Meet Your USRP Makers

Subhead

Product R&D Team USRP
Jan Schirok
Product Architect
Superhet vs. Zero-IF RF

Superhet (IF): Used in X410

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

<table>
<thead>
<tr>
<th>IF</th>
<th>IQ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advantage</td>
<td>Cleaner in-band spectrum</td>
</tr>
<tr>
<td>Mixers + Synthesizers</td>
<td>X410: 1 or 2 (only 1 for &lt; 3 GHz)</td>
</tr>
<tr>
<td>ADCs/DACs</td>
<td>1 (with high sample rate)</td>
</tr>
<tr>
<td>Complexities</td>
<td>Frequency planning</td>
</tr>
<tr>
<td></td>
<td>RF filters</td>
</tr>
</tbody>
</table>
Ideal In-band Spectrum

Spurs are usually from out-of-band mixing products

IF based

IQ based

Image Rejection

DC offset

In-band spurs based on IQ scheme
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

SCU (STM32 µC)

- 0.85V SPS
- 12 channel ADC
- I2C
- PWR Seq Control
- Fan Control

Output:
- Power Monitor
- Temp Sensor
- Fan
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)

Control for external RF modules
Control for external antennas
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)
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
FPGA: SystemVerilog

Getting to 100 GbE

- Testbenches
- AXI Interfaces
module xdev_cfg_wrapper #
// Must be a value defined in xdev_mgr_types.vh
parameter integer PROTOCOL [5:0] = ('$MT_Disp', 5, 'MT_Disabled',
parameter integer 0),
parameter integer CPU_PWR = 64,
parameter integer PHYS_MCU = 64,
parameter [7:0] PTYRAM = '8'1024,
parameter [15:0] REN0 PROTOUTER = '8'116d18d00)

// Reset
input logic reset,
input logic bus rst,
input logic clk@rst,

// Clocks
input logic refclk_p,
input logic refclk_n,
input logic clk100M,
input logic bus_clk,

// AXI Lite register access
Axilite master slave a_s_ax,

// Ethernet DMA AXI to PS memory
Axilite master slave a_i_hp,

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

// CHDR router interface
Axilstream if master e2v [4],
Axilstream if slave e2v [4],

// ETH DMA IRQs
output logic [3:0] eth_rx_irq,
output logic [3:0] eth_tx_irq,
output logic [3:0] mslc_irq,
output logic [3:0] ptvram_irq,
output logic [3:0] phys_irq,
output logic [3:0] port_irq,
output logic [3:0] link_up,
output logic [3:0] activity
};

module eth_ipv4_interface #
logic [15:0] PROTOCOL,  $protocol [0], 5, PROTOCOL [5:0] = '8'116d18d00,
int CPU_PWR, 6, CPU_PWR [0] = 8, 64,
int PHYS_MCU, 7, PHYS_MCU [0] = 8, 1024,
int NODE_INST, 0, NODE_INST [0] = 0,
int MGT_TBL_SIZE, 3, MGT_TBL_SIZE [0] = 8, 1024,
int REG_MGMT, 0, REG_MGMT [0] = 0,
int BASE, 0, BASE [0] = 0,
int DROP_UNKOWN, 0, DROP_UNKOWN [0] = 0,
int DROP_PKT, 0, DROP_PKT [0] = 0,
int PABLE, 5, PABLE [0] = 0,
int ADD_SOF, 1, ADD_SOF [0] = 1,
int SYNC, 0, SYNC [0] = 0,
int PAUSE, 0, PAUSE [0] = 0,
int NET, 0, NET [0] = 0,
int CPU_PWR, 6, CPU_PWR [0] = 0,
int PHYS_MCU, 7, PHYS_MCU [0] = 0,

// Register port: Write port (domain: bus_clk)
input logic reg_wr, 5, reg_wr [0] = 0,
input logic reg_wr_addr, 5, reg_wr_addr [0] = 0,
input logic reg_wr_data, 5, reg_wr_data [0] = 0,

