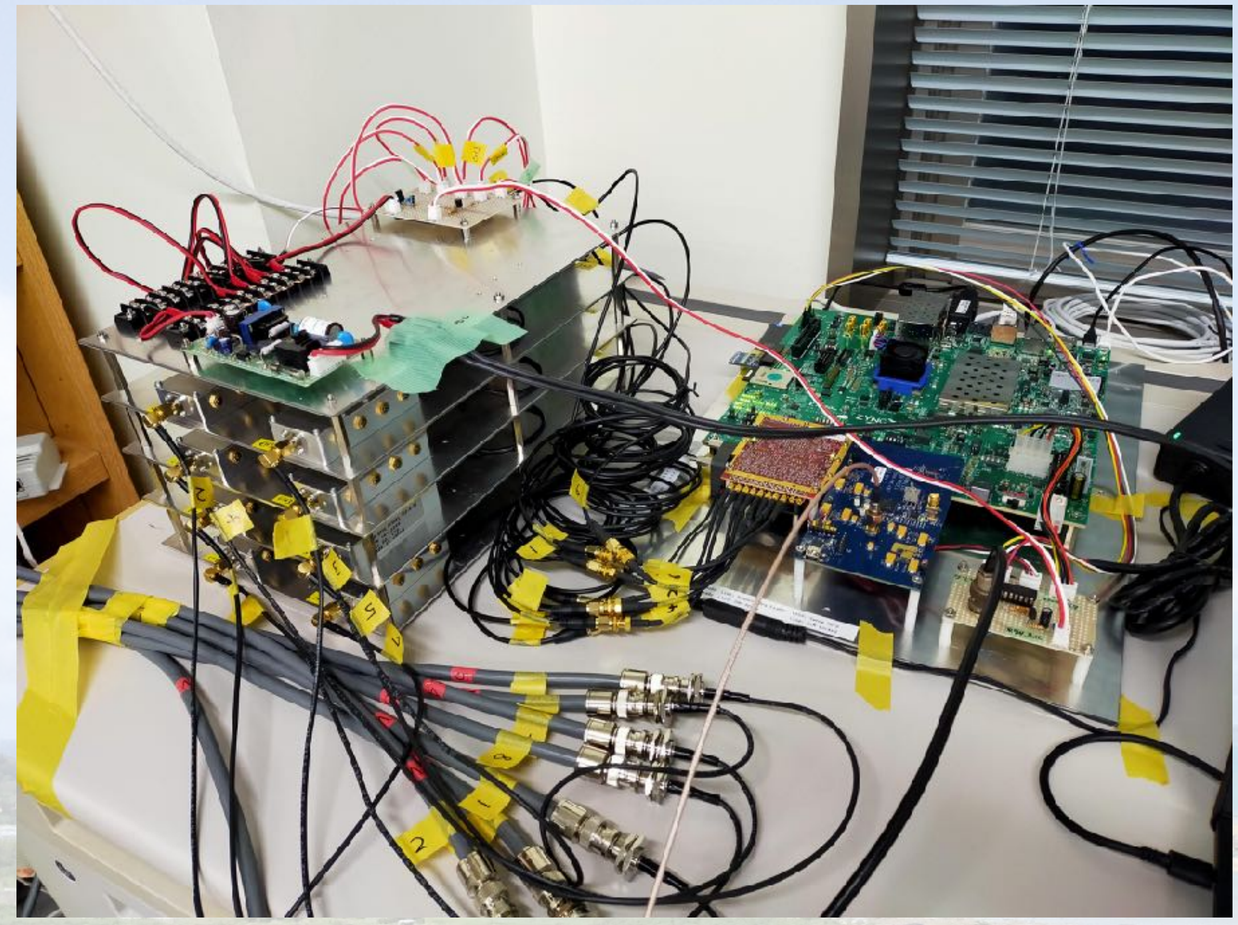
Clustering Algorithm on Xilinx Alveo U50

