down to ~2 sample periods. Its almost good enough.

I've also been thinking about triggering - something a bit different from what you've mentioned though. The issue I'd be interested in addressing is getting the remaining jitter in the timing down even further. I think the source of it is that the decimation of the sample clocks on multiple boards are out of phase. What would be really great for me, would be some way of triggering streaming start-up so that the sample clock (CGEN) decimation on multiple boards were all in phase. Any advice? (@Zack ?)

mleech 52 23 April 2018 22:50

We have essentially the same requirement here. If one can do a “trigger streaming start” with an external GPIO or two (one to enable the functionality, one as the trigger), then with boards fed from a common REFCLOCK, phase-and-time synchronization should be really good.

The only issue will be that frac-N synthesizers even fed from the same clock can be up to +/- pi out of phase with one another at PLL lock time. Some devices have a phase-resynch signal that helps with this. Dunno about the LMS70002.

-Marcus

---

cmichal 53 23 April 2018 23:01

Marcus - how good synchronization do you need? My understanding of the sampling clock is: There is a sample clock VCO that always runs at a frequency in the range ~1900 - 2900 MHz. This VCO frequency is divided down by some even number (2(H+1)). For my purposes, I don’t care about the phase of VCO at all (which I think is what you’ve mentioned) - what I care about is the start of decimating that ~2.5 GHz down to my 10 - 25 MHz sample rate. Maybe that’s good enough for you too?

---

mleech 54 24 April 2018 04:03

I need good phase coherence among all of the receiver inputs. Fixed phase *offsets* of a sample or two aren’t that big a deal. But once you start getting larger offsets, then the array is essentially not pointing where you think it’s pointing, and as it gets worse still, you don’t get any useful beam at all.

My primary requirement is low mutual phase noise among all the receivers–this is largely accomplished by having the synthesizers all referenced to a single REF CLOCK.

The secondary requirement is low mutual phase-and-sample offset among all the receivers, where “low” is necessarily a bit fuzzy. One can calibrate phase offsets in the array using various techniques that may not be available for NMR work.

Just building a conventional N-antenna interferometer array, you need mutual phase coherence and low timing and phase offsets. I’m considering an experimental 4 x 4 FFT beamformer operating at 21cm, if I can get our University partner to cover the costs… They’re currently

using our 2-channel 21cm spectrometer, but a beamformer that is looking at 16 beams on the sky all at once would be quite a nice thing, and not necessarily very expensive…

---

cmichal 55 24 April 2018 06:33

ok, I think we're mostly after the same thing here. I would hope to get to the point where the synchronization between boards is a small fraction of a sample dwell. As far as phase synchronization of the actual rf signals themselves on different boards - I don't much care. I'm generally working at different frequencies on different boards and their phase coherence doesn't matter. I need the timing of the timestamps to be accurately known though.

For you its true - I think you will always have a phase issue amongst the rf signals on multiple boards. I don't know any way around that.

I spent a little more time looking at how the ADC and DAC sample clocks are generated, and syncing them between multiple boards looks tricky. I think I do see a path to doing it, but that may be just naive optimism from someone who doesn't know enough about fpgas to know better.

The DAC and ADC clocks both come from the divided down CGEN VCO. The barrier in my mind is syncing the divisors. The CGEN PLL seems to have two possibly relevant signals: RESET_N_SYNC and PD_FDIV_O_CGEN. The former sounds like it might be just the ticket, though it seems to reset quite a few pieces of the PLL, and I wonder if that means that the loop would drop out of lock? And if so would it lock again with the same timing on all boards necessarily?

The PD_FDIC_O_CGEN just powers down the divider, presumably leaving the VCO locked. Perhaps if you powered the divider down on all the boards and powered them back up simultaneously they'd all be in sync.

Unfortunately both of those signals are bits in registers in the LMS7002M, accessed by SPI, and not obviously easy to toggle from the FPGA. Perhaps one could have a module in the FPGA that, when triggered externally by GPIO, hijacked the SPI interface and bit-banged the instructions to toggle the RESET_N_SYNC line or the divider power? Unfortunately, this looks to be somewhat (ie, way) beyond my ability - but I'd be very interested in having it work.

One thing that I could conceivably try though, is doing this in software with very low sample rates. If the sample rate could be brought down to say 1kHz or so, then toggling those registers with the existing mechanisms seems likely to tell whether this strategy is worth pursing.

Are you working on CHIME? If so, it seems likely we know some people in common.

mleech 56 24 April 2018 14:20

No, I'm not working on CHIME. Would be fun, but, no.

I run this little adventure:

http://www.ccera.ca

---

tesch 57 14 May 2018 05:16

I made a branch with these files on github for easier reference: **https://github.com/myriadrf/LimeSDR-USB_GW/compare/master...tesch1:cmichal-sync**

The salient changes to the file lms7_trx_top(2).bdf weren't obvious from a textual diff, is there a pin remapping in there somewhere? Or is that file entirely generated? (I'm installing Quartus now… maybe the answer will be obvious if there is some graphical diff tool there…)

KarlL 58 14 May 2018 08:52

A different version of Quartus was changed and so even if you generate the bdf from the same code, you'll get very different bdf.

---

cmichal 59 14 May 2018 23:19

The changes in the bdf are pretty minimal. Just added one input to the rx_path_top module and hooked it to gpio pin 0.

I've realized why the variations in my synchronization are sometimes as big as two sample periods. With only one channel running, the timestamps only count every second sample. I have a plan to try to improve matters, but it will be a few days till I get the chance to try it.

1 Like

---

cmichal 60 30 May 2018 00:54

Some progress here. I've modified the sample timestamp capture so it is accurate to single timestamp precision. I should figure out how to do a pull request…

In some overzealous moments, I went further, and figured out how to synchronize the RX sample clocks in multiple boards to each other to within a few ns. I hijacked a GPIO input on each board and feed it with a clock oscillating at the ADC frequency (divided by 16), run that into an xor gate (inside the FPGA) and compare against bit 3 of the sample count. Then take the output of the xor and export it on another gpio pin. That output then goes into a low pass filter, and is measured as an analog value on an ADC of an arduino. Then by fiddling with the VCO frequency that generates the lime's ADC/DAC clock, you can bring the lime ADC clock into synchronization - sort of a poor man's pll.

If there is interest, I could post the changes needed to make this work.

Unfortunately, the situation for TX doesn't seem quite so perfect. Even with the ADC clocks on the boards synchronized to within a couple of ns as above, there is still some variation in the timing of TX output, that I believe is due to the dividers in the TX and RX clocks. For a 10 Msps sample rate, the CGEN clock runs at 80 MHz, and is divided down by 4, then by 2. I see timing variations now that span + to - 25ns in steps of 12.5ns. It feels like the dividers in the tx and rx chains are out of step with each other. Its not bad really - good to about 1/2 a sample dwell. The timing difference only changes when you fiddle with the sample rate. Once you set it, things stay synchronized.

1 Like

---

cmichal 61 1 June 2018 04:30

It turns out that calls to set the transmitter gain also mess up sync between TX and RX within the rf chip.

So - I've added one more XOR gate in the FGPA to compare the phase of the TX clock and the RX clock, and pipe that phase detected signal onto a GPIO. I don't have an great systematic way to correct that phase, but you can just reset it repeatedly till it settles at the position you want. My solution here only works if the DAC and ADC rates are identical - though with a bit of fiddling you could probably get something similar to work if they differed by a factor of 2 or 4. I am also using interpolation/decimation in the TSP of 2. If that factor was bigger this might get ugly.

