Improving the RFNoC Polyphase Channelizer: Adding Downselection and Timestamping

Tim Kurp
EJ Kreinar
Introduction

► Venture-backed startup in Herndon, VA

► Technical Mission:
  • Launch a cluster of small satellites in Fall 2018
  • 3 satellites per cluster, flying in formation
  • Satellites in LEO (low earth orbit) in a polar orbit
  • Satellites share a common ground footprint and provide geometric diversity
  • Passively receive RF signals
  • Independently geolocate emitters from 100 MHz to 15 GHz ("DC to Daylight") using TDOA and FDOA measurements

Requirement to channelize spectrum using FPGA
Introduction: Channelization

► We want to simultaneously process signals at multiple center frequencies

► Using a single RF front end, this can be accomplished in a few ways

► Option 1: Use a large sample rate to capture the entire chunk of spectrum
  • Perform down-conversion of each subchannel in software
  • Disadvantage: Wasted bandwidth in FPGA – Software transfer, downconversion and resampling load is placed on processor
Introduction: Channelization

► We want to simultaneously process signals at multiple center frequencies
► Using a single RF front end, this can be accomplished in a few ways
► Option 2: Perform channelization (filtering and downconversion) of individual channels in the FPGA
  • Place processing load on FPGA, improve bandwidth usage in data transfer
  • Interleave channels in FPGA, deinterleave stream in software (carefully!)
Introduction: Channelization

- There are many possible ways to design a channelizer
  - Brute force mixing and filtering separately for each desired channel
  - Efficient architectures leveraging channel symmetry, FFTs
  - ‘fred harris’ style designs which utilize polyphase filters and FFT for mixing
    - 2X oversampled version used by [1]

- Interesting and efficient polyphase channelizers can be designed, when channels are evenly spaced

- Why channelize in FPGA?
  - Downsample higher bandwidths!

Open Source Channelizer: Current Status?

► Great talk and paper at GRCon 2017
► Missing channel selection, muxing, and timestamping
► Commits may have gotten lost in translation?
  • Would love to see this working!

Back to the basics...
Suppose we want to channelize N streams which are evenly spaced in center frequency.

Each channel k consists of a mixing operation, downsampling and low pass filtering. The noble identities allow us to rewrite into this form:

\[ y_k(m) = \sum_{n=0}^{NP-1} x(n + mN)e^{-j2\pi kn/N}h(n) \]

Each channel k contains the same low pass filter h(n) of length N * P.
Polyphase Channelizer Basics

This equation may be reorganized into the following form:

\[ y_k(m) = h(0)x(t) + \ldots + h(N-1)x(t-N+1)e^{-j2\pi f_s k(N-1)/N} + \ldots + h(N)x(t-N)e^{-j2\pi f_s k(N-1)/N} + \ldots \]

\[ + h(2N-1)x(t-2N+1)e^{-j2\pi f_s k(N-1)/N} + \ldots + h(NP-N)x(t-NP+N)e^{-j2\pi f_s k(N-1)/N} + \ldots + h(NP-1)x(t-NP+1)e^{-j2\pi f_s k(N-1)/N} \]
Polyphase Channelizer Basics

This equation may be reorganized into the following form:

\[
y_k(m) = h(0)x(t) + \cdots + h(N)x(t-N) + \cdots + h(N-1)x(t-N+1) e^{-j2\pi f_sk(N-1)/N} \]

In this form, each column can be seen to be a phase of filter \( h(n) \) with \( P \) taps per phase.

Polyphase Channelizer Basics

This equation may be reorganized into the following form:

\[ y_k(m) = \begin{array}{c}
    h(0)x(t) + \cdots + h(N-1)x(t-N+1)e^{-j2\pi f_s k(N-1)/N} \\
    + h(N)x(t-N) + \cdots + h(2N-1)x(t-2N+1)e^{-j2\pi f_s k(N-1)/N} \\
    \vdots \\
    + h(NP-N)x(t-NP+N) + \cdots + h(NP-1)x(t-NP+1)e^{-j2\pi f_s k(N-1)/N}
\end{array} \]

- In this form, each column can be seen to be a phase of filter h(n) with P taps per phase
- Also, each row can be seen to be a discrete fourier transform

Polyphase Channelizer Basics

- The operations may be implemented by a polyphase filter, where each lag is a phase of h(n), and a FFT
Polyphase Channelizer Basics

- The operations may be implemented by a polyphase filter, where each lag is a phase of h(n), and a FFT
Group Delay

- The polyphase channelizer is a more efficient, but exact implementation of the same basic channelization equation

\[
y_k(m) = \sum_{n=0}^{NP-1} x(n + mN) e^{-j2\pi kn/N} h(n)
\]

- Viewing the equation, it is evident that the response is identical for each channel \( k \)
  - The same filter is used for each channel
  - \( k \) only affects the mixing frequency

