FPGAs: Why, When, and How to use them (with RFNoC™) – Pt. 1

Martin Braun, Nicolas Cuervo

FOSDEM 2017, SDR Devroom
Schematic of a typical SDR

- Very rough schematic:

  - Let’s ignore the analog stuff
  - FPGA sits closest to the ADC/DAC
  - GPP is separated by some transport (USB, Ethernet, DMA FIFO, or maybe it’s just on the same PCB)
What is an FPGA?

- Wikipedia: ‘an integrated circuit designed to be configured by a customer or a designer after manufacturing – hence "field-programmable"

- In SDRs: Effectively a user-definable digital circuit between ADC/DAC and the software

- Can be redefined “any time”, but will take down the circuitry while doing so

- Typical clock rates: several hundred MHz (or more? Or less?)

- Remember these:
How are FPGAs programmed?

1. Define your circuitry (shall it filter? Shall it generate UDP packets? Shall it...)

2. Encode that in a format your FPGA toolchain understands (Verilog, VHDL, graphical tools)

3. Synthesize to netlist + generate bitstream. A bitstream is a binary representation of how the internals of the FPGA is configured. Often proprietary formats.

4. Load bitstream onto FPGA, typically using dedicated pins.
Can an FPGA run software? Well, it can, but only if you make it look like a CPU. Let’s ignore that for now.

If you can draw a digital circuit, it’ll usually work well on an FPGA.

Multiple parallel circuits are also possible, and in fact one of the strengths of FPGAs.

Latency can be controlled on the order of clock cycles.

These work well:
  - FIR filters, FFTs, Neural Networks
  - Control loops

These not so much:
  - Protocol handling, complex rulesets

(Source: https://github.com/Themaister/muFFT/blob/master/doxygen/fft.md)
Flexibility (or lack thereof)

- During “runtime”, the digital circuit can’t be easily replaced
- Building bitfiles can take a long time (depending on the tools, design, and chip between a few seconds and several hours)
- If your FPGA is controlling peripherals, those will be disabled while the FPGA is reprogrammed

(Source: Ettus Research USRP E310 Schematic files.ettus.com/schematics/e310)
Challenges: Digital Logic

- Did you pay attention in school?
- Quick, what’s this equation as a digital circuit:

\[(f * g)[n] = \sum_{m=-M}^{M} f[n - m]g[m].\]

- Concepts may seem trivial if you’re an EE major, but there’s a lot of concepts worth knowing (Types of flip flops, bus arbitration, interface designs, memory architectures, ...)
- What does this do?

(Source: https://en.wikipedia.org/wiki/Shift_register)
Challenges: Circuit Magic

- The digital logic is only half of it
- What kind of constraints are relevant for our SIPO?
- Where did the clock come from? How fast is it?
- Will the FFs keep up?
- How long do I need to read the outputs?
- Is ‘Data In’ a pin? Are QN pins? Shouldn’t I connect reset lines?
Challenges: Tools

- Most likely, you’re leaving the safe, easy confines of running gcc and clang
- You’re in for a treat! Good luck getting Vivado running on Gentoo.
- Ever heard of TCL?
Pointers

- EDA Playground: Play around with Verilog in your browser
- Yosys, Icoboard: RPi, free software
- Xilinx, Altera have eval kits e.g. from Digilent
- USRPs will let you do SDR
RF-Network-on-Chip (RFNoC)
RFNoC is for FPGAs is what GNU Radio (currently) is for GPPs.

<table>
<thead>
<tr>
<th>Feature</th>
<th>RFNoC</th>
<th>GNU Radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Provides Easy-to-use Infrastructure for SDR applications</td>
<td>✔ (Growing)</td>
<td>✔ (Huge and well-tested)</td>
</tr>
<tr>
<td>Handles Data Movement between blocks</td>
<td>✔ (AXI-Based)</td>
<td>✔ (Circular Buffers)</td>
</tr>
<tr>
<td>Takes care of boring and recurring tasks</td>
<td>✔ (Flow control, addressing, routing)</td>
<td>✔ (R/W pointer updating, tag handling…)</td>
</tr>
<tr>
<td>Provides library of blocks to get started</td>
<td>✔ (Through gr-ettus)</td>
<td>✔ (Built-in)</td>
</tr>
<tr>
<td>Works with GNU Radio Companion</td>
<td>✔ (Right?)</td>
<td>✔ (Right? RIGHT?)</td>
</tr>
<tr>
<td>Well-documented</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Writes your blocks for you</td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>
Example: Wideband Spectral Analysis

- Simple in Theory: 200 MHz real-time, Welch's Algorithm
- In practice: Several stumbling blocks. That’s the problem RFNoC is trying to solve.

Highly parallelizable operations, basic math => Ideal to shift to FPGA

Transport: Overloaded
RFNoC + GNU Radio: Work nicely together
- Ideal way to use and test RFNoC is with GNU Radio
- Data is passed between "domains" easily

Example: E310 + fosphor
RFNoC Architecture

User Application – GNU Radio

- User Application
- GNU Radio
- Crossbar
- Ingress Egress Interface
- USRP Hardware Driver
- Host PC
- USRP FPGA
- Radio Core
- Computation Engine
- Computation Engine

- RFNoC: Radio
  - Radio Select: A
  - Mode: Rx
  - Stream Args:
    - Center Frequency: 1.982G
    - Sampling Rate: 1M
    - Gain: 20
    - Antenna: TX/RX

- Stream to Vector
  - Num Items: 2
  - Vec Length: 1.024k

- FFT
  - FFT Size: 1.024k
  - Forward/Reverse: Forward
  - Window: window.blackmanharris
  - Num. Threads: 1

- Complex to Mag
  - 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: 1
  - X-Axis Label: x-Axis
  - Y-Axis Label: y-Axis
  - X-Axis Units: y-Axis
  - Y-Axis Units: Ref Level: 0
Blocks are chosen when bitfile is generated
Anatomy of an RFNoC Block

- Blocks are separate entities
  - Separate clock domain
  - Optimized for developing separately