At this point now, I have the DAC and ADC clocks synchronized on multiple LimeSDR boards with an external oscillator with reproducibility on the order of a couple of ns. You could probably do a bit better than that if motivated.

KarlL 62 6 June 2018 15:08

I'm using the modified FPGA file **http://www.phas.ubc.ca/~michal/capture_timer.rbf** .

@Zack @andrewback Do you know if there are any timing constraints on the negative timestamp?

I'm receiving packets of 128 samples or even 64, and want to detect whenever such packet has a negative timestamp. I'm using an Arduino to set and clear the FPGA[0]. If I only set the FPGA[0] for a very short time, eg ~3µs, I'll sometimes not see the negative timestamp (as if the FPGA[0] wasn't set for long enough to be detected), and when I do see the negative timestamp it would last too long (eg 10µs instead of the expected 3µs).

It's really important for me to handle short times and also to be sure that the timestamps behave properly.

KarlL 63 11 June 2018 13:12

@cmichal I've done other tests with the FPGA file you provided. I wonder if there is some delay in the timestamp becoming negative and positive.

I'm receiving 100M blocks of 128 samples at 60MS/s, and my Arduino is setting the FPGA[0] for 12µs then clearing it 34µs. I'm not expecting the Arduino to be infinitely accurate, but I would expect to see more or less the same number of blocks with negative timestamps, corresponding roughly to 12µs, and same for positive timestamps corresponding to 34µs.

But then I'd see 5, 6, 10 or 11 negative blocks in a row (I'd expect around 5 and 6), then 15, 16, 21, 22 positive blocks in a row (I'd expect around 16). I find it strange to see increments of 5.

If I lower the time when the FPGA[0] is set, then I would miss some switch between positive and negative, as if the timestamp was eventually never modified (note that at 60MS/s I should receive a block of 128 samples in less than 3µs, so waiting 10µs should be more than enough).

Do you have any idea what could be the problem?

---

cmichal 64 11 June 2018 17:56

I do have a guess. The FPGA packs samples into packets of 1360 samples. If you're using a buffer size of 128, several of those are coming from each packet. The timestamp only gets updated in the header of each packet. Are you receiving on both channels? If so, then the 5 block increment makes sense, since you get 5.3 of your blocks out of each 1360 sample packet.

I'd suggest trying a block size of 680 if you're using two channels, or 1360 if just one.

I'm curious about what the sequence of timestamps looks like. Can you paste in a list of: last few positive timestamps before negative, then the list of 21 or 22 negative ones?

LimeSDR-USB as an Oscilloscope

KarlL 65 12 June 2018 03:02

Thanks, I thought there might indeed be something like that (though I thought reading something about 1040 samples). I'm using 2 channels so I'll try with block sizes of 680.

With my previous tests I would have up to 11 negative timestamps. The positive timestamps go up by 2133 or 2134 each time (128 samples at 60MS/s). I get something like:

```
1761202933
1761205067
1761207200
1761209333
-6148914689475294922
-6148914689475292789
-6148914689475290655
-6148914689475288522
-6148914689475286389
-6148914689475284255
-6148914689475293455
-6148914689475291322
-6148914689475289189
-6148914689475287055
-6148914689475284922
1761234933
1761237067
1761239200
1761241333
```

BTW we have `1761234933 = 1761209333 + 12 * (128*1000/60)` so we correctly go back to positive timestamps as if nothing ever happened.

The negative timestamps themselves don't make much sense as is though.

KarlL 66 12 June 2018 03:41

The 680 samples matches what I've seen previously: 680 samples are received in 11.333µs, and the lowest I could leave the FPGA[0] set was 12µs. Below that I'd sometimes not see negative timestamps.

I tried setting the FPGA[0] for 12µs then clearing it for 36µs, and I get 1 or 2 negative blocks and 3 or 4 (sometimes 5) positive ones. It makes sense as the timing of the set and clear can't be perfect.

---

cmichal 67 12 June 2018 17:56

A lot of this makes sense. In the list of timestamps you posted, the negative ones increment by the same 2133/2134. The incrementing must happen in the LimeSuite library on the host computer. Then half way through the negative time stamps, they jump back again - since another packet is received that contained the same negative timestamp as the first one. Everything is a bit confused though because of the misalignment of the 128 sample buffers with the 680 sample packets.

So for the negative timestamps to make sense, we need the buffer size to be aligned with the native packet size (which actually depends on the data format. Its 1360 samples for 12bit samples, but 1020 for 16 bit samples). Its ok if you use a buffer size that is an integral multiple of the packet size. The pulse on GPIO0 needs to be at least one buffer size long, and yes, you will sometimes see one negative timestamp and sometimes two.

I'm confused though about why your stamps increment by 2133/2134. I would have expected the timestamps to increase simply by 128 each time? Are you doing the multiplying by 1000/60?

If so, by doing that, you make it much harder to figure out when the GPIO actually went high.
If you take out the leading bit of: -6148914689475294922, you get a number that doesn't make any sense as a timestamp. If you go back to what I would expect the raw timestamps to be:
105672176 (your first one multiplied by 1000/60)
105672304
105672432
105672560
First negative was then approximately at stamp 105673328. If you then set the msb: (1<<63) that gives
-9223372036749102480
which, if you then /60*1000 gives: -6148914689475295072 which is in the ballpark of your first negative number. Going backwards is going to be really tough.

I'd suggest not multiplying by 1000/60 till after you've sorted out the negative timestamps, then you can just remove the leading bit, and what's left tells you the correct time value.

KarlL 68 13 June 2018 00:57

Tanks a lot again for all the explanations.

In dual channel, 12 bits samples, with blocks of 680 samples, I indeed see constant negative timestamps, and it now makes sense that for smaller blocks the timestamps changes a bit.

Concerning the timestamps I showed, I'm just using the Soapy interface that returns `the buffer's timestamp in nanoseconds`:

pothosware/SoapySDR/blob/master/include/SoapySDR/Device.h#L359

```
349 * The implementation control switches or halt data flow.
350 *
351 * The timeNs is only valid when the flags have SOAPY_SDR_HAS_TIME.
352 * Not all implementations will support the full range of options.
353 * In this case, the implementation returns SOAPY_SDR_NOT_SUPPORTED
354 *
355 * \param device a pointer to a device instance
356 * \param stream the opaque pointer to a stream handle
357 * \param flags optional flag indicators about the stream
358 * \param timeNs optional deactivation time in nanoseconds
359 * \return 0 for success or error code on failure
360 */
361 SOAPY_SDR_API int SoapySDRDevice_deactivateStream(SoapySDRDevice *
362    SoapySDRStream *stream,
363    const int flags,
364    const long long timeNs);
365
366 /*!
367 * Read elements from a stream for reception.
368 * This is a multi-channel call, and buffs should be an array of vo
369 * whose each pointer will be filled with data from a diffeent cha
```

Looking further, it seems SoapyLMS7 converts the FPGA timestamp into nano seconds:

myriadrf/LimeSuite/blob/master/SoapyLMS7/Streaming.cpp#L418

```
408 return (status >= 0) ? status : SOAPY_SDR_STREAM_ERROR;
```

```
409 }
410
411 int SoapyLMS7::writeStream(
412 SoapySDR::Stream *stream,
413 const void * const *buffs,
414 const size_t numElems,
415 int &flags,
416 const long long timeNs,
417 const long timeoutUs)
418 {
419 auto icstream = (IConnectionStream *)stream;
420 const auto &streamID = icstream->streamID;
421
422 //input metadata
423 StreamChannel::Metadata metadata;
424 metadata.timestamp = SoapySDR::timeNsToTicks(timeNs, sampleRate);
425 metadata.flags = (flags & SOAPY_SDR_HAS_TIME) ? lime::RingFIFO::SY
426 metadata.flags |= (flags & SOAPY_SDR_END_BURST) ? lime::RingFIFO::
427
428 //write the 0th channel: get number of samples written
```

---

cmichal 69 13 June 2018 05:03

Great. Then I think it all makes sense. I'm not sure what's best to suggest in terms of finding what time the GPIO trigger occurred though - the conversion to ns messes things up quite a bit.
I'd be tempted just to hack soapy to not do that - replace that call to ticksToTimeNs with just:
timeNs = metadata.timestamp; Otherwise the backwards conversion is going to be pretty ugly.