Yasunori Osana @ University of the Ryukyus

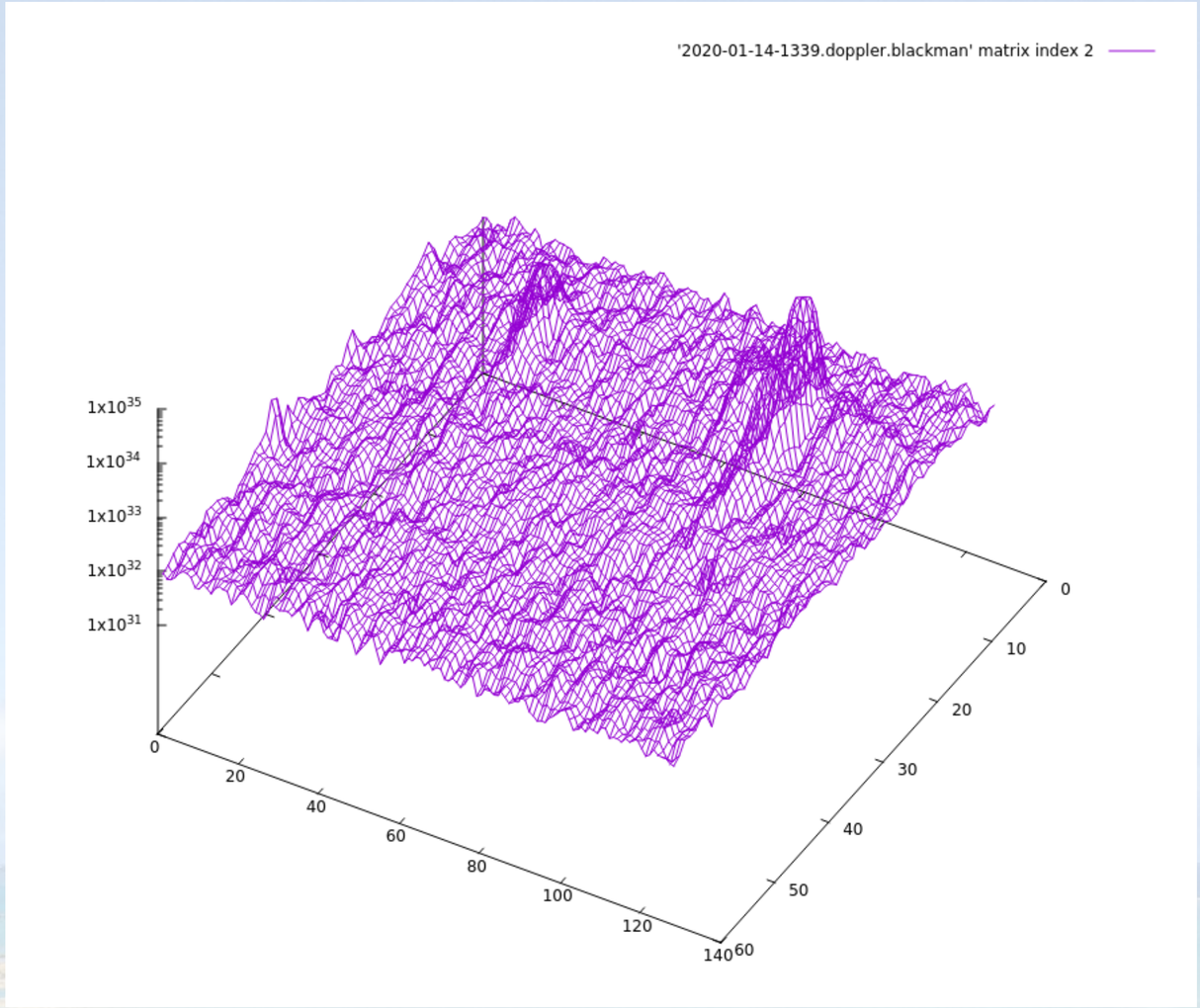
Recent work: HF ocean radar



Rx Antenna array

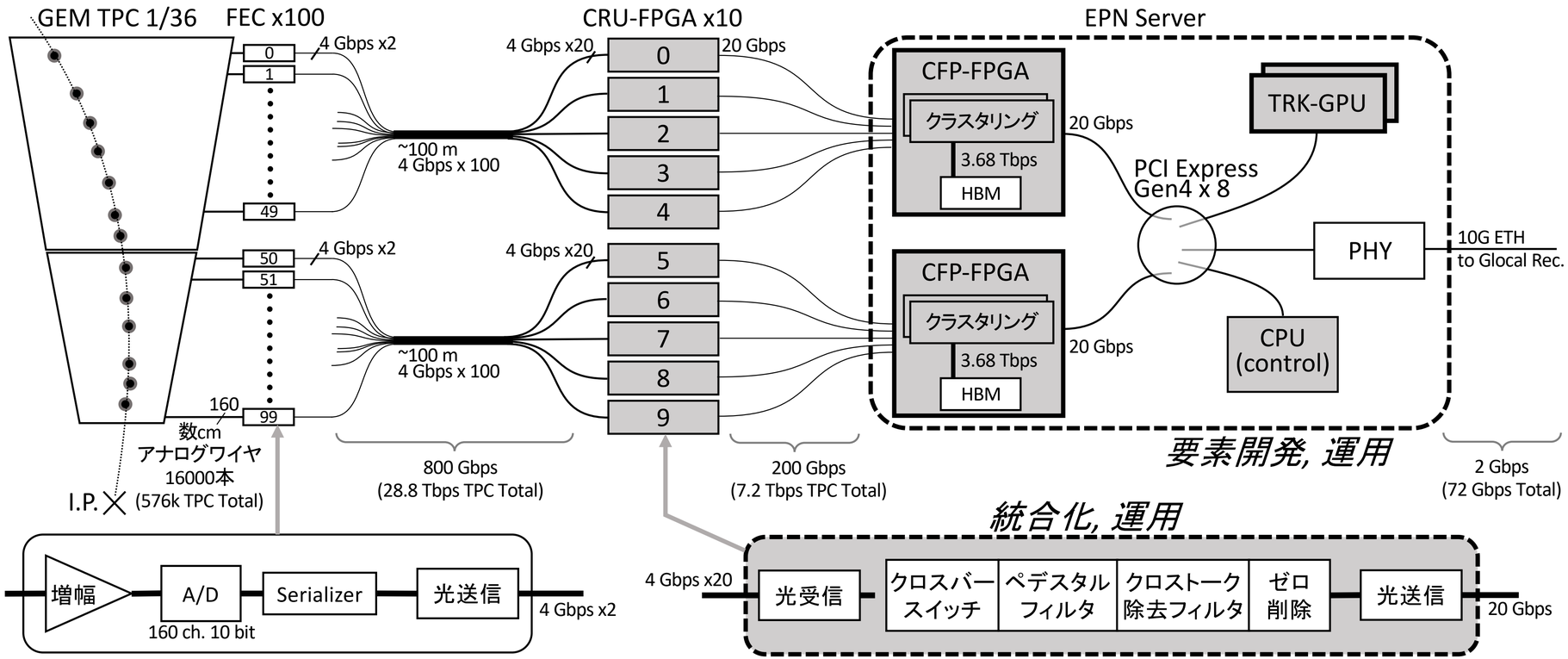
HF ocean radar

- Xilinx ZynqMP SoC (ARM64+FPGA)
- 2Gsps ADC x1 + 50W Tx amp
- 250Msps DAC x8
  - 40Gbps Rx signal, online DSP filters



Alveo U50

- FPGA acceleration card for datacenters
  - Equivalent to Xilinx Ultrascale+ XCVU35P
  - 1 QSFP28 (100GbE capable) port
  - PCIe Gen3 x16, Gen4 x8 or CCIX
  - 2x 4GB HBM on chip


GEM TPC 1/36
FEC x100
CRU-FPGA x10
EPN Server
4 Gbps x2
4 Gbps x20
20 Gbps
~100 m
4 Gbps x 100
CFP-FPGA
クラスタリング
3.68 Tbps
HBM
20 Gbps
TRK-GPU
PCI Express
Gen4 x 8
PHY
10G ETH
to Glocal Rec.
CPU
(control)
160
数cm
アナログワイヤ
16000本
(576k TPC Total)
I.P.
800 Gbps
(28.8 Tbps TPC Total)
200 Gbps
(7.2 Tbps TPC Total)
2 Gbps
(72 Gbps Total)
要素開発, 運用
統合化, 運用
増幅
A/D
Serializer
光送信
160 ch. 10 bit
光受信
クロスバー
スイッチ
ペデスタル
フィルタ
クロストーク
除去フィルタ
ゼロ
削除

