Meet Your USRP Makers

Subhead

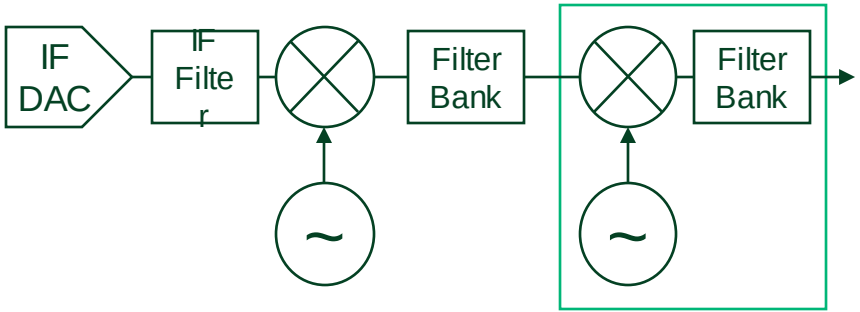
Product R&D Team USRP

Jan Schirok

Product Architect

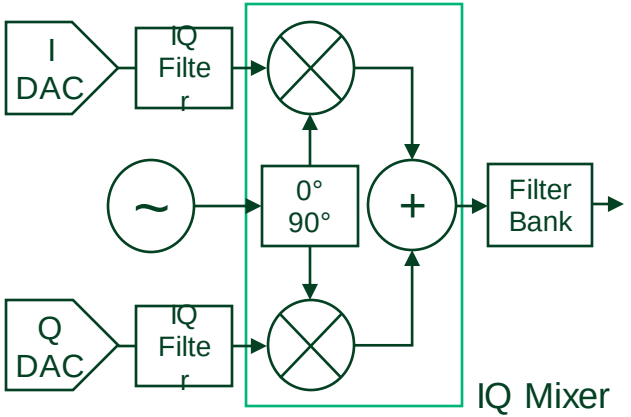
Superhet vs. Zero-IF RF

Superhet (IF): Used in X410



Only for Fc < 3 GHz

Zero-IF (IQ): Used in many other USRPs

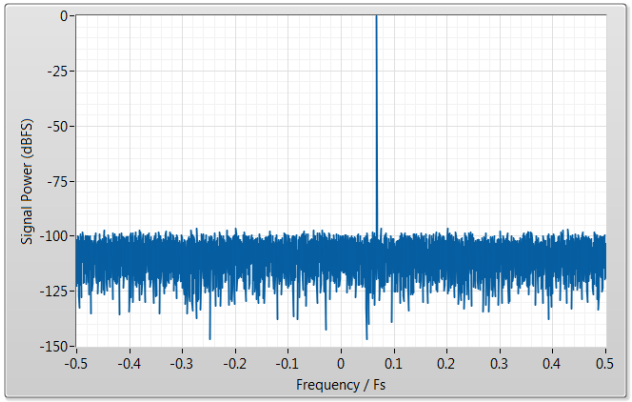


IQ Mixer

	IF	IQ
Advantage	Cleaner in-band spectrum	Cleaner out-of-band spectrum
Mixers + Synthesizers	X410: 1 or 2 (only 1 for < 3 GHz)	UBX: 1 or 2 (2 for < 500 MHz)
ADCs/DACs	1 (with high sample rate)	2 (with moderate sample rate)
Complexities	Frequency planning RF filters	In-band spurs (IQ mismatch/DC offset) IQ mixer selection