KarlL 70 6 August 2018 09:09

Hi, I'm using also an Arduino to pilot the GPIO of the Lime. The problem is that they don't use the same pin sizes, so I had to hack an existing cable, and the result is not really pretty.

What Arduino are you using? And were you able to find a ready made cable to plug your Arduino and Lime?

Thanks.

---

cmichal 71 6 August 2018 13:08

My solution is not pretty. I bought a cable that has the right connector for the GPIOs on both ends and cut it in half. There are things like this; **https://www.adafruit.com/product/2094**
that might make it prettier. I have some vague recollection of spotting a cable with a 10 pin 0.05" connector at one end and an 0.1" 10 pin connector at the other, but even that wasn't going to be great.

Timed /synchronized GPIO pins for external device control

Omnigamer 72 20 August 2018 15:34

Hello, I'm curious if you ever found a reasonable solution for your issue @KarlL ? I'm trying to do something similar.

I'm still digesting all of the details given throughout this thread, but the timers and timing conversions in both HW and SW seem to make it messy. If the timestamp handling leads to some inconsistency, is there possibly a different way to set it up so that samples are only added to packets when a trigger signal is active? The most blunt way of doing this doesn't seem to be workable (disabling the LMS RX hardware entirely; the pin is not accessible), but it might be possible with some shenanigans in the rx_path logic. I briefly tested some, but there's a lot I didn't understand at the time about how it sets up the packets. The above discussion helped that, at least!

For my purposes, I only really care about samples that occur during an external trigger signal; the rest can be dumped, and the hardware state reset and held to a neutral configuration until the next trigger. Essentially, I want the LimeSDR to act like an oscilloscope. I'll do some more fiddling this afternoon and see if the packet timing state can be easily reset.

KarlL 73 21 August 2018 00:48

I'm not sure what you're trying to achieve. For my purposes, it's fine: whenever I see that the timestamp becomes negative, it means the external signal was triggered. If you only have one LimeSDR, this could be useful to know something happened, if you have several LimeSDR, you can use that to synchronize them quite precisely.

I don't really care about the real timestamp when the timestamp is negative.

I work in dual channels, so I receive by blocks of 680 samples (see **Synchronize two LimeSDR** for why).

Omnigamer 74 21 August 2018 15:39

Sorry I didn't mention it in my earlier post, but I'm looking to maximize bandwidth as much as possible. For that case it is beneficial if I can prevent adding samples I don't care about to the queue, which allows a higher sampling rate to not get bottlenecked by USB3 comms… in theory, anyway. Messing with timestamps on top of that might be useful for downstream processing, but is probably not required.

KarlL 75 24 August 2018 09:01

I installed Quartus Prime Version 17.1.0 Build 590. I simply opened the project `lms7_trx.qpf` then in `Project Navigator`, I chose `Revisions` and clicked `Compile All`. The compilation went through, but when testing the `rbf` file, it does not seem to be working, ie I never see the timestamp become negative.

The rbf file is around 560kB while the once provided by @cmichal is around 564kB.

Were you able to compile it and use it? Are there other things to do than just click `Compile All`?

---

cmichal 76 24 August 2018 12:55

I haven't tried to build from that repo, but have done more work for my application, and "packaged" my patches a bit. If you visit: **http://www.phas.ubc.ca/~michal/LimeSDR** there's a link to two versions of the patches. One is similar to what was posted above, but has some additional modifications: as long as you don't need to use other GPIOs, you can completely ignore those other modifications.

The other patch (labelled untested) does all the same stuff, but is to be applied to a much newer version of the gateware. There was a pretty substantial reworking of the top level of the gateware at some point, and I adjusted my patches. Those files have readme's in them that say what commit of the gateware they apply cleanly to.

I didn't have a chance to test the later patches thoroughly - but I believe the situation is that the "untested" gateware worked, but there was a problem in the LimeSuite that was from that same time period. So you could use the gateware built from the newer patches, but might want to stick with the older LimeSuite. Or just stick with the earlier versions.

There are some build instructions in the readmes - but basically yes, I just hit Compile All in Quartus.

I should add that in the newer versions the patches are a bit more readable because the graphical toplevel layout is all gone, all of the source files touched are vhdl.

---

mc955 77 14 September 2018 17:59

So I am trying to do the same thing here with a LimeSDR Mini. I have just started my final year project at university where I am trying to design and build a system capable of tracking the 3D flight path of 'an object' by listenening to the objects telemetry packets from multiple locations and then computing the cross correlation of the IQ streams from the independant recievers to work out the TDOA, and hence the location of the object. The system therefore requires multiple independant SDRs to be syncronised easily - ideally to within 1 or 2 samples at 30MS/s.

In order to do this the obvious choice is to use the 1Hz PPS output from a GPS module as an external trigger to syncronicse the SDRs as described above.

The gateware for the Lime mini has a slightly different top level setup but is very similar to that of the Lime USB so I have been able to succesfully modify the gateware to implement the external trigger functionality. The PPS is fed into a GPIO pin and as above the sample index of the 0 to 1 transition is embedded in the USB packet header, with the MSB set as a flag for the PC.

With this flashed to the FPGA I also modified the file 'streaming.cpp' in LimeSuite/SoapyLMS7 and rebuilt so that the raw timestamp from the USB packet is returned to the user application.

```
//timeNs = SoapySDR::ticksToTimeNs(metadata.timestamp, sampleRate);
timeNs = metadata.timestamp;
```

Next a python script was written to look for packets with a modified timestamp and then report the number of samples between the current and previous sync events.

```
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="lime")
```

```
sdr = SoapySDR.Device(args)

#apply settings
sdr.setSampleRate(SOAPY_SDR_RX, 0, 30e6)
sdr.setFrequency(SOAPY_SDR_RX, 0, 868e6)

#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]*1020, numpy.complex64)
```

At 30MS/s with a 40us wide PPS pulse the output from this script was as follows:

```
matt@ubuntu:~/Documents/iib-project/lime-sdr$ python3 delta-test.py
{addr=24607:1027, driver=lime, label=LimeSDR Mini [USB 3.0] 1D3AC940C7E517,
name=LimeSDR Mini, serial=1D3AC940C7E517}
[INFO] Make connection: 'LimeSDR Mini [USB 3.0] 1D3AC940C7E517'
[INFO] Reference clock 40.00 MHz
[INFO] Device name: LimeSDR-Mini
[INFO] Reference: 40 MHz
[INFO] LMS7002M calibration values caching Disable
[INFO] Rx calibration finished
25030102
30000018
30000018
30000018
30000018
30000018
30000018
30000018
30000018
30000018
30000018
30000018
30000018
```

Now this seems to suggest that everything is working as intended, however there is somthing that is confusing me.