Arria 10 vs U50

	Arria10 GX1150	Alveo U50/XCVU35P
ALMs (8-LUTs) / 6-LUTs	427,200	871,680
FFs	1,708,800	1,743,360
M20K/RAM36K	2,713	1,344
UltraRAM (144K)	-	640
18x19 / 28x18 multiplier	3,036	2,880

HBM: High-Bandwidth Memory

- Stacked DRAM
  - Good: Large and very fast
  - Bad: Unpredictable latency
- Physically 8ch
  - x2 pseudo ch each
- 1024 DDR signals at 450MHz
  = 256 SDR x 8_{phys} x 2_{pseudo}

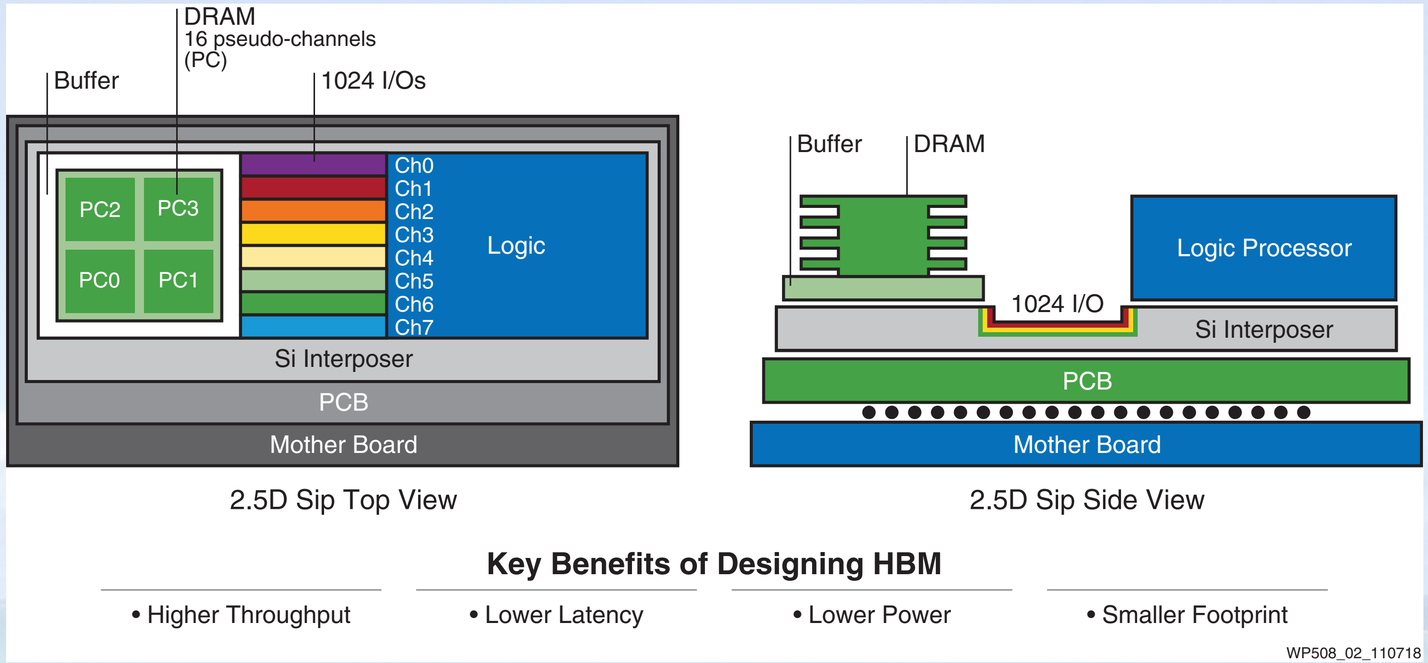