- This is wonderful, as each channel can get the same timestamp in a given block!
GNURadio Channelizer Design

- An architecture is required in which the core provides channelization in the FPGA
  - Split output into channels; Create timetags for each channel

- All channels are routed (interleaved) to software
  - RFNoC FPGA-routing of channels will be a future architectural improvement

- The implemented design provides a timestamping framework in which the core itself is replaceable
  - We chose the simple channelizer example presented
  - [1] may also be substituted

GNURadio Channelizer Architecture

FPGA Side
- CHDR Framing
- Channelizer Core
- Channel Downselect

Soft. Side
- Channel Select

RFNoC Block

Software Demux /Timestamp Block

Interleaved N Channels ➔ Interleaved Subset of Channels ➔ Properly Time-Tagged Streams

Core is Interchangeable!
Timestamping Methodology

- Channels are interleaved and downselected out of core into a single stream.

- Stamping of tick count in FPGA must be done in a way that is deterministic when the stream is received in software:
  - Must be able to associate a time tag with a channel number, given that we know which channels have been selected.
  - Solution must be resistant to dropped packets, provide ability to re-tag deinterleaved streams reliably.

- Solution: Manually form CHDR packets, with payload containing a number of samples that is a multiple of the channel select count. Stamp the packet as if the first sample was from channel 0.
Timestamping Methodology Illustrated

- Example: 4 channel channelizer core, channels (0,1,2,3)
- Downselect and output channels 1,2
- Packing a multiple of the channel select count into CHDR ensures that if a packet is dropped, stamps are deterministic
- Since group delay is equal on all channels, use duplicate time tag on all output channels in software deinterleave

Channelizer Core → Downselect → CHDR Packet → Deinterleave

Duplicated Time Tags

CHDR Packet: 3 2 1 0 3 2 1 0

Downselect: 2 1 2 1

Timestamp: Tick count from sample 0.
FPGA Side

- Noc Block:
  - Simple Mode (0)
  - Manual generation of CHDR packets
    - Input timestamps are placed in FIFO
    - Synchronization of output timestamps with data:
      - Newest stamp in FIFO is latched, on CHDR framer
      - XILINX FFT generates TLAST on block boundaries
    - Channel downselection
      - A downselect module pulls valid low on indices to be dropped
      - It also buffers each block, and moves TLAST to the highest channel selected. Example: 8 channel channelizer with select (0,1,3,5) TLAST will occur on 5
      - TLAST causes CHDR packet to be accepted with the latched timestamp
    - On TLAST, the timestamp is incremented by the tick-rate * N, where N is the channel count (pre-downselect), and latched for the next CHDR
  - An EOB resets everything, cannot tolerate timestamps on unknown channel idx

![Channelizer Core Diagram]

- Multi Channel FIR Filter w/ commutation
- XILINX FFT 9.0

- Downselect
  - Timestamp: Tick count from sample 0

- CHDR Packet
Software Side Deinterleave

- Existing GNURadio Deinterleave is used as core
- Around core, must time-tag each stream
  - We know that time tags will be received on the lowest index channel that was selected
  - We also know that group delay is identical on all channels
  - When received, the same time tag is sent out on each stream
- Don't forget about rate and offset!
  - There is a rate conversion of 1/N where N is total channel count
  - Rate parameter in tag must be updated accordingly
  - Also, the offset parameter must be updated for subsequent tags
  - To update offset, the number of output samples on each channel must be tracked internally
Example Flowgraph
Example Flowgraph with Downselection

Total Channels = 8
Output Channels = 4
Example Flowgraph Output

Deinterleaved Output

Input
Lessons Learned: RFNoC/Gnuradio Infrastructure Updates

- Add Gnuradio item tags on output samples
  - Time, Frequency, Sample Rate
- Provide RFNoC timestamp metadata from “file meta source”
  - Allows accurate tests of RFNoC time propagation during playback
- Expose “ticks per sample” to RFNoC CEs

1 tick/sample

8 ticks/sample
Future Work

- Reconfigurable number of channels
- Demux channels inside RFNoC polyphase channelizer gnuradio block
  - Override general_work in rfnoc_block_impl
- Route channels to downstream FPGA blocks
  - Channels need to be de-interleaved and timestamped individually in the FPGA (rather than in software)
  - Nontrivial development. Requires N axi_wrappers?
- Polyphase reconstruction
  - Using multiple adjacent channels, reconstruct a larger channel than the “minimum” channel size
Summary

- Trying to revive the RFNoC channelizer
- Implemented channel downselection in FPGA
- Implemented channel muxing in software
- Confirmed and implemented accurate FPGA and Gnuradio timestamp propagation behavior across channels

FPGA channelizer will be an invaluable utility!

We are actively looking for more interaction and support.
Thank You

Tim Kurp (tim.kurp@he360.com)
EJ Kreinar (ej@he360.com)