Why does this only work when I read the samples 1020 at a time? If I set the buffer size to 1360 then it all goes quite wrong. I see from the **stream protocol** that 16 bit samples are sent 1020 to a packet - does the SDR automatically choose the largest bit depth that it can happily spew out within the USB bandwidth?

From looking at the VHDL and the FT601 datasheet it is not obvious what is going on so if somebody could shed a little bit of light on the specifics regarding the USB packets on the lime mini that would be really helpful.

In addition I am happy to share the modified gateware should anyone want it. Big thanks to @cmichal for your work on the gateware for the Lime USB.

Matt

---

cmichal 78 14 September 2018 18:22

It doesn't surprise me at all that things go bonkers unless you set your buffer size to a multiple of the FPGA packet size. As soon as its not a multiple, then the library has to fiddle with the timestamps from the FPGA, and I'm sure the negative numbers get badly confused.

The number of samples you can fit in a packet does depend on the data format. I've been using the 12bit packet format (LMS_FMT_I12) , which gets 1360 samples per packet. If you use the 16 bit format you only get 1020 per packet. You should be able to set the buffer size to 2040 or 4080 and have things work ok.

In the LimeSuite library, you can choose the data format in the dataFmt field of the stream, which you can set before calling LMS_SetupStream.

---

mc955 79 14 September 2018 19:08

Do you have an example program that sets up a stream using the LMS_FMT_I12 data format and then looks at the timestamp of each packet, making use of the LMS API? I'm struggling to find a decent example online.

---

cmichal 80 14 September 2018 19:12

There are a couple of examples included in the LimeSuite source code. dualRXTX.cpp is a useful starting point.
You can find them here:

> **myriadrf/LimeSuite**

Driver and GUI for LMS7002M-based SDR platforms. Contribute to myriadrf/LimeSuite development by creating an account on GitHub.

---

mc955 81 14 September 2018 21:40

Thanks I'll take a look at that. I was looking through the documentation, including the pdf on compilation, and I cannot seem to find instructions on how to compile the program using g++? Where can I find the apppropriate command line instructions?

---

cmichal 82 14 September 2018 21:52

The easiest thing, I think, to do is to install libLimeSuite.so in /usr/local/lib
(on linux it seems to need to have two symlinks, so there is libLimeSuite.so, libLimeSuite.so.18.01.0 and libLimeSuite.18.01-1, or whatever version numbers you've got, all pointing to the same file).

Then there are two include files that need to be found at compile time:
LMS7002M_parameters.h and LimeSuite.h.
If you drop the first into /usr/local/include and the second into
/usr/local/include/lime
Then:
g++ dualRXTX.cpp -lLimeSuite -o dualRXTX

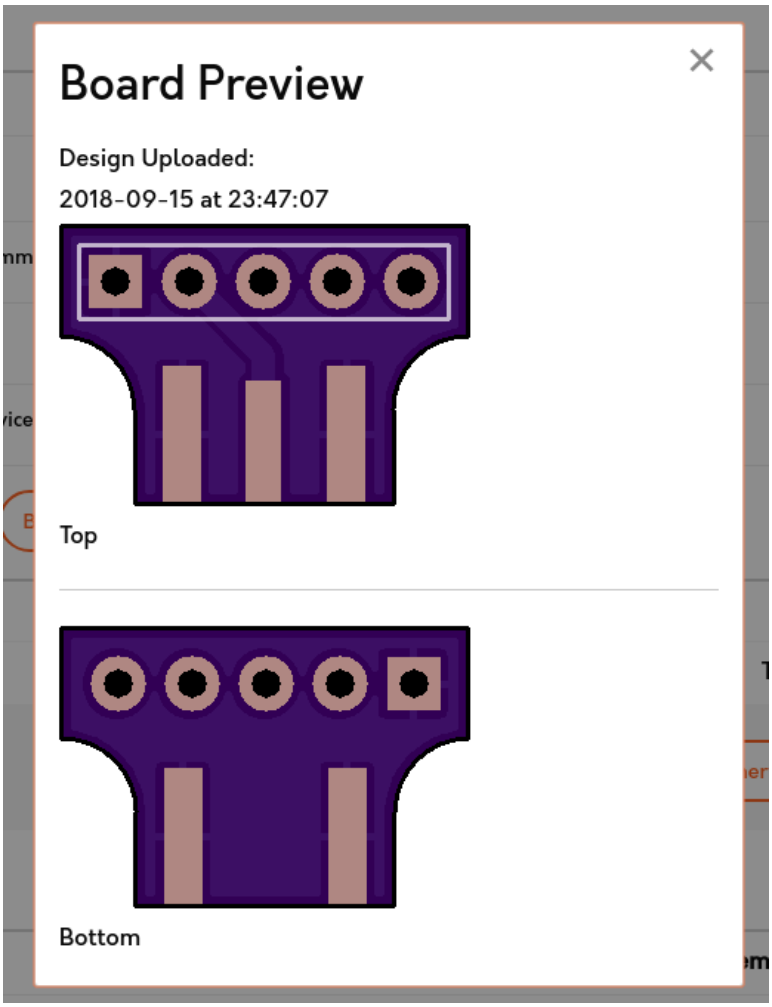
will do the job.

---

mc955 83 16 September 2018 00:01

Thanks, that worked a treat.

In terms of cables I have designed a tiny PCB that solders onto the LimeSDR-Mini's 0.1" GPIO header and then exposes an SMA connector allowing easy connection to the PPS output of a GPS unit via a standard SMA cable.



If anyone would like a board then let me know (UK only).

---

mc955 84 16 September 2018 18:33

@cmichal - When sampling at 30.72MS/s using the I16 data format, the function LMS_GetStreamStatus reports around 1077952577 dropped packets each second. I assume this is just the driver being confused by the modified time stamps as the link rate seems to stay the same at 123.404 MB/s when I disable the PPS sync functionality (e.g. hold gpio 0 low). I can't quite work out however where that number comes from given the increase in the time stamp of 8x16^15 and the sample rate and number of samples per packet.

---

cmichal 85 18 September 2018 16:29

I'm afraid I can't help you on that one. I don't think I've looked at the dropped packets field - I tend to just watch the packet timestamps. It isn't obvious to me where that number is coming from. I expect you're right that it is just the driver subtracting something that's negative - and maybe it gets turned into a 32 bit number

somewhere? I expect a dive into the LimeSuite source looking for where that field gets set would illuminate it somewhat.
Good luck!

---

mc955 86 21 September 2018 23:08

@cmichal Yeah looks like a mess, I won't worry about it.

On a separate note, have you or anybody else used this method to trigger a TX? What would be very useful for my application is to also be able to have TX triggered by the PPS pulse, allowing time of flight measurement.

With the work we have done so far we know the index of the rx sample corresponding to the 0->1 transition of the trigger. Now in the case of the TX stream the timestamp in the metadata is defined as ' time when the first sample in the submitted buffer should be sent'. My question now I guess is are these the same counters? From looking at the vhdl I think this is the case, but it isn't immediately clear. From what I can gather in the **Limesuite source code**, when the first stream is opened the FPGA is setup for streaming, which I guess starts the counter running, and from then on isn't reconfigured or disabled provided there is one active stream.

