Oscilloscope Trigger Architecture

Andrew Zonenberg
Antikernel Labs

Introduction

This document describes a high level architecture for digital triggering in FPGA based oscilloscopes and similar test equipment. The intent is to create an open standard which can be implemented by any manufacturer and ease portability of software between instruments.

Hardware Platform

TODO: draw figures here

The block diagram level view of a target instrument consist of :

- One or more channels
- Top level coordination / sync logic

A channel consists of:

- Analog frontend
- A/D converter
- FPGA logic
- Memory
- Interface to top level logic

Top level logic consists of:

- Interface to channel(s)
- FPGA logic
- Interface to front panel GUI, host PC, etc

For the purposes of this document, it is assumed that each channel and the top level logic are physically implemented on separate FPGAs. (Multiple channels, or channels and top level logic, may be freely merged into a smaller number of FPGAs however the subsystems are still conceptually separate.)

The raw data coming off the ADC to each channel can be in the tens to hundreds of Gbps (e.g. 12 bits * 10 Gsps = 120 Gbps), and it is likely that most or all of the channel FPGA's transceivers will be in use for JESD204 interfaces. This requires the interface between the channel and top level logic be implemented in a small number of transceiver channels and/or lower speed LVDS GPIOs. An example of a plausible interface between a channel and the top level logic is 8 LVDS lanes at 1 Gbps per lane (8 Gbps).

As a result, there is a bandwidth barrier between the channel and top level: it is not possible to stream all of the raw ADC data to the top level logic in real time. This presents challenges for triggering as triggering must be done at the top level to some extent (since there are no direct paths between channels).

Trigger Dataflow

TODO: figure here

Waveform data is received from the JESD204 interface and optionally preprocessed by per-channel calibration / correction logic (ADC linearity correction, de-embedding, etc). This gives a stream of N-bit samples at the channel acquisition rate.

The corrected samples are then fed to a circular buffer in (typically external) memory. The trigger subsystem ultimately must provide a signal to the acquisition logic to stop the buffer at the correct time to ensure that the target signal has been captured with the trigger event at the correct timestamp.

The corrected sample data also goes to a block of per-channel trigger logic. This logic is located on the channel FPGA and can process the full rate data but only for this one channel. It can implement simple triggers such as edge and pulse width, but can also do more complex tasks such as thresholding, DSP PLLs for clock recovery, and deserialization of thresholded data after CDR.

The outputs from this logic are then turned into a stream of events. An event may signify anything from “rising edge” to “found K28.5 D16.2 control character sequence in 8b/10b stream”. Events consist of an implementation defined TBD-bit integer ID describing the type of event and a 128-bit timestamp consisting of {64-bit Unix timestamp, 64-bit counter of femtoseconds since UTC second}. *TODO: do we send the entire timestamp over the link? Can probably save a lot of traffic by only sending low bits if we can bound the latency, since all FPGAs are closely located with only a few ns of propagation delay.*

The event stream is then sent over the low bandwidth link to the top level logic. The event stream must thus fit within the capacity of this link. If capacity is exceeded, an overflow event is generated internally. (*TODO: how to handle this - drop some events? Reduce input sample rate going into trigger subsystem?*)

The top level logic then processes the event stream via some sort of state machine and generates a trigger output event, which is sent back to each of the channels. This event includes the timestamp of the trigger, such that the channel logic can halt acquisition at the correct time even if the top level processing took a significant, potentially variable amount of time to determine that the trigger needed to happen.

Per channel trigger block

Turns corrected sample data into a stream of trigger events

This block will consist of a series of fixed function blocks implemented in ordinary FPGA fabric (edge detectors, thresholding, memories, etc) and some sort of runtime programmable logic to connect them together and process their output.

Processing needs to be parallel and able to handle multiple input samples per clock. As an example, a 10 Gsps input might be processed with 200 MHz FPGA logic handling 80 samples per clock.

Top level trigger block

Some kind of programmable state machine that takes events in and outputs trigger events.

Lower speed protocols such as I2C and SPI will be largely implemented here.