RFNoC 4 Workshop

Part 1

Jonathon Pendulum – Ettus Research
Neel Pandeya – Ettus Research
GRCon 2020
Schedule

- Part 1
  - RFNoC 4 Framework Overview
  - Hands on Demos

- Part 2
  - FPGA Architecture
  - Software Implementation
  - GNU Radio Integration
  - Hands on RFNoC Block Development
Host-Based SDR – Current Situation

- PC + Flexible RF Hardware + SDR Framework
Host-Based SDR – Current Situation

- PC + Flexible RF Hardware + SDR Framework
  - GPP: Multi-core + SIMD -- GNU Radio
GNU Radio

Open source toolkit for developing software radios

[Image of GNU Radio interface and code output]

[INFO] [UHD] linux: GNU C++ version 9.2.1 20191008; Boost_106700; UHD_4.0.0.0-0-g4c8ba577
[INFO] [X300] X300 initialization sequence...
[INFO] [X300] Maximum frame size: 8000 bytes.
[INFO] [X300] Radio 1x clock: 200 MHz
[WARNING] [RFNOC::BLOCK_FACTORY] Could not find block with Noc.ID 0xb16, 0xffff
[INFO] [MULTI_USRP] 1) catch time transition at pps edge
[INFO] [MULTI_USRP] 2) set times next pps (synchronously)
gr:log !INFO: audio source - Audio sink arch:alsa
Host-Based SDR – Current Situation

- PC + Flexible RF Hardware + SDR Framework
  - GPP: Multi-core + SIMD -- GNU Radio
Host-Based SDR – Current Situation

- PC + Flexible RF Hardware + SDR Framework
  - GPP: Multi-core + SIMD -- GNU Radio
  - GPU: High performance FP -- OpenCL, gr-fosphor
Host-Based SDR – Current Situation

- **PC + Flexible RF Hardware + SDR Framework**
  - **GPP:** Multi-core + SIMD -- GNU Radio
  - **GPU:** High performance FP -- OpenCL, gr-fosphor
  - **RF HW:** Wide bandwidth, large FPGA -- Rate change DSP
# Universal Software Radio Peripheral

<table>
<thead>
<tr>
<th></th>
<th>Gen 1</th>
<th>Gen 2</th>
<th>Gen 3 (E310)</th>
<th>Gen 3 (X310)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FPGA</strong></td>
<td>Cyclone 1</td>
<td>Spartan 3</td>
<td>Zynq</td>
<td>Kintex 7</td>
</tr>
<tr>
<td><strong>Logic Cells</strong></td>
<td>12K</td>
<td>53K</td>
<td>85K</td>
<td>406K</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>26KB</td>
<td>252KB</td>
<td>560KB</td>
<td>3180KB</td>
</tr>
<tr>
<td><strong>Multipliers</strong></td>
<td>NONE!</td>
<td>126</td>
<td>220</td>
<td>1540</td>
</tr>
<tr>
<td><strong>Clock Rate</strong></td>
<td>64 MHz</td>
<td>100 MHz</td>
<td>200 MHz</td>
<td>250 MHz</td>
</tr>
<tr>
<td><strong>RF Bandwidth</strong></td>
<td>8 MHz</td>
<td>50 MHz</td>
<td>128 MHz</td>
<td>640 MHz</td>
</tr>
<tr>
<td><strong>Free Space</strong></td>
<td>NONE!</td>
<td>~50%</td>
<td>~60%</td>
<td>~75%</td>
</tr>
</tbody>
</table>
Challenges

- Massive processing requirements
  - Welsh’s algorithm for Power Spectrum Estimation
  - 1024 FFT + $|X|^2$ + Moving Average at 200 MSPS
Massive processing requirements
- Welsh’s algorithm for Power Spectrum Estimation
- $1024 \text{ FFT} + |X|^2 + \text{Moving Average at 200 MSPS}$

Challenges

Highly Parallelizable Math
Massive processing requirements
- Welsh’s algorithm for Power Spectrum Estimation
- 1024 FFT + |X|^2 + Moving Average at 200 MSPS