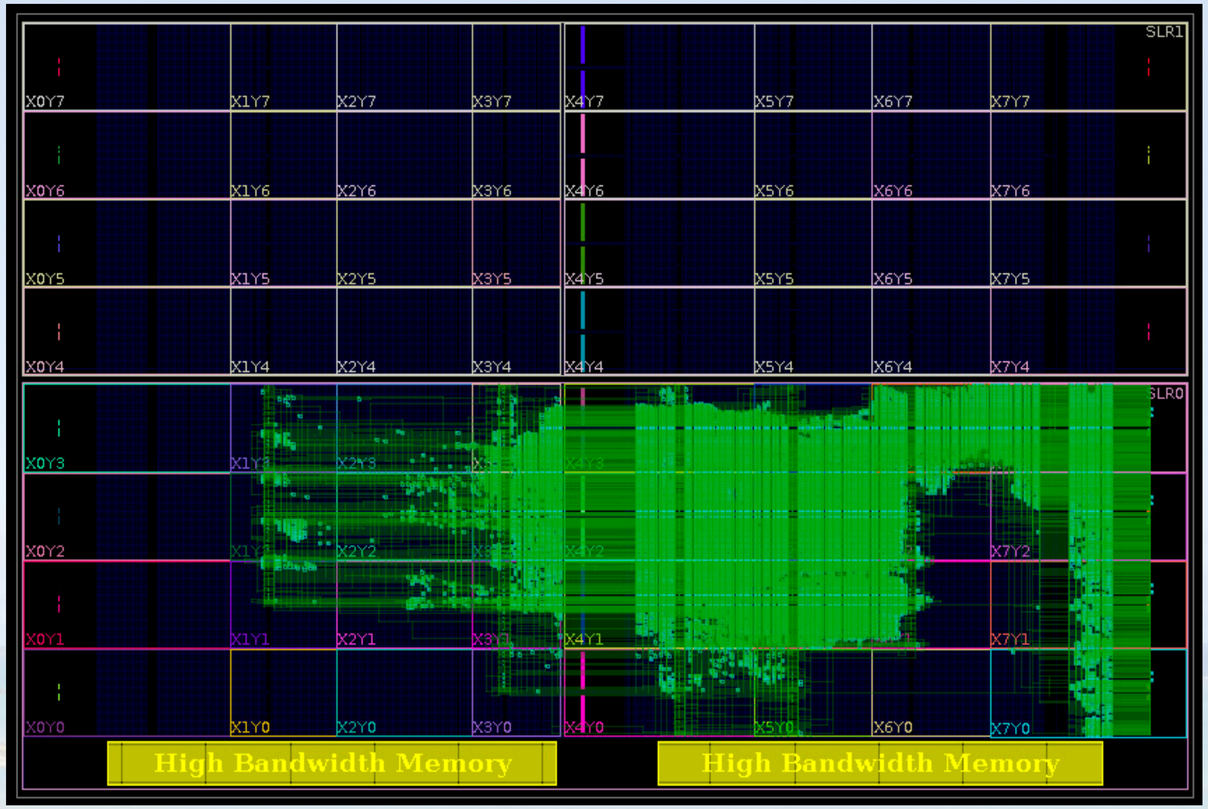


Image from Xilinx WP508

Device Layout

- Has 2 FPGA dies: SLR0 and 1
  - SLR = super logic region
  - 20k wires between SLRs
- PCIe: SLR0 right (4ch x 4 = 16)
- QSFP: SLR1 left top (4ch)
- Bottom: 2 HBM stacks



XDMA: Xilinx PCIe DMA Engine

- Supports PCIe Gen3 x16 and Gen4 x8 = 128Gbps
  - on FPGA side, 512bit @ 250MHz = 128Gbps
  - AXI stream or memory mapped interface