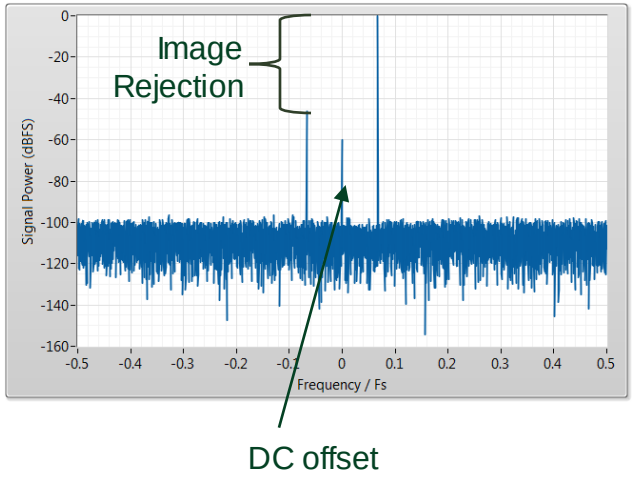
Ideal In-band Spectrum

IF based



Spurs are usually from out-of band mixing products

IQ based

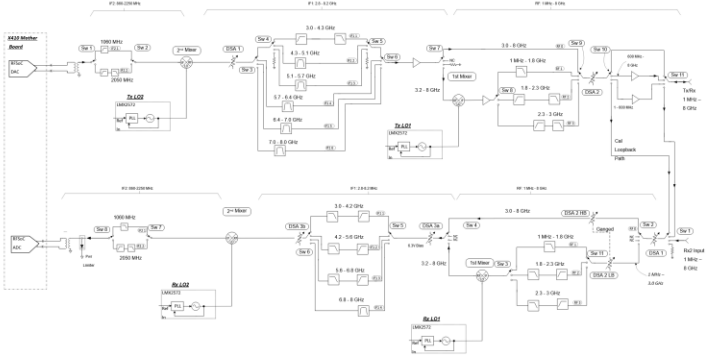


In-band spurs based on IQ scheme

X410 IF Frequency Plan

```
214 static const std::vector<tune_map_item_t> rx_tune_map = {
215 //  | min_band_freq | max_band_freq | rf_fir | if1_fir | if2_fir | mix1 m, n | mix2 m, n | if1_freq_min | if1_freq_max | if2_freq_min | if2_freq_max |
216     {      1e6,          200e6,          1,       1,        2,      -1,  1,      -1,  1,       4100e6,        4100e6,        1850e6,        1850e6    },
217     {    200e6,          400e6,          1,       1,        2,      -1,  1,      -1,  1,       4100e6,        4100e6,        1850e6,        1850e6    },
218     {    400e6,          500e6,          1,       1,        2,      -1,  1,      -1,  1,       4100e6,        4100e6,        1850e6,        1850e6    },
219     {    500e6,          900e6,          1,       1,        2,      -1,  1,      -1,  1,       4100e6,        4100e6,        1850e6,        1850e6    },
220     {    900e6,         1800e6,          1,       1,        2,      -1,  1,      -1,  1,       4100e6,        4100e6,        2150e6,        2150e6    },
221     {   1800e6,         2300e6,          2,       1,        1,      -1,  1,      -1,  1,       4100e6,        4100e6,        1060e6,        1060e6    },
222     {   2300e6,         2700e6,          3,       1,        1,      -1,  1,      -1,  1,       4100e6,        3700e6,        1060e6,        1060e6    },
223     {   2700e6,         3000e6,          3,       4,        2,       1, -1,       1, -1,       7000e6,        7100e6,        2050e6,        2080e6    },
224     {   3000e6,         4200e6,          0,       1,        2,       0,  0,      -1,  1,             0,             0,        1850e6,        1850e6    },
225     {   4200e6,         4500e6,          0,       2,        2,       0,  0,      -1,  1,             0,             0,        1850e6,        1850e6    },
226     {   4500e6,         4700e6,          0,       2,        1,       0,  0,      -1,  1,             0,             0,        1060e6,        1060e6    },
227     {   4700e6,         5300e6,          0,       2,        1,       0,  0,      -1,  1,             0,             0,        1060e6,        1060e6    },
228     {   5300e6,         5600e6,          0,       2,        1,       0,  0,       1, -1,             0,             0,        1060e6,        1060e6    },
229     {   5600e6,         6800e6,          0,       3,        1,       0,  0,       1, -1,             0,             0,        1060e6,        1060e6    },
230     {   6800e6,         7400e6,          0,       4,        1,       0,  0,       1, -1,             0,             0,        1060e6,        1060e6    },
231     {   7400e6,         8000e6,          0,       4,        2,       0,  0,       1, -1,             0,             0,        1850e6,        1850e6    },
232 };
```

https://github.com/EttusResearch/uhd/blob/master/host/lib/include/uhdlib/usrp/dboard/zbx/zbx_constants.hpp



https://files.ettus.com/manual/page_zbx.html

X410 uses IF-based architecture

Challenge is to cover wide frequency range

ZBX is 1M – 7.2 GHz (tunes up to 8.0 GHz)

