RFNoC™ Tutorial: FPGA Design

Jonathon Pendulum

8/26/2015
Agenda

- RFNoC FPGA Design Basics
  - Prerequisites
  - Directory structure, Makefiles, Vivado
  - AXI-Stream, Settings Bus
  - Composing designs with existing base blocks
- Writing a Computation Engine
  - Moving Average Example
  - NoC Shell, AXI Wrapper
  - Vivado / external IP integration
  - HLS
- Design Tips
Hardware / Software Prerequisites

- **Hardware:**
  - PC or Laptop, decent amount of RAM (4+ GB)
  - USRP X300, X310, or E310
  - Xilinx JTAG debugger module (optional)
    - Only needed for E310, X3x0 series has built in USB JTAG

- **Software:**
  - Linux based OS
  - Xilinx Vivado 2014.4
    - E310 can use free Webpack version
  - Modelsim (optional)
  - UHD
  - GNU Radio (optional, but highly recommended)
    - gr-ettus
Tutorial Setup

1. uhd & fpga rfnoc-tutorial branch installed
   1. Make sure to pull latest changes if you have not
2. Copy rfnoc_tutorial from USB stick / Dropbox
   1. Dropbox URL: http://tinyurl.com/rfnoc-tutorial
   2. tar -xzf rfnoc_tutorial.tar.gz
3. Copy rfnoc_tutorial/build-X310_RFNOC_HGS & build-ip into fpga/usrp3/top/x300/
4. Run “usrp_x3xx_fpga_burner --addr 192.168.10.2 --fpga-path rfnoc_tutorial/usrp_x310_fpga_rfnoc_tutorial.bit --type HGS”
5. Open Vivado project
   1. cd fpga/usrp3/top/x300
   2. source setupenv.sh
   3. vivado build-X310_RFNOC_HGS/project_1/project_1.xpr
FPGA Directory Structure

- **usrp3** -- Third gen devices (X3x0, E310, B2x0)
  - **tools** -- Build infrastructure software
  - **sim** -- Simulations
  - **lib** -- HDL shared between devices
    - **ip** -- IP cores shared between devices
    - **rfnoc** -- NoC Shell, AXI Wrapper, NoC blocks, base blocks
  - **top** -- Device specific files & toplevel
    - **x300**
      - **ip** -- Device specific IP cores
      - **x300.v**
      - **rfnoc_ce_auto_inst_x310.v**
      - **Makefile**
    - **e300** -- similar structure as above
Building RFNoC FPGA Images

- Make sure Vivado 2014.4 installed
  - Helpful if installed in default directory (in /opt/Xilinux)
- Go to toplevel directory
  - usrp3/top/x300, e300
- source setupenv.sh
  - Sets up Xilinx tools
- make X310_HGS_RFNOC (or E310_RFNOC)
  - To launch Vivado GUI: make GUI=1 X310_HGS_RFNOC
- Build takes about an hour
  - First build even longer due to building IP
Building RFNoC FPGA Images

- Output in build directory
  - usrp_x310_fpga_HGS_RFNOC.bit
  - Includes a report file with utilization / timing info
- Program bitfile with usrp_x3xx_fpga_burner
- make clean
- make cleanall -- also removes build-ip
AXI-Stream

- Part of ARM AMBA standard
- Simple handshake / flow control protocol:
  - Upstream block asserts tvalid
  - Downstream block asserts tready
  - Data is consumed when tvalid & tready == 1
  - tlast used to delimit packets

![Waveform diagram showing clk, tready, tvalid, tlast, and tdata with D/C, D0, D1, D2, and D/C levels.](image)
AXI-Stream

- tdata & tlast have no meaning when tvalid == 0
- Once tvalid is asserted, it cannot be deasserted without at least one tready cycle
- Why is AXI-Stream?
  - Industry standard => Lots of existing IP
  - No need for complicated strobes – data flows through
  - Enhances reusability & composability
Settings Bus

- Common bus in USRP FPGA designs
- Implements control & status registers
  - No flow control
- set_addr, set_data, set_stb, rb_data
  - Why no rb_addr? It is a control register