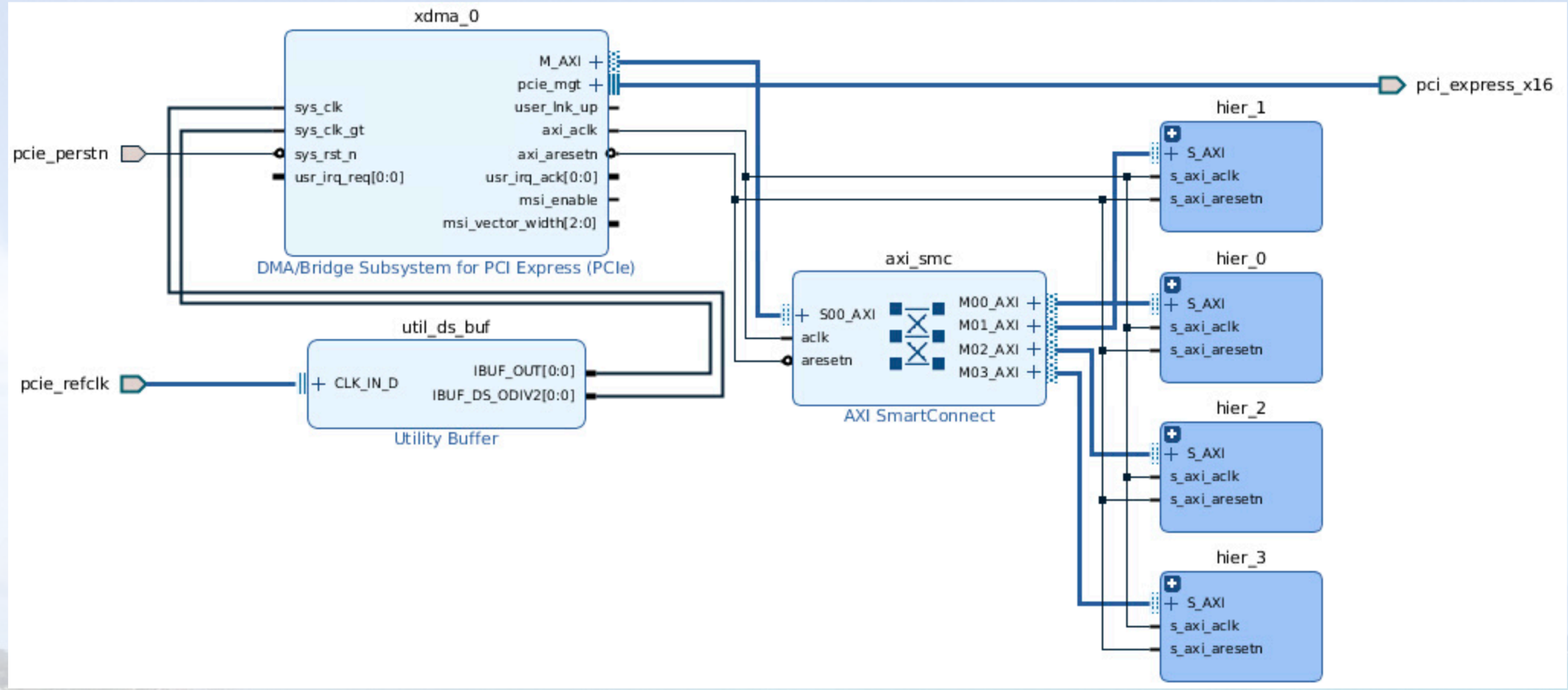
Xilinx design tools

- Vivado: HDL design and IP block-based design
- Vivado HLS: High-level synthesis from C and C++, part of Vivado design suite
- ~~Vitis: SW-HW co-design environment for x86+FPGA~~
  - Vitis includes Vivado and Vivado HLS