→ Frequency planning

→ Choices of:
IF frequencies
2 LO frequencies
Filter in each filter bank

Per channel, ZBX contains

4 Synthesizers, ~20 Amplifiers, > 40 Filters

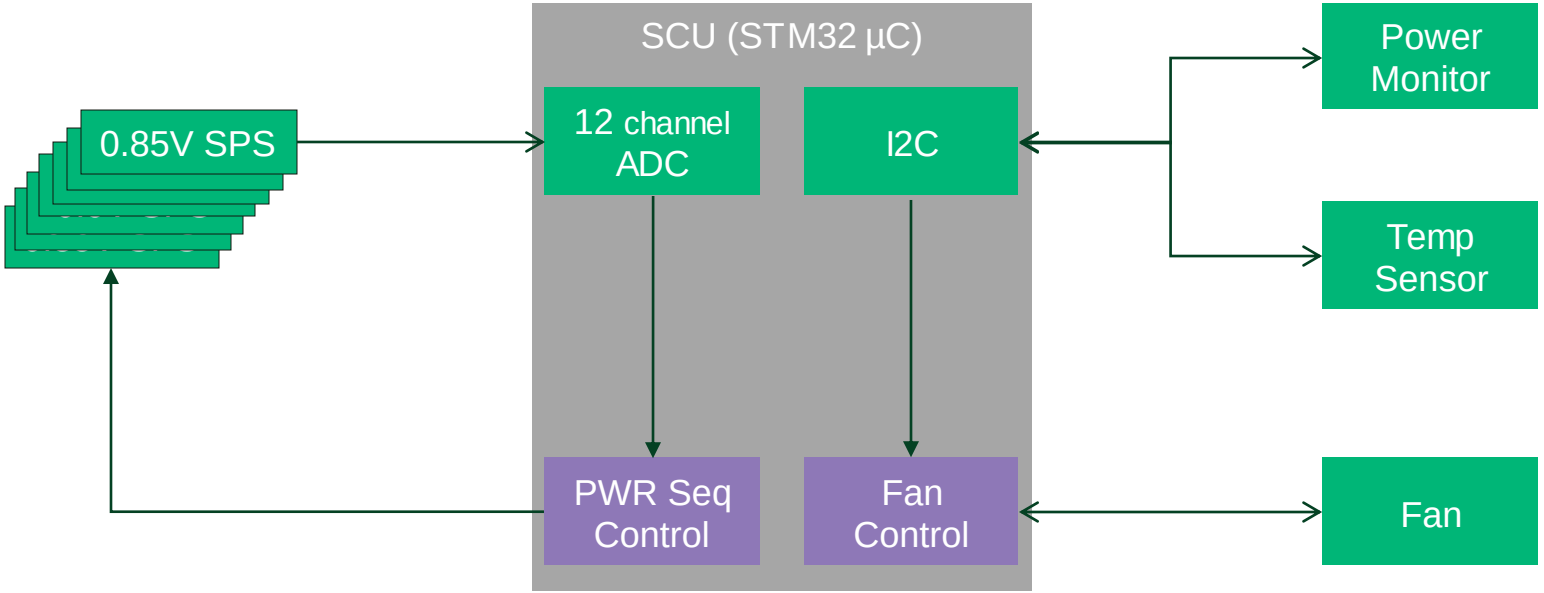
X410 has 4 of those channels

→ A lot of RF to put on a board

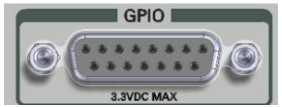
Volkan Öz

Analog Design Engineer

Device Monitoring



GPIO Interface

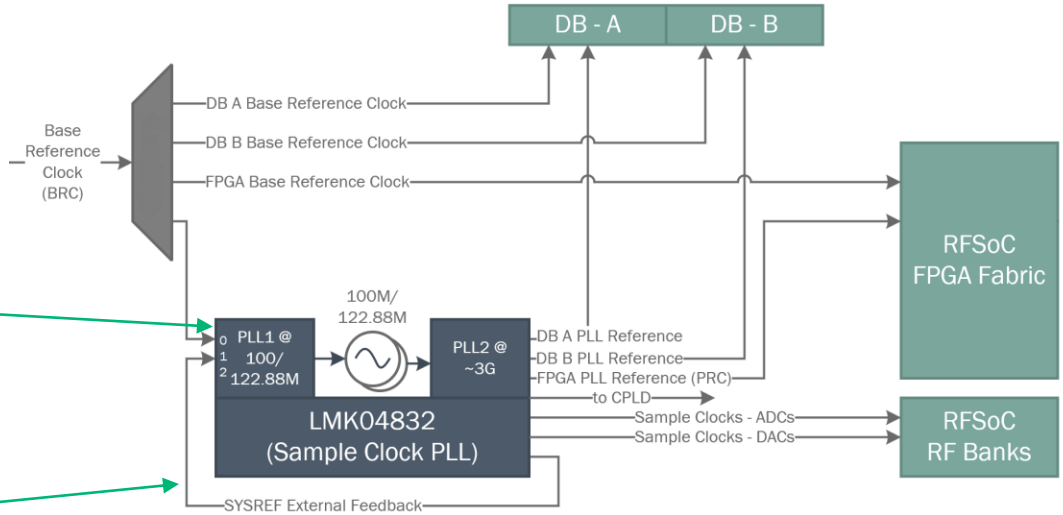


- 1x 12 I/O lines
- Maximum data rate <10Mbps
- I/O voltage 3.3V

- 2x 12 I/O lines per connector
- Maximum data rate 100Mbps
- Selectable I/O voltage (3.3V, 2.5V, or 1.8V)

Clocking

PLL1 Divider Reset
Improved phase coherence
(Single- and Multi-Device)



Nested 0-Delay Mode
Fixed deterministic phase relationship input and output clocks

Lars Amsel

Software Engineer

.NET API for USRPs

- Motivation
  - UHD written in C++, with APIs for Python and C
  - Only the C-API is usable from .NET/C# using pInvoke.
  - C-API is only a subset of the multi_usrp API and has no support for RFNoC
  - .NET-API that wraps multi_usrp and RFNoC-API (like the Python API does) would ease USRP usage from C# or LabVIEW
- Proof of Concept
  - Discarded automation tools like SWIG or cppSharp
    - Difficult to use with UHD because UHD makes heavy use of concepts from stdlib which are not allowed in .NET
    - No additional dependency for UHD