That would suggest you should be able to:

1. Enable RX channel
2. Enable TX channel
3. Setup RX stream
4. Record index of PPS event
5. Setup TX stream, then optionally stop & destroy RX stream, but leave channel enabled to tick counter
6. Send buffer of samples to SDR with a time stamp of PPS event + sample rate, so the packet is sent at exactly the next PPS event.

I'd be grateful if anyone with more understanding of how the synchronization works would be able to advise on this in some more depth.

---

cmichal 87 21 September 2018 23:32

Hi,

If I understand you, then the answer is yes, this can be done, and I've done it. Yes, the TX timestamp is based on the RX counter.

I have a digital pulse sequencer that produces a pulse and supplies it to the LimeSDR GPIO, causing capture of the timestamp. The start of that pulse is my t=0. I then produce pulsed output on the limeSDRs that are reliably synchronized with later digital pulse from the digital pulse sequencer.
I allow a fair bit of latency between the capture and the real start of TX (~200ms?), 1 second will be plenty.

My ordering isn't exactly what you have, it is:

1. Enable RX and TX channels
2. Set up RX and TX streams
3. Start receiving data on RX stream, waiting for GPIO/PPS event
4. After timestamp capture, send TX packets with timestamps at desired point in the future
5. continue (always) receiving on RX

This actually works very well. There are a few gotchas to watch out for, like it seems that you need to make sure that the TX buffer is empty before you send a TX packets with a discontinuous timestamp.

This works well down to single timestamp precision. If you want to do better than that, it can be done too, though its a fair bit more work.

I'm not sure if I tried disabling the RX stream or not in the SISO case. At some point I had both channels running and found that if you then disabled one of the channels, nothing good happened. I think you're best off just continuing to stream RX data, even if you're just throwing it away.

I've thought a tiny bit about getting rid of the latency - about having a mode where the timestamp counter doesn't increment until the GPIO signal is received so that you could transmit pulses one counter tick after the GPIO signal, but haven't looked seriously into implementing it.

---

mc955 88 22 September 2018 00:53

@cmichal Great to hear, that's exactly what I'm looking for.

For my application, I have a transmitter and multiple receivers, all Lime-Mini's. The source will be on a plane and the receivers scattered around an airfield. The latency between PPS occurring and transmission isn't a problem as I can just offset the two events by a known number of samples and adjust my t=0 accordingly.

Single sample precision is fine, the jitter of the PPS is about one sample period (30ns) so there would be no benefit to trying to go further.

In that case I'll leave the RX enabled - do I still need to read it from the stream with LMS_RecvStream or will the PC side (the one assigned in lms_stream_t) FIFO just overwrite itself and not complain?

Also finally would you just be able to clarify what you meant by ‘you need to make sure that the TX buffer is empty before you send a TX packets with a discontinuous timestamp’. Are you referring to the TX buffer in the FPGA or the PC side FIFO whose size is defined in lms_stream_t streamId? Either way how can you flush it to ensure it is empty before TX?

Thanks,
Matt

---

cmichal 89 22 September 2018 01:03

I think you probably do need to read the data with LMS_RecvStream - otherwise it will just pile up in a buffer somewhere - on the host computer I expect. It may not be totally necessary, but I think you’re less likely to find weird problems if you just read and throw away the data.

On the tx side, if you send a packet with a timestamp of 2 seconds, then immediately send another packet with a timestamp of 3 seconds (before the previous one has actually been transmitted), I think what happens is the second packet is transmitted as soon as the first one is finished - it won’t wait.

1 Like

---

mc955 90 24 September 2018 16:47

Thanks for your help, it seems to be working.

Interestingly as I am only using the TX stream to send a 8 buffers of 1360 samples once every second, nothing happens unless I set tx_metadata.flushPartialPacket = true;

In the comments of Lime.h it says ‘send samples to HW even if packet is not completely filled’. I assume this is referring to the 4096 byte packets sent over USB but I am confused because surely I should be able to completely fill 8?

---

mc955 91 24 September 2018 18:34

@cmichal Actually I take that earlier comment back, it seems to be sending late for some reason. Think it might be due to flushPartialPacket being set true. Any advice on how to use the TX stream in a bursty way without enabling this?

cmichal 92 24 September 2018 20:39

I thought when I looked into it, flushPartialPacket was only useful if your tx buffer size was not commensurate with the FPGA packet size. I have always kept my buffer size to be an integral multiple of the packet size, and so I think flushPartialPacket doesn't do anything.

When you say late - is it late by a fixed amount? Or does the delay vary? I wouldn't be at all surprised if there is some offset between the tx and rx times, but it should be fixed.

If you don't have the tx buffer size set to a multiple of the packet size, I strongly recommend it.

---

mc955 93 24 September 2018 21:17

@cmichal So the delay doesn't seem that consistent, however I have noticed something else that seems to be a bigger problem.

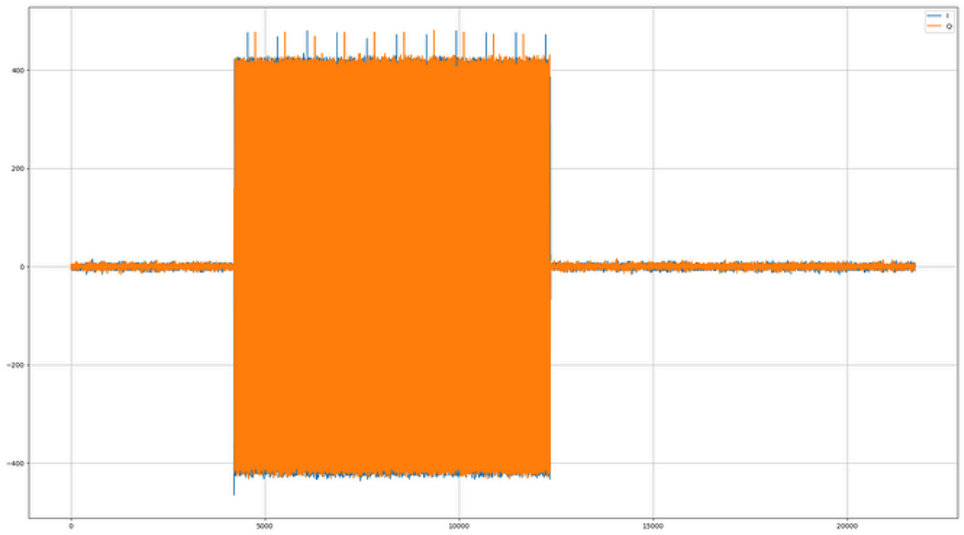
Firstly here is a link to the code I am running - **https://github.com/mattcoates/iib-project/blob/master/software/pps_sync_tx/main.cpp**

TX and RX are connected together externally via a 30dB attenuator.