![Timing Diagram]
Many basic functions already written / generated

- Handwritten code in usrp3/lib/rfnoc
- IP cores in usrp3/lib/ip

Notable blocks:

- Math: mult, mult_add, cmult, mult_rc, multiply, addsub, cadd, divide_int32 (in lib/ip)
- DSP: complex_to_magsqr, complex_to_magphase, cordic_rotate, conj, moving_sum, axi_round, axi_clip, puncture, peak_detector, phase_accum
- Utility: axi_fifo, delay, split_stream, axi_join, axi_serializer, axi_deserializer, axi_packer, axi_unpacker
Composing with Base Blocks
Existing RFNoC blocks

- **Function:**
  - FFT, Windowing, FIR, Vector Averaging, Log Power, Polyphase filter bank, AddSub

- **Utility:**
  - Packet resizer, Split stream, Keep one in N, FIFO
  - Null source and sink

- **Application:**
  - OFDM: Schmidl Cox, One tap Equalizer, Constellation Demapper
  - Fosphor, Radio Core
Existing RFNoC blocks

RFNoC: Radio
- Radio Select: B
- Mode: Rx
- Stream Args: Center Frequency: 2.4G
- Sampling Rate: 20M
- Gain: 10
- Antenna: TX/RX
- Force Vector Length: 64

RFNoC: FIFO
- Device Format: Complex int16
- FIFO Select: 0
- Force Vector Length: 64

RFNoC: OFDM Sync
- FFT Size: 64
- Cyclic Prefix Length: 16
- Threshold: 850m
- delay: 248
- Max Num Symbols: 8

RFNoC: FFT
- FFT Size: 64
- Forward/Reverse: Forward
- Shift: Yes
- FFT Output: Complex
- FFT Scaling Word: 1.365k

RFNoC: OFDM Equalizer
- FFT Size: 64

RFNoC: OFDM Constellation Demap
- FFT Size: 64
- Modulation: QPSK
- Scaling: QPSK

RFNoC: FIFO
- Device Format: Byte
- FIFO Select: 1
- Force Vector Length: 64

File Sink
- File: data.bin
- Vec Length: 64
- Unbuffered: Off
- Append file: Overwrite
1. uhd & fpga rfnoc-tutorial branch installed
2. Successfully programmed USRP X310 with bit file
3. Can open Vivado project
   1. Vivado 2014.4 installed
   2. Vivado evaluation license
   3. Copied build files
4. Can run uhd_usrp_probe
Built-in Computation Engines

PCI Express

GigE x 2

Radio x 2

FIFO x 2

FFT

FIR

Fosphor

LogPwr

Moving Average x 2

Window

Null Sink Source

Keep One in N
Built-in Computation Engines

- PCI Express
- GigE x 2
- Radio x 2
- FIFO x 2
- FFT
- Crossbar
- FIR
- Fosphor
- LogPwr
- Moving Average x 2
- Window
- Null Sink Source
- Keep One in N
- `usrp3/lib/rfnoc/noc_block_moving_average.v`
- \[ y[n] = \frac{1}{M} \sum_{k=0}^{M-1} x[n - k] \]
- Software approach:
  - Array of samples \( x \)
  - \( y[n] = y[n-1] + x[n] - x[n-M] \), \( MA[n] = y[n]*(1/M) \)
- Hardware approach similar:
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

Split

Re

Im

Moving Sum

Xilinx Divider

Join

SR 128: Next Dst

SR 192: Sum Len

SR 193: Divisor

SR 255: Rb Addr

RB 0: Sum Len

RB 1: Divisor

CVITA

AXI-Stream

Settings Bus

Xilinx Divider

Sample Data
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

Split

Moving Sum

Xilinx Divider

Join

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA

AXI-Stream

Settings Bus

NoC ID: 0xAAD2
CVITA over AXI-Stream

- RFNoC’s packet format
- CVITA = Compressed VITA
- Also called CHDR = Compressed Header
## CVITA Packet Protocol