Overloaded transport
- 200e6 samp/sec * 32 bits/samp => 6.4 Gb/sec
Challenges

- Massive processing requirements
  - Welsh’s algorithm for Power Spectrum Estimation
  - 1024 FFT + |X|^2 + Moving Average at 200 MSPS
- Overloaded transport
  - 200e6 samp/sec * 32 bits/samp => 6.4 Gb/sec
- Latency and Determinism
  - Ethernet latency, OS scheduling, precise timing

Transport Overloaded

Highly Parallelizable Math

UHD: USRP Source
Sync: Unknown PPS
Samp rate (Sps): 32k
Ch0: Center Freq (Hz): 0
Ch0: AGC: Default
Ch0: Gain Value: 0
Ch0: Gain Type: Absolute (dB)
Ch0: Antenna: RX2

Stream to Vector

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar... Shift: Yes
Num. Threads: 1

Complex to Mag^2
Vec Length: 1.024k

Moving Average
Length: 1k Scale: 1 Max Iter: 4k Length of Vectors: 1.024k

QT GUI Vector Sink
Vector Size: 1.024k X-Axis Start Value: 0 X-Axis Step Value: 1 X-Axis Units: Y-Axis Units: Ref Level: 0
Opportunity: Use the FPGA!

- Everything USRP is open source, available online (code, firmware, schematics)
- Contains big and expensive FPGA!
- Why do customers not use it?
FPGAs are Hard

FPGA Design Process

- Building for 3 hours, doesn’t work, forgot to connect clock
- Timing constraints not met
  - Worst Negative Slack (WNS): -1.244 ns
  - Total Negative Slack (TNS): -2154.861 ns
- “It worked in simulation…”
- Design doesn’t fit
  - 2 errors
  - 26 critical warnings
  - 1356 warnings

“This is fine.”
Domain vs FPGA Experts

- FPGA development is not a requirement of a communications engineering curriculum
- Math in FPGAs is hard
- Complicated system architecture

almost pure-noise channels. This intuition is clarified more by the following inequality. It is shown in [1] that for any B-DMC $W$,

$$1 - I(W) \leq Z(W) \leq \sqrt{1 - I(W)^2} \quad (2)$$

where $I(W)$ is the symmetric capacity of $W$.

Let $W^N$ denote the channels that results from $N$ independent copies of $W$ i.e. the channel $\langle \{0,1\}^N, \mathcal{Y}^N, W^N \rangle$ given by

$$W^N(y^n_1|x^n_1) \overset{\text{def}}{=} \prod_{i=1}^{N} W(y_i|x_i) \quad (3)$$

where $x^n_1 = (x_1, x_2, \ldots, x_N)$ and $y^n_1 = (y_1, y_2, \ldots, y_N)$. Then the combined channel $\langle \{0,1\}^N, \mathcal{Y}^N, W \rangle$ is defined with transition probabilities given by

$$\tilde{W}(y^n_1|u^n_1) \overset{\text{def}}{=} W^N(y^n_1|u^n_1 G_N) = W^N(y^n_1|u^n_1 R_N G^\otimes n)$$
**RFNoC: RF Network on Chip**

- Make USRP FPGA acceleration more accessible
- Software API + FPGA infrastructure
  - Handles FPGA – Host communication / dataflow
- Provides users simple software and HDL interfaces
  - Infrastructure transparent to user -- reusable code
  - Trade off flexibility versus resource utilization
- Open source
- Fully supported in GNU Radio
  - Modularity and composability
RFNoC Architecture

User Application – GNU Radio

USRP Hardware Driver (UHD)

Ethernet MAC & PHY

NoC Core

RFNoC Radio Block

RFNoC Block

RFNoC Block
Users implement custom FPGA logic in “RFNoC Blocks”
- Architecturally independent of other logic
- Easy to add and remove RFNoC Blocks
RFNoC Architecture

