Ultra-wideband SDR architecture for AMD RFSoCs “using” PYNQ based GNU Radio blocks

Marius Šiaučiulis, Louise H. Crockett and Robert W. Stewart

https://github.com/marsiau
https://github.com/strath-sdr/
Contents

• Introduction
  – PYNQ
  – RFSoC
• Hardware Setup Overview
• Hardware Architecture
• Software Architecture
  – JupyterNotebook on RFSoC
  – GNU Radio transmitter
  – GNU Radio receiver
• gr-pynq - sneak peek
• Conclusion
Modern Software Defined Radio (SDR) systems are capable of multi-gigabit-per-second sampling rates, producing/consuming massive amounts of digitized RF data.

Developing systems for SoC devices can be challenging for non-specialists, due to the learning curve of the design tools and processes.

It is often convenient to generate signals using software frameworks before moving to hardware implementation.

We present an open-source RFSoC design capable of Tx/Rx offload in a 100GBit/s network channel with minimum latency. This design uses the:

- AMD PYNQ & RFSoC 4x2 platform – next couple of slides.
- GNU Radio – open-source development toolkit for SDR applications.
Introduction – PYNQ

• AMD PYNQ is an open-source project that exposes Intellectual Property (IP) cores within the FPGA fabric as a collection of Python objects.

• Provides Python APIs for:
  - Loading and managing FPGA bitstreams (called overlays)
  - Interacting with memory mapped interfaces
  - Handling PL interrupts (through asyncio)
  - Direct Memory Access (DMA) IP

• RFSoC-PYNQ Python wrappers for RFSoC specific functionality:
  - xrfclk - to configure external RF reference clocks
  - xrfdc - controlling and interacting with the RF Data Converters (RFDCs)
  - xsdfec - drivers for Soft-Decision Forward Error Correction (SD-FEC) block
Introduction – RFSoC 4x2

- RF Digital to Analog Converters (RF-DACs)
  - 2x
  - 14-bit
  - 9.85Gsp/s
- RF Analog to Digital Converters (RF-ADCs)
  - 4x
  - 14-bit
  - 5Gsp/s
- 100Gbit/s QSFP network interface
- 64-bit Quad core Arm CPU running PYNQ Linux
- Programmable Logic (PL / FPGA)

RFSoC 4x2
http://www.rfsoc-pynq.io/rfsoc_4x2_overview.html
GNU Radio flograph is used to generate OFDM signals at 61.44 MSPS.

I/Q samples are sent/received over a 100Gbit QSFP Network.

GNU Radio with gr-fosphor is utilized for Graphics Processing Unit (GPU) accelerated, real-time spectrum plotting.

An Ethernet connection is used to access a JupyterLab interface running on the board and remotely control the RF-DC parameters.
The high-speed network architecture is implemented entirely on the Programmable Logic (PL), enabling maximum network bandwidth and minimum latency.

User Datagram Protocol (UDP) is used for data transfer over the QSFP28 network.

RF-DAC Ch. B Digital Up-Converter (DUC) interpolates the signal by 80.

RF-ADC Ch. B Digital Down-Converter (DDC) decimates by 40.

RF-ADC Ch. C setup with flexible decimation of 2-16.
Software Architecture - Board

- JupyterLab provides an interactive Python session for dynamic adjustment of software and hardware parameters.

- PYNQ framework provides Python APIs for interacting with the Programmable Logic (PL), controlling parameters of the RF-ADCs, RF-DACs, and programming the internal reference clocks.

- Starts an XML-RPC server for remote commands.

JupyterLab interface

3. Configuring the Overlay

Setup Netlayer IP

The Netlayer IP converts axi stream data into UDP packets and allows talking to other network devices. To learn more about Netlayer IP refer to XUP Vitis Network example README.

First we setup the QSFP IP address for the board.

```python
board_ip = '192.168.4.99'
ol.netlayer.set_ip_address(board_ip, debug=True)
```

Next we set up a socket and populate the socket table with the relevant information.

```python
client_ip = '192.168.4.1'
ol.netlayer.sockets[0] = (client_ip, 60133, 60133, True)
ol.netlayer.populate_socket_table()
```
GNU Radio Tx flowgraph
GNU Radio Rx flowgraph

- XMLRPC blocks enable adjustment of remote board parameters.
- Variable Blocks are used to make the design parametric.
- QT GUI Blocks facilitate the user input options.
## gr-pynq blocks

<table>
<thead>
<tr>
<th>Block Name</th>
<th>Function</th>
<th>Status</th>
<th>Test Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overlay</td>
<td>Loads &amp; manages FPGA bitstreams</td>
<td>Done</td>
<td>Done</td>
</tr>
<tr>
<td>DMA Sink</td>
<td>PS to PL data transfer</td>
<td>Done</td>
<td>Done</td>
</tr>
<tr>
<td>DMA Source</td>
<td>PL to PS data transfer</td>
<td>Done</td>
<td>WIP</td>
</tr>
<tr>
<td>xrfclk</td>
<td>Controls external RF clocks</td>
<td>WIP</td>
<td>WIP</td>
</tr>
<tr>
<td>xrfdc tile</td>
<td>Controls RF-DC tiles</td>
<td>WIP</td>
<td>WIP</td>
</tr>
</tbody>
</table>

Example gr-pynq blocks

**PYNQ Overlay**
- ID: pynq_overlay_0
- Bitstream: None
- dtbo: None
- Download bitstream: True
- Ignore the driver version: False

**DMA Sink**
- DMA block name: none
- DMA buffer size: 1.024k
- Data is complex: True
- Block input data type: Float 32
- Cacheable buffer: True
Summary / Conclusions

• Presented open-source RFSoC design capable of Tx/Rx offload in a 100GBit/s network channel with GNU Radio integration.

• Remote radio control can be achieved using XML-RPC blocks.

• The architecture opens a UDP sockets allowing for arbitrary data transfer, making it applicable not only to RF data and communications but also to various instrumentation applications.

• With the increasing capabilities of SDR technology, offloading to/from power-limited edge devices from/to datacenters will become increasingly important, especially for spectrum monitoring and disaggregated radio applications.
Thanks for listening!
Engage with us:

- https://sdr.eee.strath.ac.uk
- @strathSDR
- github.com/strath-sdr