### Packet type based on bits 63, 62, & 60

<table>
<thead>
<tr>
<th>63</th>
<th>62</th>
<th>60</th>
<th>Packet Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Data</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>Data (End of Burst)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Flow Control</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Command</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Response</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Response (Error)</td>
</tr>
</tbody>
</table>

### Table

<table>
<thead>
<tr>
<th>Pkt Type</th>
<th>Has Time</th>
<th>EOB</th>
<th>Seq #</th>
<th>Length (in bytes)</th>
<th>SRC SID</th>
<th>DST SID</th>
</tr>
</thead>
<tbody>
<tr>
<td>63 - 62</td>
<td>61</td>
<td>60</td>
<td>59 - 48</td>
<td>47 - 32</td>
<td>31 - 16</td>
<td>15 - 0</td>
</tr>
</tbody>
</table>

- Fractional Time (Optional, Has Time = 1)
- Payload
- ...
Stream IDs

- 16 bit Stream ID
  - 256 unique crossbar (or device) IDs
  - 16 ports per crossbar
  - 16 ports per block

- Example Crossbar ID: 2, Port: 1, Block Port: 0
  SID: 2.1.0

<table>
<thead>
<tr>
<th>SID</th>
<th>Crossbar Port</th>
<th>Block Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossbar</td>
<td>7 - 4</td>
<td>3 - 0</td>
</tr>
<tr>
<td>15 - 8</td>
<td>7 - 4</td>
<td>3 - 0</td>
</tr>
</tbody>
</table>
Block Ports

- CEs support up to 16 block ports
- Allows CEs to accept multiple streams
  - Time share a function (i.e. FFT)
  - Use data from multiple sources (i.e. addsub block)
- Each block port has dedicated NoC shell output
  - Sample data packets only (str_src_t*, str_sink_t*)
    - Bit width widened to 64*N where N = number of block ports
    - Individual sample data FIFOs for each block port
  - Command packets share interface (cmdout_t*, ackin_t*)
    - Resource utilization can increase quickly
- Shared input bus
  - Does not allow receiving two packets simultaneously
NoC Block Moving Average

noc_block_moving_avg.v

NoC ID: 0xAAD2

Command & Response Packets
Data Packets

Sample Data

Split
Re
Im
Moving Sum
Moving Sum
Moving Sum
Xilinx Divider
Xilinx Divider
Join
SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA
AXI-Stream
Settings Bus
NoC Shell

From Crossbar

Clock Crossing FIFO

Demux

Data
Response
Command
Flow Control

FIFO

Command Processor

Source Flow Control

Flow Control Responder

Flow Control
Response
Command

To User

Settings Bus

From User

Observe

Crossbar Clock Domain

Computation Engine Clock Domain

To Crossbar
NoC Shell

Clock Crossing FIFO

Demux

Data
Response
Command
Flow Control

FIFO

Command Processor

Source Flow Control

Flow Control Responder

Flow Control

From User

To User

Settings Bus

From Crossbar

Clock Crossing FIFO

To Crossbar

Crossbar Clock Domain

Computation Engine Clock Domain

Observe
Flow Control

- Fine and course grained flow control
  - Fine grained: AXI-Stream
  - Course grained: Credit based flow control
    - Every endpoint (or consumer) has a receive window
    - Every source (or producer) knows window size
    - Producers send packets until consumer window is full
    - Consumer notifies (acks) producer as window empties
      - i.e. gives credits back to producer
    - Some similarities to TCP flow control
A few additional design rules

1. One consumer per producer
   - Otherwise very difficult to keep track of window
   - Use block ports to allow a CE to receive multiple streams

2. Producers must buffer entire packet before releasing
   - Prevents deadlock in crossbar
   - Prevents slow packets from causing congestion

3. All routing based on first line of CHDR header
   - Higher performance

4. Samples / data always dropped at producer
   - Deterministic -- ensures no lost packets in the middle
   - Single point to restart stream
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

Split

Re Im

Moving Sum

Xilinx Divider

Join

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA

AXI-Stream

Settings Bus

NoC ID: 0xAAD2

Xilinx Divider

Split

Moving Sum

Moving Sum

Re Im
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

Split

Moving Sum

Xilinx Divider

Join

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA

AXI-Stream

Settings Bus

NoC ID: 0xAAD2

Xilinx Divider

Xilinx Divider
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA

AXI-Stream

Settings Bus

Xilinx Divider

Moving Sum

Split

Re

Im

Xilinx Divider

Join

RB 0: Sum Len
RB 1: Divisor

NoC ID: 0xAAD2
AXI Wrapper

From NoC Shell

Addr == 129 or 130?