An example output for one second is:

```
PPS sync occured at sample 136397134
Samples since last PPS = 30720016
Offset = 14
TX scheduled for sample 143197134
Capture scheduled for sample 143195760
popping from TX, samples popped 0/1360
RX Data rate: 92.5368 MB/s
TX Dropped: 0
Capturing at 143195760
```

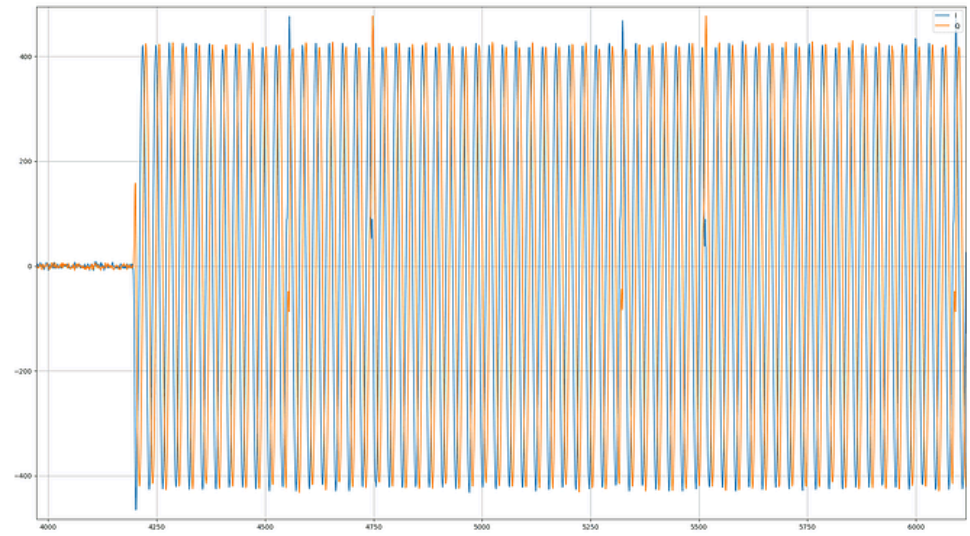
The resulting plot of the output file looks like this:



The output of the python plotting script is:

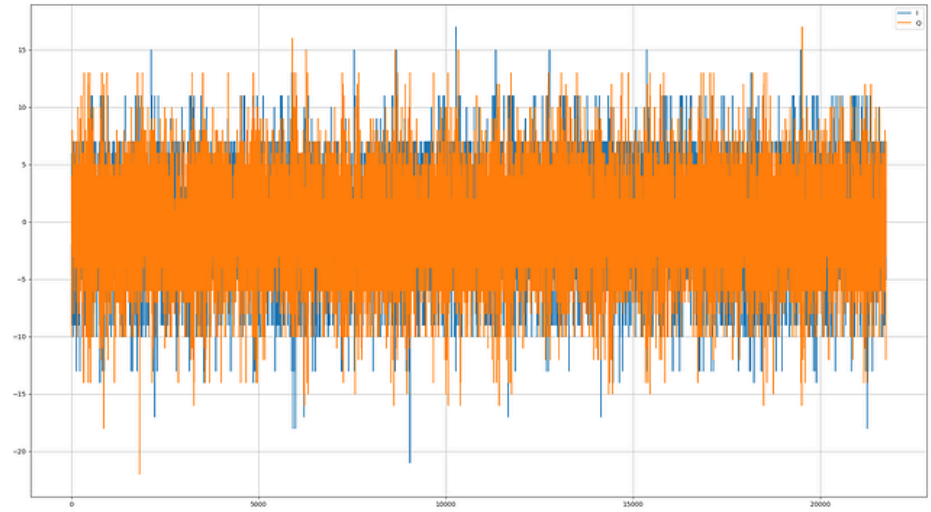
```
File contains 21760 samples.
Duration: 708.3333 us

2018-09-24 20:55:08
File begins with sample 143195760
PPS sync occured at sample 136397134
TX occured at sample 143197134
TX offset = 1360 + 14 = 1374
```

This adds up with the above output from the C++. The offset is stated as being 1374 samples. The below image however shows that the TX event does not occur until sample 4196. In addition whilst the TX buffer is 1360*8=10880 samples long, and all 10880 are sent to the stream, the burst in image 1 above only lasts from 4196 to 12360, which is 8164 samples. In addition you can see discontinuities in the output waveform below.



Writing the TX buffer out to file and plotting looks like this.

Now setting the flushPartialPacket to false and running the same code above gives the following output showing that TX has not occured.



In this case in the terminal each second the message `popping from TX, samples popped 0/1360` appears again.

I can’t seem to get my head around what is going on here, any help would be greatly appreciated.

---

cmichal 94 24 September 2018 22:08

I don’t think I’ve seen messages about TX samples being popped - so that’s a bit weird. There are two things jumping out at me, neither of which feels likely to address your immediate problem:

1. make sure you don’t schedule TX to start on an odd-numbered timestamp.

2. if TX and RX are the same frequency, you should disable one of the PLL's. Maybe you're doing that somewhere I don't see? There's a post on the forum here **[LimeSDR-USB] Synchronization and Toggle TDD/FDD Mode** that gives a couple of lines of code to do it.

The only other thing that's coming to mind is that at some point in June I fetched the then-latest LimeSuite source, and immediately started having some weird synchronization problems. I fell back to the version I had been using from Jan 2. (commit e5cde28f53ab838a140ea32a67e3f6d569ff4bce).
I didn't pursue the problems, and haven't looked at the current code, but you might try that older version of the library.

KarlL 95 25 September 2018 00:47

> mc955:
>
> popping from TX, samples popped 0/1360

This should mean that TX tried to get samples to send, but couldn't get anything, as if you didn't give enough samples or you didn't give them fast enough.

---

mc955 96 26 September 2018 23:33

@KarlL I think this message may just be a result of the streamer thread that manages the fifo looking for some samples but not finding any, which makes sense because we send them very infrequently. This message only appears frequently during program execution if the `flushPartialPacket` flag is set to false.

@cmichal Thanks for the tips, I have added in the snippet to use only one PLL, however as expected nothing changed. May I ask why the timestamp must always be even?

With regards to the LimeSuite version, based off the git commit messages it looks like there have been lots of bug fixes for the Lime-Mini since January this year which makes me reluctant to revert back as it may introduce some errors elsewhere.

I have however made some progress. I flashed the original gateware back and made a short program to do the following:

1. Setup streams and receive a few buffers to allow everything to stabilize
2. Read a buffer of samples and record the timestamp
3. Send a buffer of samples with a timestamp 256 * 1360 samples in the future

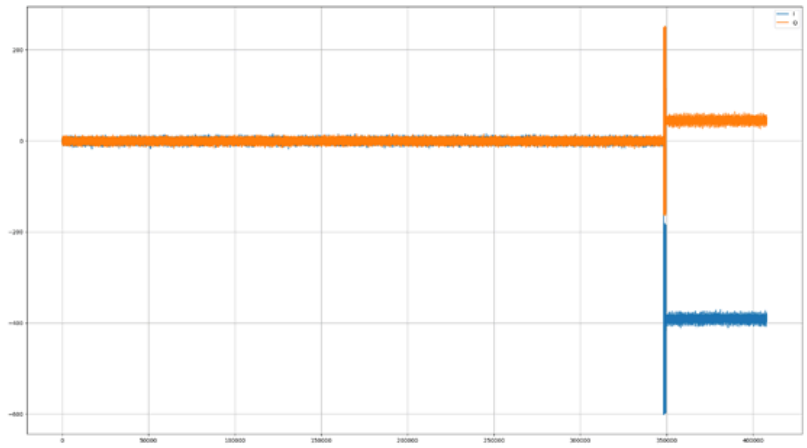
4. Record the next 300 buffers and plot the output

The code can be found here:
https://github.com/mattcoates/iib-project/blob/master/software/tx_testing/main.cpp

An example output is:

```
Devices found: 1 0: LimeSDR Mini, media=USB 3.0, module=FT601,
addr=24607:1027, serial=1D3AC940C7E517

Reference clock 40.00 MHz

RX Center frequency: 868 MHz
TX Center frequency: 868 MHz
Selected RX path 3: LNAW
Selected TX path 2: BAND2
Host interface sample rate: 30.72 MHz
RF ADC sample rate: 122.88MHz
RX LPF configured
RX LPF bandwitdh: 10 MHz
TX LPF configured
TX LPF bandwitdh: 10 MHz
Normalized RX Gain: 0.739726
RX Gain: 54 dB
Normalized TX Gain: 0.394737
TX Gain: 30 dB
Rx calibration finished
Tx calibration finished
RX Event at 199920
TX Scheduled for 548080
Delta = 348160
Final RX stamp: 606560
RX dropped: 0
TX dropped: 0
Device closed
```

The output graph is:



The `tx_buffer` is 1360 samples long. Here the burst lasts from sample 348260 to 349620 which gives a length of 1360 as required. I ran this program a few times, and in each case the TX event looked to start at 348260, which is 100 samples (or 3.26us) after the scheduled start of 348160. I wonder where this latency comes from, and if it can be predicted.