  - Use C++/CLR, a MS C++ dialect which is predestinated for wrapping C++ into .NET

.NET API for USRPs

- C++/CLR allows for native code but parameter and return values are restricted to .NET calls
  - UHD would need minimal adaptation (so it does not use native primitives in its header files)
  - Wrapping UHD objects like smart pointers
  - Most multi_usrp calls can be simply forwarded in the wrapper class
  - UHD uses exceptions for error signaling, which does not leave the .NET boundary
  - More work is needed for non primitive types, but arrays can be casted (no need for copy)

```
namespace uhd {
    namespace usrp {
        public ref class MultiUsrp
        {
        public:
            MultiUsrp(String^ addr) {
                addr = "addr=" + addr;
                _usrp = multi_usrp::make(uhd::device_addr_t(string_to_char_array(addr)));
            }
            ~MultiUsrp() {

            }
            void set_rx_antenna(String^ antenna) {
                set_rx_antenna(antenna, 0);
            }
            void set_rx_antenna(String^ antenna, size_t chan) {
                _usrp->set_rx_antenna(string_to_char_array(antenna), chan);
            }

            String^ get_rx_antenna() {
                return get_rx_antenna(0);
            }
            String^ get_rx_antenna(size_t chan) {
                return gcnew String(this->_usrp->get_rx_antenna(chan).c_str());
            }
```