- Users implement custom FPGA logic in “RFNoC Blocks”
  - Architecturally independent of other logic
  - Easy to add and remove RFNoC Blocks
- Ettus provides a library of pre-made RFNoC Blocks
  - RFNoC Radio Block connects to the RF Frontend, SPI, GPIO
RFNoC Architecture

- NoC Core connects RFNoC Blocks to each other and I/O interfaces (e.g. Ethernet, PCIe, etc.)
- Supports both full crossbar connections and static routing
  - Trade off lower resource utilization and latency for flexibility
- Autogenerated based on RFNoC Block topology
RFNoC Architecture

- FPGA connection back to the host
  - External connection: 1 GigE, 10 GigE, PCIe, Aurora
  - Internal connection: AXI4 DMA to Zynq ARM processor (e.g. N310)
  - Parallel interfaces (e.g. X310 has 2 x 10 GigE)
- Connected to NoC Core, but not a RFNoC Block
- Transparent protocol conversion
RFNoC Architecture

User Application – GNU Radio

Software API to:
- Configure USRP hardware & RFNoC FPGA infrastructure
- Provide user sample data (r/w buffers) & control (r/w regs) interfaces
- Many other functions too (timed commands, external ref select, etc.)
RFNoC Architecture

User Application – GNU Radio

- User application
  - Standalone: C, C++, Python
  - Framework: GNU Radio

USRP Hardware Driver (UHD)

RFNoC Radio Block

RFNoC Block

RFNoC Block
RFNoC Architecture

User Application – GNU Radio

Example: Plotting Frequency Spectrum

- RFNoC RX Radio
  - Number of Channels: 1
  - Block Args:
    - Device Select: -1
    - Instance Select: -1
    - Sample Rate (Hz): 0
    - Antenna Select: RLO
    - Center Frequency (Hz): 0
    - Gain: 0
    - Automatic Gain Control: Default
    - Bandwidth (Hz): 0
    - DC Offset Correction: False
    - IQ Balance: False

- RFNoC Rx Streamer
- Stream to Vector
- FFT
  - FFT Size: 1.024k
  - Forward/Reverse: Forward
  - Window: window.blackmanhar... Shift: 0
  - Num. Threads: 1
- Complex to Mag^2
  - Vec Length: 1.024k
  - Log10
    - m: 20
    - k: 0
    - Vec Length: 1.024k
- QT GUI Vector Sink
  - Vector Size: 1.024k
  - X-Axis Start Value: 0
  - X-Axis Step Value: 3
  - Y-Axis Units:
  - Ref Level: 0

- RFNoC Block
- RFNoC Block
- RFNoC Block

NoC Core

HOST PC

USRP FPGA
RFNoC Architecture

User Application – GNU Radio

- RFNoC RX Radio in GNU Radio represents the RFNoC Radio block
  - RFNoC Radio Block has two GNU Radio blocks
  - **RX**: RFNoC RX Radio, **TX**: RFNoC TX Radio
RFNoC Architecture

User Application – GNU Radio

Block Diagram:
- RFNoC RX Radio
- Ethernet MAC & PHY
- NoC Core
- USRP Hardware Driver (UHD)

Connections:
- USBPC
- USRP FPGA

Details:
- Number of Channels: 1
- Block Args: Device Select: -1, Instance Select: -1
- Sample Rate (Hz): 0, Antenna Select: NO
- Center Frequency (Hz): 0, Gain: 0
- Automatic Gain Control: Default, Bandwidth (Hz): 0
- DC Offset Correction: False, IQ Balance: False

FFT:
- Size: 1.024k
- Forward/Revers: Forward
- Window: window.blackmannhar...
- Shift: 0
- Num. Threads: 1

Complex to Mag^2
- Vec Length: 1.024k

Log10
- n: 20
- k: 0
- Vec Length: 1.024k

QT GUI Vector Sink
- Vector Size: 1.024k
- X-Axis Start Value: 0
- X-Axis Step Value: 3
- X-Axis Units: Ref Level: 0
RFNoC Architecture