// Register port: Read port (domain: bus_clk)
input logic reg_rd, 5, reg_rd [0] = 0,
input logic reg_rd_addr, 5, reg_rd_addr [0] = 0,
input logic reg_rd_resp, 5, reg_rd_resp [0] = 0,
input logic reg_rd_data, 5, reg_rd_data [0] = 0,

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

// Ethernet MAC (domain: eth_rx_clk)
output logic eth_pause_req,
Axilstream if master eth_tx, 5, eth_tx [0] = 0,
Axilstream if slave eth_rx, 5, eth_rx [0] = 0,

// CHDR router interface (domain: eth_rx_clk)
Axilstream if master eth_tx, 5, eth_tx [0] = 0,
Axilstream if slave eth_rx, 5, eth_rx [0] = 0,

// CPU CPU (domain: e2c_clk if SYNC=0, else eth_rx_clk)
Axilstream if master e2c, 5, e2c [0] = 0,
Axilstream if slave e2c, 5, e2c [0] = 0,

// Misc
output logic [3:0] reg_clk_out,
output logic [3:0] port_info,
output logic link_up,
output logic activity
};

module xdev_mgmt_if #
parameter PROTOCOL, 5, PROTOCOL [5:0] = '8'116d18d00,
parameter [11:0] REG_BASE, 11, REG_BASE [0] = 0,
parameter REG_MGMT, 5, REG_MGMT [0] = 0,
parameter REG_MGMT_SIZE, 5, REG_MGMT_SIZE [0] = 0,
parameter PORTM, 7, PORTM [0] = 0,
parameter LANE_CNT, 0, LANE_CNT [0] = 0,

// Reset
input logic reset,
input logic bus rst,
input logic mgst rst,

// Clocks
input logic clk100M,
input logic bus clk,
input logic refclk_p,
input logic refclk_n,

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

// Common signals for single lane 10 GbE
output logic [0] gpl0 reset,
output logic [0] gpl0 lock,
output logic [0] gpl0 clk,
output logic [0] gpl0 refclk,
output logic [0] gpl1 reset,
output logic [0] gpl1 lock,
output logic [0] gpl1 clk,
output logic [0] gpl1 refclk,

// AXI Lite
Axilite master slave a_axi,

// Data port
input logic mgst req,
input logic mgst addr_reg_w,
input logic [31:0] mgst addr_reg_r,
input logic mgst data_reg_w,
input logic mgst data_reg_r,

// Register port
input logic reg_wr, 5, reg_wr [0] = 0,
input logic reg_wr_addr, 5, reg_wr_addr [0] = 0,
input logic reg_wr_data, 5, reg_wr_data [0] = 0,
input logic reg_rd, 5, reg_rd [0] = 0,
input logic reg_rd_addr, 5, reg_rd_addr [0] = 0,
input logic reg_rd_resp, 5, reg_rd_resp [0] = 0,
input logic reg_rd_data, 5, reg_rd_data [0] = 0,

// Misc
output logic reg_clk_out,
output logic [3:0] port_info,
output logic link_up,
output logic activity
};
FPGA: Timing closure

- Pessimistic placement
- Routing estimation trick

```tcl
# 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
# STEPS1: Create project, add sources, refresh IP
vivado_utils::initialize_project
# STEPS2: Run synthesis
vivado_utils::synthesize_design
vivado_utils::generate_post_synth_reports
# STEPS3: Run implementation strategy
set_strategy [vivado_strategies::set_impl preset “Performance_ExplorePostRoutePhyOpt”]
# Turn on uncertainty on 10GE 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 10GE clocks
dict set_strategy “route_design_pre_hook” [set_clock_uncertainty 0.0 -quiet -setup [get_clocks txoutclk_out*]]
vivado_strategies::implement_design $strategy
# STEPS4: Generate reports
vivado_utils::generate_post_route_reports
# STEPS5: Generate a bitstream, netlist and debug probes
set property BITSTREAM.COMFIG.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.