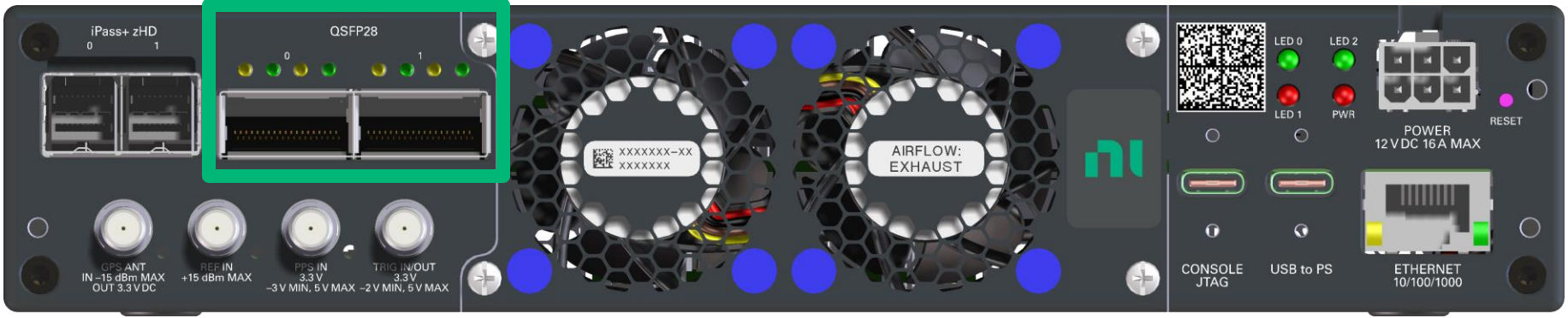
```
            RXStreamer^ get_rx_stream() {
                uhd::stream_args_t args("fc32", "sc16");
                std::vector<size_t> channels;
                channels.push_back(1);

                args.channels = channels;
                try
                {
                    return gcnew RXStreamer(_usrp->get_rx_stream(args));
                }
                catch (const std::exception& e)
                {
                    std::cout << e.what() << std::endl;
                }

            }
        private:
            m_shared_ptr<multi_usrp> _usrp;

            static const char* string_to_char_array(String^ string)
            {
                const char* str = (const char*)(Marshal::StringToHGlobalAnsi(string)).ToPointer();
                return str;
            }
```