User Application – GNU Radio

Profiling reveals FFT is a hotspot
RFNoC Architecture

User Application – GNU Radio

- Replace software FFT with FPGA accelerated FFT using RFNoC
RFNoC Architecture

User Application – GNU Radio

- Radio Core -> FFT route could be made static
User Application – GNU Radio

- Profile again and decide if more FPGA acceleration is needed
RFNoC Block Overview

RFNoC Radio
- Depacketizer
- FIFO
- TX Interface
- RX Interface

RFNoC FFT
- Depacketizer
- FIFO
- AXI-Stream
- Xilinx FFT IP
- Packetizer
- FIFO

NoC Core

To Host PC

RX Sample Data
RFNoC Block Overview

- **Block to block communication:**
  - FIFO to FIFO, packetized, flow control (unless static route)
  - Transparent to user – built into RFNoC infrastructure
RFNoC Block Overview

- User interfaces to RFNoC via AXI-Stream
  - Industry standard (ARM), easy to use
  - Large library of existing IP cores
RFNoC Block Overview

- User writes their own custom HDL or drops in IP
  - VHDL, Verilog, SystemVerilog, Vivado HLS
  - Xilinx IP, Vivado Block Diagram
RFNoC Block Overview

- Each block is in their own clock domain
  - Improves throughput
  - Easier timing closure
Cognitive Radio

RFNoC FFT Block (Xilinx IP)

NoC Core

Ethernet MAC & PHY

RX Samples

RFNoC Radio Block

Spectrum Policy (Soft Processor)

TX Modulator (Vivado HLS)
Cognitive Radio

- RFNoC FFT Block (Xilinx IP)
- NoC Core
- Ethernet MAC & PHY
- RX Samples
- RX Spectrum
- RFNoC Radio Block
- Spectrum Policy (Soft Processor)
- TX Modulator (Vivado HLS)
Cognitive Radio

- RFNoC FFT Block
- NoC Core
- Ethernet MAC & PHY
- RX Samples
- RX Spectrum
- Trigger Command
- RFNoC Radio Block
- RFNoC FFT Block (Xilinx IP)
- Spectrum Policy (Soft Processor)
- TX Modulator (Vivado HLS)
Cognitive Radio

- RFNoC FFT Block (Xilinx IP)
- NoC Core
- Ethernet MAC & PHY
- RX Samples
- RX Spectrum
- Spectrum Policy (Soft Processor)
- TX Modulator (Vivado HLS)
- Payload from Host
- Trigger Command

- Ettus Research
Cognitive Radio

- RFNoC FFT Block (Xilinx IP)
- NoC Core
- Ethernet MAC & PHY
- TX Samples
- Payload from Host
- Trigger Command
- RX Samples
- RX Spectrum

- RFNoC Radio Block
- Spectrum Policy (Soft Processor)
- TX Modulator (Vivado HLS)
- Ettus Research
Summary

- Make FPGA acceleration more accessible on USRPs
- Tightly integrated with GNU Radio
- Library of existing RFNoC Blocks
  - FFT, FIR, Signal Generator, Fosphor
- Portable between all third generation USRPs
  - X3x0, E3xx, N3xx
- Completely open source
- kb.ettus.com/RFNoC_Getting_Started_Guides
- Next: FPGA & Software Development
What’s New in RFNoC 4

- Better Documentation

- Software Enhancements
  - Stability and Testing
  - Python support for RFNoC

- FPGA Improvements
  - Scalable to faster sampling rates (250 MSPS+)
  - Instantiate far more RFNoC Blocks
  - Static routing between RFNoC blocks
    - Trade off latency and resource utilization versus flexibility

- GNU Radio 3.8 Support
Fosphor is a real-time GPU-accelerated or FPGA-accelerated spectrum display tool.

Running on a USRP X310 with a WBX daughterboard.

The system is running Ubuntu 20.04 with GNU Radio 3.8.2.0.

All calculations for the FFT and waterfall are being done on the FPGA, not on the CPU.

The CPU is minimally loaded, even for large bandwidths.