Why the SoftPLL is not a HardPLL....

by Tomasz Wlostowski, CERN

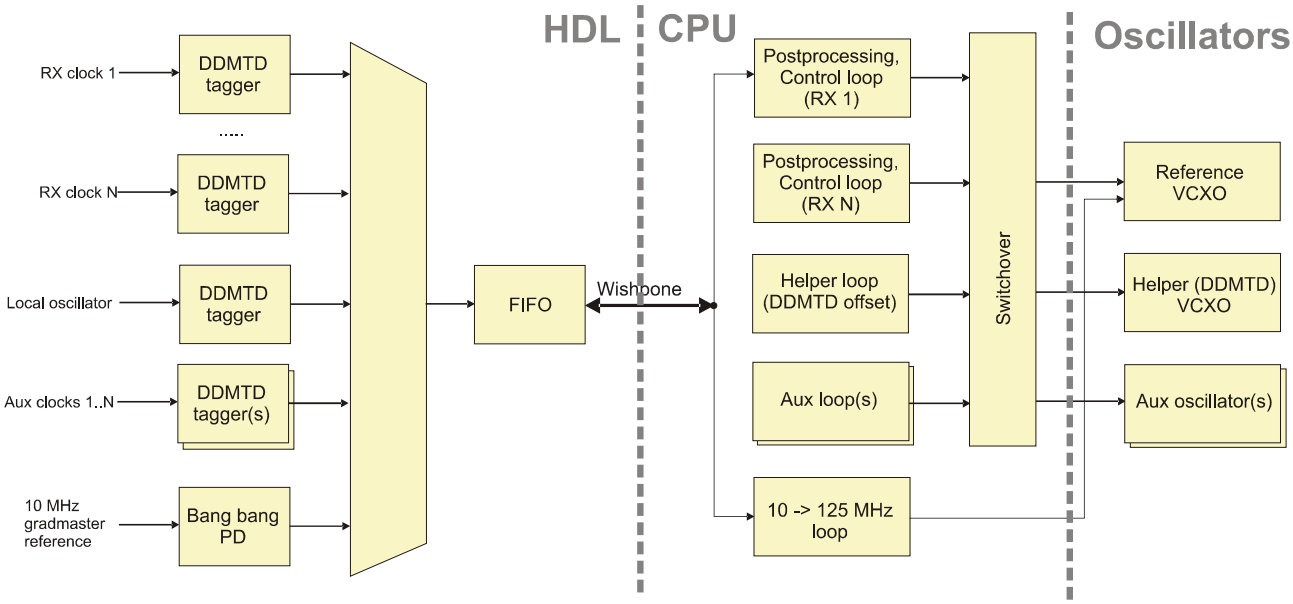
Background

The SoftPLL is the core component of WR synchronization stack. Depending on the mode of a WR node, it performs numerous tasks:

- on WR Slaves it takes the RX clock from the gigabit transceiver and produces a low jitter local reference clock with programmable precise (picosecond range) phase shift with respect to the input clock. This is achieved by using DDMTD phase detectors. SoftPLL can discipline up to 8 local oscillators, each with independent phase offset. An example is the SVEC card with two Fine Delay mezzanines, where each mezzanine's local 125 MHz oscillator is controlled by the SoftPLL.
- on WR Masters it tracks the phase of the reference clock against a number of loop-backed clocks. This information is used alongside PTP timestamps to calculate accurate link roundtrip delay.
- on WR Grandmasters, aside from the tracking returning clocks, it multiplies 10 MHz cesium/GPS reference to 125 MHz Ethernet reference. Because 125 MHz is not divisible by 10 MHz, it also ensures the alignment of edges of these clocks with respect to the PPS signal.

How it works

SoftPLL design consists of two components: a HDL module which does low level frequency/phase detection, resulting in streams of phase/frequency samples (called *tags*) and a software module, running on a realtime embedded CPU which executes the actual control loop algorithm and a number of additional features:



Why in software?

1. **Because control algorithms for PLLs are not so simple**

The PLL control algorithm is not as simple as one would imagine. WR needs at least two control loops – for local reference and DDMTD helper oscillators. These loops input DDMTD tags (counter value at which an edge occured in a DDMTD output signal), unwrap their values, ensure there are no boundary cycles/jumps, calculate phase error, add phase offset, detect lock conditions. As a bonus, add programmable holdover/switchover on top of these loops, on-line capture of PLL response, frequency/phase drift detection and another fancy PLL for 10 MHz input.

Now, imagine the time needed to implement all these modules in pure VHDL with the same level of flexibility (everything tunable on the fly).

2. **Because HDL is difficult to debug.**

My experience with V2 HDL-only PLL and SoftPLL shows that debugging a HDL-only design takes a lot of time, especially for larger designs (WR switch synthesis time is ~2.5 hours). Adding debugging features required developing (and also testing) dedicated HDL modules. Generic solutions (like ChipScope) were found to be too slow and/or unflexible. Simulations do not help much, because it takes much more time to simulate the dynamics of PLL than to synthesize it and test in hardware with try and error method.

On the other hand, in SoftPLL, adding a debug data output means adding one function call and reloading the LM32 firmware. It also costs no extra LUTs.

3. **Because SoftPLL takes much less space**

Let me give an example here:

The WR switch has 18 ports, so it needs 18 phase trackers (WR tracks all RX clocks simultaneously). A single HDL phase tracker (that only does unwrapping and averaging) takes ~200 LUTs on Virtex 6. Single LM32 core takes ~1400 LUTs. 200 * 18 >> 1400, not taking the associated interconnect logic into account. That justifies the use of LM32 in the switch.

In the WR core, LM32 would be present regardless of the implementation of the PLL. Control algorithm running on the CPU takes negligible amount of FPGA resources (in case of the SPEC: 0 LUTs, 0 flipflops and ~9 kB of BRAM).