Write Enable

Control FIFO

Deframer

.Resize Input

Header FIFO

1 0

Framer FIFO

From User IP

To User IP

Next Destination

User Provided CHDR Header

Sample Data

From User IP

To NoC Shell

CHDR

AXI-Stream

Settings Bus
AXI Wrapper

From NoC Shell

Addr == 129 or 130?

Write Enable

Control FIFO

Deframer

Resize Input

Header FIFO

Next Destination

User Provided CHDR Header

Simple Mode

1 0

Framer

Resize Output

To User IP

From User IP

To NoC Shell

CHDR

AXI-Stream

Settings Bus
AXI Wrapper

From NoC Shell → Control FIFO → Deframer → Header FIFO → Framer → FIFO

Addr == 129 or 130?
Write Enable

Control Data → Resize Input → Sample Data → To User IP

User Provided CHDR Header

Next Destination

Simple Mode

1 0

Sample Data → Resize Output → From User IP

CHDR
AXI-Stream
Settings Bus
AXI Wrapper

From NoC Shell

Addr == 129 or 130?

Control FIFO

Write Enable

Control Data

Sample Data

Next Destination

User Provided CHDR Header

From User IP

Simple Mode

1 0

Header FIFO

Resize Input

Sample Data

Resize Output

To User IP

From NoC Shell

CHDR

AXI-Stream

Settings Bus
AXI Wrapper

From NoC Shell

Addr == 129 or 130?

Write Enable

Control FIFO

Deframer

Resize Input

Header FIFO

Next Destination

User Provided CHDR Header

Simple Mode

1 0

Resize Output

From User IP

To NoC Shell

CHDR

AXI-Stream

Settings Bus
NoC Block Moving Average

noc_block_moving_avg.v

NoC ID: 0xAAD2

Command & Response Packets

Data Packets

Sample Data

Split

Re

Im

Moving Sum

Moving Sum

Xilinx Divider

Xilinx Divider

Join

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA
AXI-Stream
Settings Bus
Register Space

- Settings Bus 8-bit address = 256 regs
- NoC Shell Regs: 0 – 127
  - 0 – 63: Flow control (4 per block port, x16)
  - 127: NoC Shell readback address
- NoC Shell Readback Regs:
  - 0: NoC ID
  - 1: Window Size
- User Regs: 128 – 255
  - 128: Next destination (AXI Wrapper, set by software)
  - 129: Control data (tlast not asserted)
  - 130: Control data (tlast asserted)
  - 255: User readback address
# Settings Bus

- Control packet payload sets addr & data
  - \([63:0] = \{24'd0, set_addr[7:0], set_data[31:0]\}\)

- Response packet payload has readback data
  - Write readback address 255 \(\Rightarrow\) Response packet has readback register data

<table>
<thead>
<tr>
<th>clk</th>
<th>set_stb</th>
<th>set_addr</th>
<th>set_data</th>
<th>rb_data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>D/C</td>
<td>D/C</td>
<td>D/C</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A0</td>
<td>D/C</td>
<td>R0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>D/C</td>
<td>D/C</td>
<td>R1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A1</td>
<td>D/C</td>
<td></td>
</tr>
</tbody>
</table>
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