Humberto Jimenez

Digital Engineer

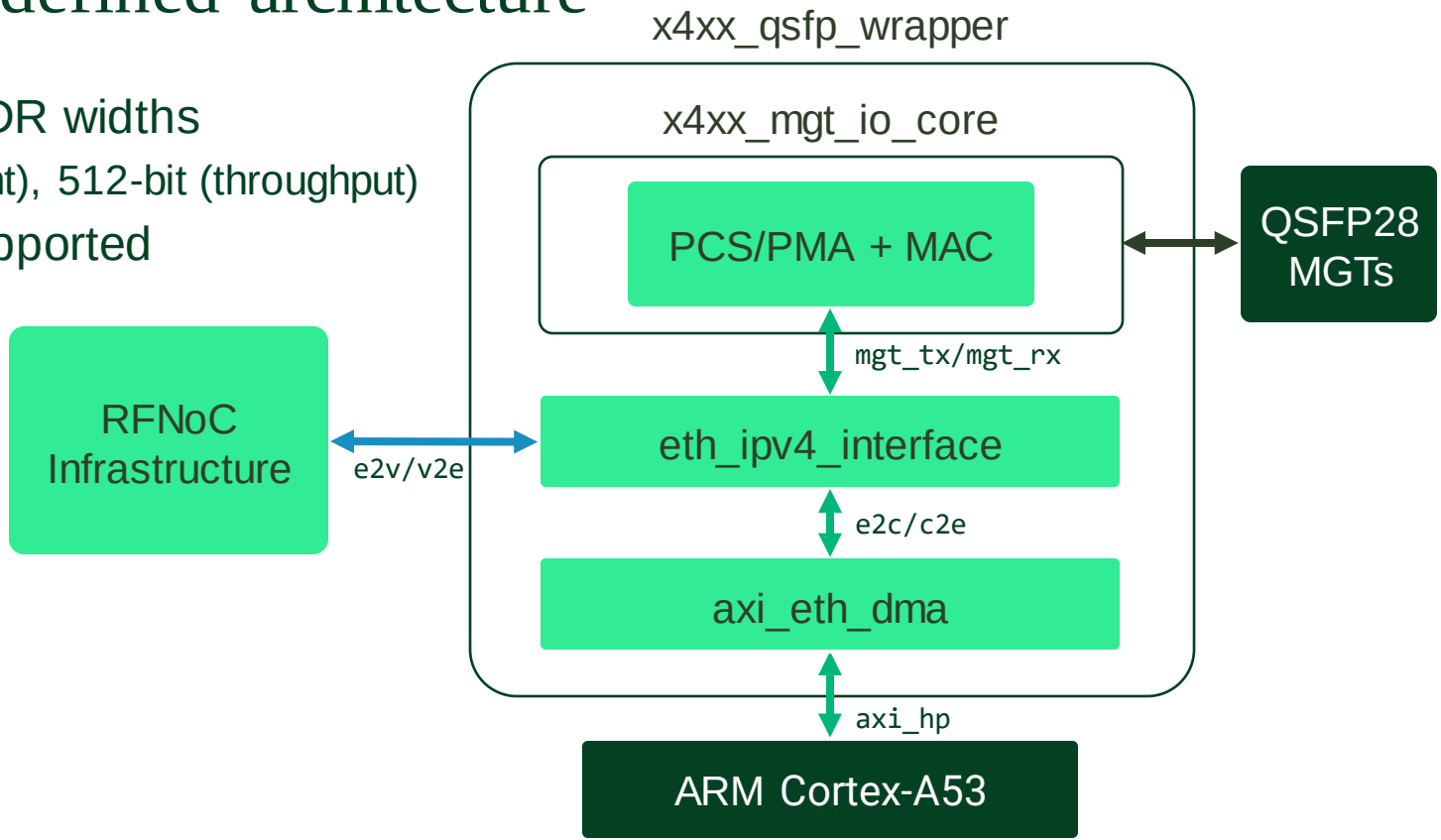
Hardware Capabilities

- Two QSFP28 connectors
- 4 MGTs per connector
- Each MGT lane rated up to 25 Gb/s
- Total aggregated max BW = 200 Gb/s



FPGA: Well-defined architecture

- Adjustable CHDR widths
  - 64-bit (footprint), 512-bit (throughput)
- RX pausing supported



```
module x4xx_qsfp_wrapper_all_tb;

  x4xx_qsfp_wrapper_tb #(
    .TEST_NAME ("100GbE_F"),
    .PROTOCOL0 (`MGT_100GbE),
    .CHDR_W    (512),
    .USE_MAC   (0)
  ) ETH_100Gb_fast ();

  x4xx_qsfp_wrapper_tb #(
    .TEST_NAME ("10GbE_F"),
    .PROTOCOL0 (`MGT_10GbE),
    .CHDR_W    (64),
    .USE_MAC   (0)
  ) ETH_10Gb_fast ();

  x4xx_qsfp_wrapper_tb #(
    .TEST_NAME ("10GbE_x4_F"),
    .PROTOCOL0 (`MGT_10GbE),
    .PROTOCOL1 (`MGT_10GbE),
    .PROTOCOL2 (`MGT_10GbE),
    .PROTOCOL3 (`MGT_10GbE),
    .CHDR_W    (64),
    .USE_MAC   (0)
  ) ETH_10Gb_x4_fast ();

  x4xx_qsfp_wrapper_tb #(
    .TEST_NAME ("100GbE_512S"),
    .PROTOCOL0 (`MGT_100GbE),
    .CHDR_W    (512),
    .USE_MAC   (1)
  ) ETH_100Gb_512serial ();

  x4xx_qsfp_wrapper_tb #(
    .TEST_NAME ("100GbE_128S"),
    .PROTOCOL0 (`MGT_100GbE),
    .CHDR_W    (128),
    .USE_MAC   (1)
  ) ETH_100Gb_128serial ();

  x4xx_qsfp_wrapper_tb #(
    .TEST_NAME ("10GbE_S"),
    .PROTOCOL0 (`MGT_10GbE),
    .CHDR_W    (64),
    .USE_MAC   (1)
  ) ETH_10Gb_serial ();

  bit clk,rst;

  sim_clock_gen #(100.0) clk_gen (clk, rst);

  // Wait for all done
  always_ff@(posedge clk) begin
    if (ETH_100Gb_fast.test.done      &&
        ETH_10Gb_fast.test.done       &&
        ETH_10Gb_x4_fast.test.done    &&
        ETH_100Gb_512serial.test.done &&
        ETH_100Gb_128serial.test.done &&
        ETH_10Gb_serial.test.done
    ) $finish(1);
  end

endmodule
```