As before this only works if `flushPartialPacket` is set to true, even if I make the stream FIFO and `tx_buffer` the same size. I think I may have to look into the source to see what happens here.

Im also confused about the strange DC offset that has appeared in the output.

Anyway, I'll keep trying at it and see what happens. I have a suspicion that it may not be wise to shedule a TX event at some point when the most current HW timestamp has the MSB set, so I may move that a few packets into the future.

Thanks.

---

cmichal 97 26 September 2018 23:52

Does the DC offset correlate with turning off the second PLL?

I don't know your FPGA modifications, but for mine, the MSB is only set in a header for received data, the timestamps that get looked at for TX are read out upstream from there, and so should never see the MSB.

The odd-number issue is because in single channel mode the timestamp counter increments after two pairs of IQ samples are received, then to actually count the correct number of samples, the count gets shifted up one bit. So in that mode, the timestamp is always an even number.

Again, this is from the LimeSDR (not mini) FPGA, things may be different on the mini.

I think I have seen a latency of 3ish microseconds, similar to you, though with the LMS7002 configuration unchanged that's been constant.

I have always had flushPartialPacket set to False. Weird.

---

mc955 98 27 September 2018 00:42

I don't think it has anything to do with the PLL as I saw it before I added in the code to turn it off, and also I RX 1000 buffers from the stream and throw them away prior to reading a timestamp and sending the TX samples.

So my FPGA modifications are the same, but in either case the LMS driver's most current HW timestamp has the MSB flipped, and if the driver does some sort of sanity check before sending the packet to the FPGA it may discard or modify it.

The timestamp thing makes sense. Lots of the VHDL for the mini is the same as the full size version, it even has the original comments and references to the cyclone family…

Yeah its puzzling why `flushPartialPacket` must be set true, or even why it matters at all if our packet size is a multiple of 1360. Its really annoying that there isn't a loop back example for the LMS API that I can have a look at.

---

mc955 99 28 September 2018 19:33

So I think I have it working now. The reason why we were seeing a pulse that was shorter than 10880 was because the first packet of 1360 samples was being sent immediately after the SendStream call and the remaining 1360*9 were being transmitted at the correct time, so the first sample of the TX we see in our output is at tx_metadata.timestamp + 1360 + 100, which makes sense.

I had a feeling this was due to the streamer thread not liking being active for so long with nothing to send, so what I did to fix the issue was the following:

1. Start TX stream
2. Send TX samples with a timestamp 75 packets in the future (e.g. 1360*75)
3. Wait until the rx_metadata.timestamp was equal to the time when the TX pulse has been sent
4. Stop TX stream

This seems to work well now, and each second a buffer of 10880 samples is transmitted at exactly a predetermined number of samples after the PPS event. The code can be found **here** in case anyone is interested.

Finally, @cmichal you mentioned that in single channel mode, the sample counter is incremented only every other sample. I think this is occurring on the Lime-Mini too as when I try to time the number of samples between PPS events, I always get an even number even though there is some small variance due to the VCTCXO. Do you think there would be a way to modify the gate ware easily to increment the counter on every received sample?

---

cmichal 100 28 September 2018 20:33

@mc955 Nice going! I'm pretty surprised you have to do that - it seems very odd that there's something ok with having 75 packets delay with no data, but not longer… But maybe there's something more subtle going on. With my setup, I send some packets, leave long delays with nothing, the send a few more packets, and things seem to work. Though maybe what you've found is related to the problems I started seeing with a later LimeSuite library. I'll have to get back to that someday, though it will be a while.

For the question of odd numbered packets, I did find a solution to have odd numbered samples recorded. Then if I want transmit to start on an odd numbered packet, I just include a zero or three at the beginning of the TX packet, and set the packet to go 1 or 3 samples early.

The modification to the gateware I made is a little more invasive that what I posted earlier. Basically I've got a new register in the rx_path_top that keeps track of how many clock ticks have passed since the sample packet number was incremented. Then when the GPIO trigger is observed, you can tell if you're on an even or odd numbered sample. I tried to make this so it would work generally, but there appear to be several different modes (SISO, MIMO, DDR etc) for which there are several flags. And honestly from the documentation in the code, I couldn't quite resolve all of what all of those flags do.

You can see the changes to rx_path_top.vhd here:
http://www.phas.ubc.ca/~michal/Lime/rx_path_top.diff
and the complete file here
http://www.phas.ubc.ca/~michal/Lime/rx_path_top.vhd

There are some bits in there that you don't need. Anything involving trigger_seen, last_bit3, trigger_time, trig_pulse_duration, trig_out_pulse, trig_sync, trig_sync_sync, and div_clk shouldn't matter to you and can be removed. I'll apologize for the faulty indentation in this code - Quartus seems to mess it up whether I use tabs or spaces…

Controlling a SP4T from limemini

KarlL 101 3 October 2018 10:30

I took the time to retry compiling your 1st version and it does work. I guess I must have forgotten to update some files the previous time I tried.

I just found some small mistakes in the `readme.txt`:

It should be
`cp /path/to/rx_path_top.vhd src/rx_path_top/rx_path/synth/`
instead of
`cp /path/to/rx_path_path_top.vhd src/rx_path_top/synth/`

Also instead of opening `LimeSDR-USB_lms7_trx.qsf` I opened the project file `lms7_trx.qpf` and built it.

I've also compiled your 2nd version and the synchronization seems to be working too. I'm using my own C++ code, not LimeSuite.

Thanks again for sharing your code.

KarlL 102 3 October 2018 10:50

I'm using @cmichal modified gateware and it works great. I should have access to a mini so I was thinking of testing your modification too (from **LimeSDR-Mini-PPS-Sync-GW**)

From what I understand, your modification seems to be working similarly to the one for the LimeSDR (provided that you use the correct buffer sizes, 1360 in sc12, 1020 in sc16). The only problem you have is when you try to send something in the future, the 1st packet is sent at once while the others are sent at the right time?

Thanks.

---

mc955 103 3 October 2018 11:30

@KarlL Correct, the modifications are the same as those made for the LimeSDR by @cmichal above. EGPIO0 is used as the signal source, and take care to make sure the pulse remains high for at least one packet duration - nothing bad happens if you keep it high for a few so I'd just make it 200us or so.

I was experiencing problems with TX'ing in the future as you said, however in the **latest commit** of my software (that uses the LimeSuite API), this has been fixed. I managed to fix the problem by stopping the stream and then starting it again in between long periods of 'silence'.