Split

Moving Sum

Xilinx Divider

Join

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA

AXI-Stream

Settings Bus
NoC Block Moving Average

- noc_block_moving_avg.v
- NoC ID: 0xAAD2

NoC Shell

- Command & Response Packets
- Data Packets

AXI Wrapper

- Sample Data

Split
- Re
- Im

Moving Sum
- Xilinx Divider

Moving Sum
- Xilinx Divider

Join

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

CVITA
- AXI-Stream
- Settings Bus
NoC Block Moving Average

noc_block_moving_avg.v

NoC Shell

Command & Response Packets

Data Packets

AXI Wrapper

Sample Data

SR 128: Next Dst
SR 192: Sum Len
SR 193: Divisor
SR 255: Rb Addr
RB 0: Sum Len
RB 1: Divisor

Split

Re

Im

Moving Sum

Xilinx Divider

Join

Moving Sum

Xilinx Divider

CVITA

AXI-Stream

Settings Bus
Macro based test bench infrastructure for RFNoC

See usrp3/sim/rfnoc/sim_rfnoc_lib.vh
RFNoC Test bench Infrastructure

- Sets up a realistic, simulated RFNoC infrastructure
- Requires ModelSim (Vivado support soon!)
Moving Average Test bench

- noc_block_moving_avg_tb.sv
In system debugging

- Vivado ILA: Integrated Logic Analyzer
  - Real time logic analyzer inserted into design
  - Integrated into Vivado GUI
  - Uses logic resources (Slices, BRAM)

- Multiple ways to instantiate, see Xilinx docs

- Easiest to mark nets & registers for debug in HDL
  - Verilog:
    (* dont_touch = “true”, mark_debug = “true” *) wire a;
    (* dont_touch = “true”, mark_debug = “true” *) reg b;
  - VHDL:
    ATTRIBUTE MARK_DEBUG : string;
    ATTRIBUTE MARK_DEBUG of a : SIGNAL IS "TRUE";
    ...

How do we add CEs to the build system?

1. Place new CE in usrp3/lib/rfnoc
2. Add block(s) to Makefile.inc
3. Edit rfnoc_auto_ce_inst_x300.v
   - Or rfnoc_auto_ce_inst_x310.v, rfnoc_auto_ce_inst_e310.v
4. Run make X310_HGS_RFNOC
   1. Or X300_HGS_RFNOC, E310_RFNOC
High Level Synthesis

C code => HDL!
- Constraints on implementation
- Requires pragmas / tcl constraints
- Need to refer closely to docs
- Separate license

AddSub example
- lib/hls/addsub.c
- Set noc_block_addsub parameter USE_HLS = 1
- Replaces addsub.v with HLS output
- make X310_RFNOC_HLS_HGS
Avoid bubble cycles to improve throughput

When splitting an AXI stream, use FIFOs (i.e. split_stream_fifo.v)
  - Prevents back up in one stream from blocking another
  - May need to adjust FIFO depth

When combining streams, make sure to account for different path latency
  - Inserting a FIFO on the shorter path can prevent bubble cycles (waiting for data to “catch up”)
  - Manual intervention, best done in simulation
RFNoC FPGA Design Tips

- Be mindful of dead lock situations
  - Mostly avoided by carefully following “do not deassert tvalid without tready” rule
- Watch out for tready, tvalid combinatorial paths
  - Subtle source of timing issues (when chaining multiple blocks)
  - Consider inserting register: axi_fifo_flop.v
- Ignore output without a tvalid (when debugging)
- Crossbar statistics tool:
  firmware/usrp3/x300/x300_debug.py
RFNoC FPGA Design Tips

- Check if your block / functionality already exists
  - Growing list of base blocks
  - AXI Wrapper RESIZE_INPUT / OUTPUT options

- Subtle Verilog bugs
  - Case sensitive, automatic wire instantiation
    - rst vs reset
    - clk vs ce_clk
  - Width truncation / extension
  - Bit packing / concatenation oddities
    - Tend to see this in test benches with default sized local parameters
Simulate!
Future Improvements

- Radio Lite
  - DSP separated from Radio Core, very useful for E310
- Improvements to NoC Shell
- Additional base blocks
- DRAM RFNoC block
- Resource utilization reduction
- No promise dates 😊
- Any suggestions?
Guided Debugging

- GNU Radio graphical debugging
- Vivado ILA + GNU Radio