FPGA: SystemVerilog

Getting to 100 GbE

- Testbenches
- AXI Interfaces

```
module x4xx_qsfp_wrapper #(
  // Must be a value defined in x4xx_mgt_types.vh
  parameter integer PROTOCOL [3:0] = {`MGT_Disabled,
                                      `MGT_Disabled,
                                      `MGT_Disabled,
                                      `MGT_Disabled},
  parameter         CPU_W          = 64,
  parameter         CHDR_W         = 64,
  parameter         BYTE_MTU       = $clog2(8*1024),
  parameter  [ 7:0] PORTNUM        = 8'd0,
  parameter  [15:0] RFNOC_PROTOVER = {8'd1, 8'd0}
)(
  // Resets
  input logic areset,
  input logic bus_rst,
  input logic clk40_rst,

  // Clocks
  input logic refclk_p,
  input logic refclk_n,
  input logic clk100,
  input logic bus_clk,

  // AXI-Lite register access
  AxiLiteIf.slave s_axi,

  // Ethernet DMA AXI to PS memory
  AxiIf.master axi_hp,

  // MGT high-speed IO
  output logic [3:0] tx_p,
  output logic [3:0] tx_n,
  input  logic [3:0] rx_p,
  input  logic [3:0] rx_n,

  // CHDR router interface
  AxiStreamIf.master e2v [4],
  AxiStreamIf.slave  v2e [4],

  // ETH DMA IRQs
  output logic [3:0] eth_rx_irq,
  output logic [3:0] eth_tx_irq,

  // Misc.
  output logic        rx_rec_clk_out,
  input  logic [15:0] device_id,

  output logic [3:0][31:0] port_info,

  output logic [3:0] link_up,
  output logic [3:0] activity
);
```

```
module eth_ipv4_interface #(
  logic [15:0] PROTOVER         = {8'd1, 8'd0},
  int          CPU_FIFO_SIZE    = $clog2(8*1024),
  int          CHDR_FIFO_SIZE   = $clog2(8*1024),
  int          NODE_INST        = 0,
  int          RT_TBL_SIZE      = 6,
  int          REG_AWIDTH       = 14,
  int          BASE             = 0,
  bit          DROP_UNKNOWN_MAC = 0,
  bit          DROP_MIN_PACKET  = 0,
  int          PREAMBLE_BYTES   = 6,
  bit          ADD_SOF          = 1,
  bit          SYNC             = 0,
  bit          PAUSE_EN         = 0,
  int          ENET_W           = 64,
  int          CPU_W            = 64,
  int          CHDR_W           = 64
) (
  input logic        bus_clk,
  input logic        bus_rst,
  input logic [15:0] device_id,

  // Register port: Write port (domain: bus_clk)
  input logic                  reg_wr_req,
  input logic [REG_AWIDTH-1:0] reg_wr_addr,
  input logic [31:0]           reg_wr_data,

  // Register port: Read port (domain: bus_clk)
  input  logic                  reg_rd_req,
  input  logic [REG_AWIDTH-1:0] reg_rd_addr,
  output logic                  reg_rd_resp,
  output logic     [31:0]       reg_rd_data,

  // Status ports (domain: bus_clk)
  output logic [47:0] my_mac,
  output logic [31:0] my_ip,
  output logic [15:0] my_udp_chdr_port,

  // Ethernet MAC (domain: eth_rx.clk)
  output logic       eth_pause_req,
  AxiStreamIf.master eth_tx, // tUser = {1'b0,trailing bytes};
  AxiStreamIf.slave  eth_rx, // tUser = {error,trailing bytes};

  // CHDR router interface (domain: eth_rx.clk)
  AxiStreamIf.master e2v, // tUser = {*not used*};
  AxiStreamIf.slave  v2e, // tUser = {*not used*};

  // CPU DMA
  // (domain: e2c.clk if SYNC=0, else eth_rx.clk)
  AxiStreamIf.master e2c, // tUser = {sof,trailing bytes};
  // (domain: c2e.clk if SYNC=0, else eth_rx.clk)
  AxiStreamIf.slave  c2e  // tUser = {1'b0,trailing bytes};

);
```

```
module x4xx_mgt_io_core #(
  parameter         PROTOCOL   = `MGT_100GbE,
  parameter [13:0] REG_BASE    = 14'h0,
  parameter         REG_DWIDTH = 32,
  parameter         REG_AWIDTH = 14,
  parameter [ 7:0] PORTNUM     = 8'd0,
  parameter         LANENUM    = 0
) (
  // Resets
  input  logic areset,
  input  logic bus_rst,

  output logic mgt_rst,

  // Clocks
  input  logic clk100,
  input  logic bus_clk,
  input  logic refclk_p,
  input  logic refclk_n,

  output logic mgt_clk,

  // QSFP high-speed IO
  output logic [3:0] tx_p,
  output logic [3:0] tx_n,
  input  logic [3:0] rx_p,
  input  logic [3:0] rx_n,

  // Common signals for single lane 10 GbE
  output logic [0:0] qpll0_reset,
  input  logic [0:0] qpll0_lock,
  input  logic [0:0] qpll0_clk,
  input  logic [0:0] qpll0_refclk,
  output logic [0:0] qpll1_reset,
  input  logic [0:0] qpll1_lock,
  input  logic [0:0] qpll1_clk,
  input  logic [0:0] qpll1_refclk,

  // AXI-Lite
  AxiLiteIf.slave m_axi_mac,

  // Data port
  // Interface clocks on mgt_tx and mgt_rx are NOT used (logic uses mgt_clk).
  AxiStreamIf.slave  mgt_tx,
  AxiStreamIf.master mgt_rx,
  input logic        mgt_pause_req,

  // Register port
  input  logic                  reg_wr_req,
  input  logic [REG_AWIDTH-1:0] reg_wr_addr,
  input  logic [REG_DWIDTH-1:0] reg_wr_data,
  input  logic                  reg_rd_req,
  input  logic [REG_AWIDTH-1:0] reg_rd_addr,
  output logic                  reg_rd_resp,
  output logic [REG_DWIDTH-1:0] reg_rd_data,

  // Misc.
  output logic        rx_rec_clk_out,
  output logic [31:0] port_info,
  output logic        link_up,
  output logic        activity
);
```