Let me know how you get on. I've had a few days off but today I'm going to have a look at getting the counter to increment on every sample in the mini in order to double the timing accuracy. Considering the LimeMini is only a SISO device you would have thought it would have done this anyway, but I guess it has been left as before for consistency.

KarlL 104 4 October 2018 00:27

Thanks @mc955 . I indeed make sure my pulse stays high long enough so that I don't miss it. I usually see it's high one block, sometimes 2. I'm using an Arduino to generate that pulse, and the timing is quite constant, or at least constant enough.

Just to clarify things. Is the issue with the transmission in the future only in your version of the gateware, or also in the official one? If it's in the official one, we should mention it to the team. It would be more convenient to just be able to leave the stream open. Also stopping then starting the stream will probably take some time.

I'll try to test your GW soon and get back at you then.

---

mc955 105 4 October 2018 10:27

@KarlL The modifications I made to the gateware were very minor and only affected the timestamp reported by the SDR in RX. I think if anything is at fault here, it is most likely the Lime suite library as I haven't made any changes to the TX path in the VHDL. When I was initially testing to just get sending at a certain sample working I was using the stock gateware and the latest commit of the LMS API, however I sent the packets more frequently so didn't notice any issues. It was only when I moved to using the PPS to send once per second that I saw problems.

@cmichal I've had a good look at the code you posted and it seems to make sense. I am assuming that here `inst0_fifo_wrreq` is a signal that goes high when a new sample is put into the FIFO?

The other thing is I can't quite see how you arrived at this expression for selecting the mode:

```
my_mode <= "00" when (mimo_en_sync = '0' and ddr_en_sync = '1') else
"01" when ((mimo_en_sync = '1' or trxiqpulse_sync = '1') and ch_en_sync =
"11") else
"10" when ((mimo_en_sync = '1' or trxiqpulse_sync = '1') and (ch_en_sync(0)
xor ch_en_sync(1)) = '1' ) else
"11";
```

I think this may just be down to the confusing/poor comments and lack of documentation for the VHDL but if you'd be able to clarify a bit more the difference between SDR/DDR/TRXIQ Pulse that would be really useful.

Thanks,
Matt

EDIT: I've had a good look at the VHDL and I think I agree with what you've done. I believe the table at line 43 in txiq.vhd is incorrect as it contradicts there own code, so I have opened a **github issue** for it.

May I ask how you deduced how many clock cycles there are per increment of the counter in each of the modes?

---

cmichal 106 4 October 2018 18:48

I think I copied/adapted that from some other source file. I spent a few hours combing through a few places in the vhdl where those various modes affected things - and looked at things like under what conditions the timestamp counter was bit shifted. My recollection was that the comments documenting things weren't entirely self consistent (or at least didn't seem so to me).

I think you've got the right idea about inst0_fifo_wrreq - it marks a sample arriving from the rf chip.

---

mc955 107 5 October 2018 11:51

@cmichal Surely for your code to be correct `inst0_fifo_wrreq` would need to mark an increment in the sample counter as opposed to a sample arriving from the chip?

For example here:

```
if my_mode ="00" and since_wrreq(0) = '1' then
flag_capture_q <= '1' & smpl_nr_cnt(62 downto 1 ) & '1';
```

When `my_mode="00"` your comments say that we are in "SISO DDR - here there is one clock tick per sample, but the counter increments every second time". Now the above if statement increments `flag_capture_q` by one if there has been once clock cycle since `inst0_fifo_wrreq` was high. However in SISO DDR you said that a sample is put into the FIFO on every clock cycle, which would suggest that `since_wrreq` should always be zero.

You can make similar arguments for each of the other modes, but I think it all looks correct if the signal `inst0_fifo_wrreq` in fact marks an increment in the sample counter.

I might be completely misunderstanding what is going on here, but I'd be interested to hear what you think.

EDIT: That being said I have just made the modifications and it does seem to be working, I get odd time stamps being reported and the results seem believable when timing the number of samples between the 1Hz signal.

```
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720001
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30720000
Samples since last PPS = 30719999
```

Previously I would just see alot of 30720000 and the occasional 30720002 or 30719998.

---

cmichal 108 5 October 2018 17:08

woops - sorry. I last looked at that seriously about 6 months ago. But of course you're right. inst0_fifo_wrreq marks loading samples into the fifo, coinciding with sample counter increments. My since_wrreq counts how many samples have arrived since then.

DOCDAILEY 109 6 November 2019 15:27

Can someone comment on the status of PPS to sync both baseband and LO? I am looking to sync 2 limesdr's for 4 x coherent receive. I have been digging around and cant figure out what the hardware mod is (for sure) and how to use this functionality. I presume I put a sma to a gpio pin.

KarlL 110 6 November 2019 15:54

For the LimeSDR you can get the modified gateware from **that answer above**.

You need to use the GPIO pin 0. When it's low, the timestamps returned by the Lime are normal, but when it's high the timestamp stops changing and it's high bit becomes 1 (if you use SoapySDR, the timestamp will be negative). Once you put it low again, the timestamps continue as if nothing ever happened (so you don't lose track of time).

So to synchronize your 2 devices, you would put both GPIO[0] low, then high at the same time, then back low. At first both devices will have normal timestamps, then at one point the high bit of the timestamp will become 1. The first timestamp for each device with a high bit of 1 correspond to the same time.

RX Timing

RX Timing

DOCDAILEY 111 6 November 2019 17:01

Thanks. So… if I fed a pps signal to GPIO1 it would continually reset the timestamps every second? DOes this sync the pll's or just make sure I can appropriately align the samples in post-processing? I am trying to avoid doing the realignment in post so I can concentrate on adjusting phase after startup and after changing LO. I want a deterministic phase relationship. It can be off…as long as it is always off by the same number of samples.

KarlL 112 7 November 2019 01:11

Changing GPIO[0] will just allow you to sync the devices in post processing. Depending on how long you run your devices and how stable they are, you should be able to determine that a specific timestamp on device A correspond to a specific timestamp on device B. It will be precise up to a certain number of samples, as the FPGA gives one timestamp to the 1st sample of a group of samples (in 12 bit mode, one block is 1360 samples, so 1360 samples in mono channel and 680 samples in dual channels).

DOCDAILEY 113 7 November 2019 04:26

Thanks. I am convinced enough to buy 2 and fiddle. I will be back when I can't figure out how to read the stamp in gnuradio. Earlier if I don't make it that far.

---

mc955 114 13 November 2019 22:44

I don't think you will be able to do the synchronisation using GNU radio, this mod requires you to use the LMS C++ API directly to retrieve the PPS index that has been stashed inside the packet header.

---

mc955 115 13 November 2019 22:54

You should be able to synchronise completely independent SDRs to within a *singe* baseband sample (so T = 1/30.72MS/s = 33ns) not just to the nearest group. It requires further gateware mods from the post you linked earlier (I believe it has been mentioned in this thread though), but it can be done.

In each of the sample streams from the separate SDRs you should be able to identify the sample corresponding to the 0 to 1 transition of GPIO[0], and you can then align them in post processing. The sample rates will differ by a few Hz if your SDRs are independent (not sharing an external clock) but that can be dealt with relatively easily.

Szilandris 116 30 July 2020 08:57

Hi there,

Did you succeed in creating your 4 channel coherent receiver?
I would like to do the same and it would be helpful if you could share some of your experiences

Thanks in advance