FPGA: Timing closure

- Pessimistic placement
- Routing estimation trick

```
#
# Copyright 2021 Ettus Research, a National Instruments Brand
#
# SPDX-License-Identifier: LGPL-3.0-or-later
#

source $::env(VIV_TOOLS_DIR)/scripts/viv_utils.tcl
source $::env(VIV_TOOLS_DIR)/scripts/viv_strategies.tcl

# STEP#1: Create project, add sources, refresh IP
vivado_utils::initialize_project

# STEP#2: Run synthesis


vivado_utils::synthesize_design
vivado_utils::generate_post_synth_reports

# STEP#3: Run implementation strategy
set strategy [vivado_strategies::get_impl_preset "Performance_ExplorePostRoutePhysOpt"]
# Turn up uncertainty on 100Gb clocks(-quiet so if it fails because the clocks don't exist, it won't error)
set_clock_uncertainty 0.5 -quiet -setup [get_clocks txoutclk_out*]
# Vivado has been underestimating routing delays.
dict set strategy "place_design.directive"                  "ExtraNetDelay_high"
# Turn down uncertainty on 100Gb clocks
dict set strategy "route_design.pre_hook" {set_clock_uncertainty 0.0 -quiet -setup [get_clocks txoutclk_out*]}
vivado_strategies::implement_design $strategy

# STEP#4: Generate reports
vivado_utils::generate_post_route_reports

# STEP#5: Generate a bitstream, netlist and debug probes
set_property BITSTREAM.CONFIG.USR_ACCESS TIMESTAMP [get_designs *]
set byte_swap_bin 1
vivado_utils::write_implementation_outputs $byte_swap_bin

# Cleanup
vivado_utils::close_batch_project
```

State of Affairs

- Performance w/ DPDK on fast computer with recommended optimizations
  - From X410 to host: 2 x 400 MHz BW = 32 Gbps
  - From host to X410: 4 x 400 MHz BW = 64 Gbps
- Next Steps
  - Ease FPGA timing closure on 100 GbE + 400 MHz BW
  - Finalize ARM driver and UHD support

.NET API for USRPs

- Demo in C#/LabVIEW for using the multi_usrp recv function
- We are not decided whether we want to spend the effort to create and maintain an additional .NET API for multi_usrp/rfnoc.
- Looking forward for community feedback.