Xilinx IP Interface

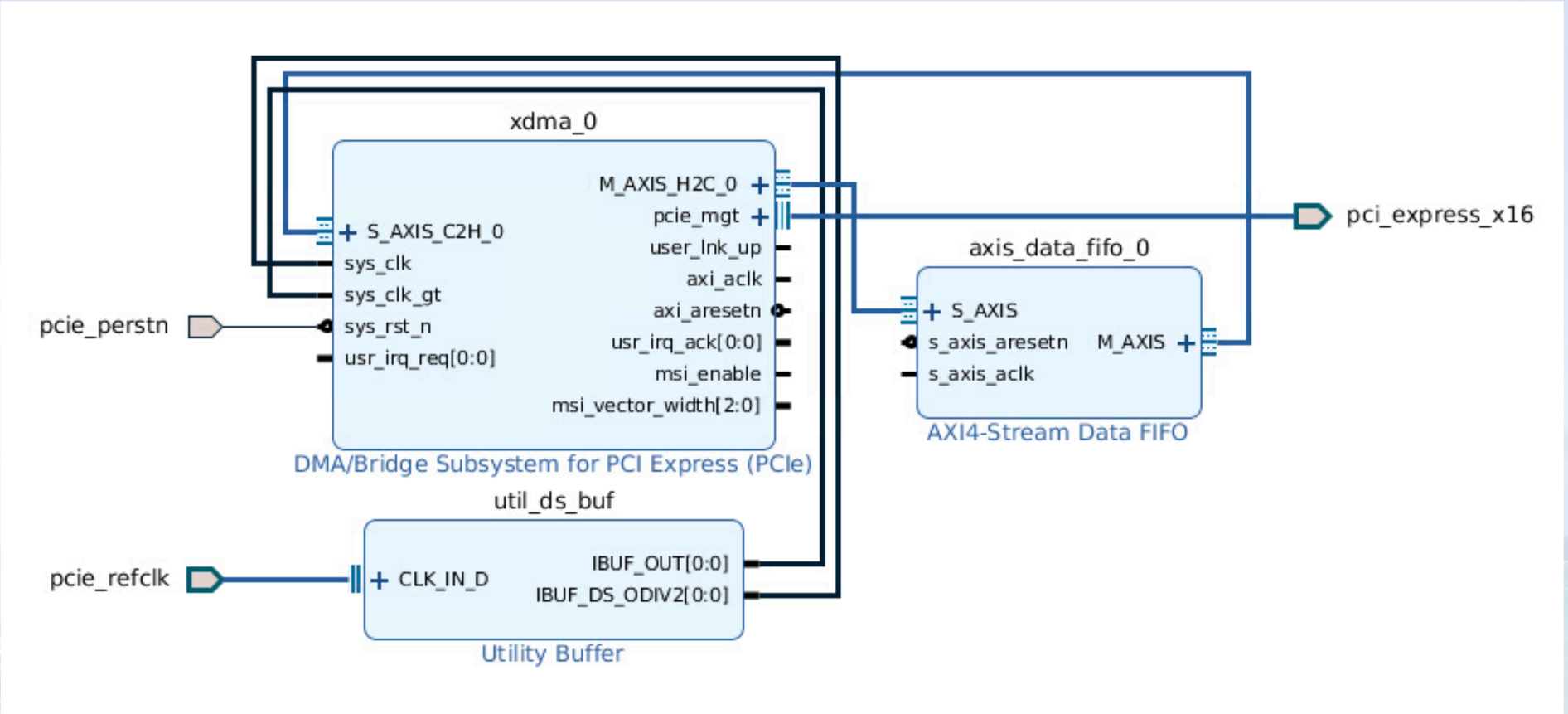
- Many IP cores based on AMBA AXI4 interface protocol: the ARM standard
  - AXI4: Full featured bus with address + data, with burst transfer
  - AXI4-Lite: Simplified bus without burst mode
  - AXI4-Stream: Point-to-point, unidirectional data stream interface w/o address
    - AXI4-Stream is easy for HDL coders, while AXI4 is not easy to implement
- Very similar to Avalon-MM and Avalon-ST in Intel FPGA

AXI4 design example: XDMA + RAM



Cell	Slave Interface	Slave Segment	Offset Address	Range	High Address
xdma_0					
M_AXI (64 address bits : 16E)					
hier_0/axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0000_0000	2M	0x0000_0000_001F_FFFF
hier_1/axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0020_0000	2M	0x0000_0000_003F_FFFF
hier_2/axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0040_0000	2M	0x0000_0000_005F_FFFF
hier_3/axi_bram_ctrl_0	S_AXI	Mem0	0x0000_0000_0060_0000	2M	0x0000_0000_007F_FFFF

AXI Stream design example



Vivado HLS (and almost same in Intel HLS)

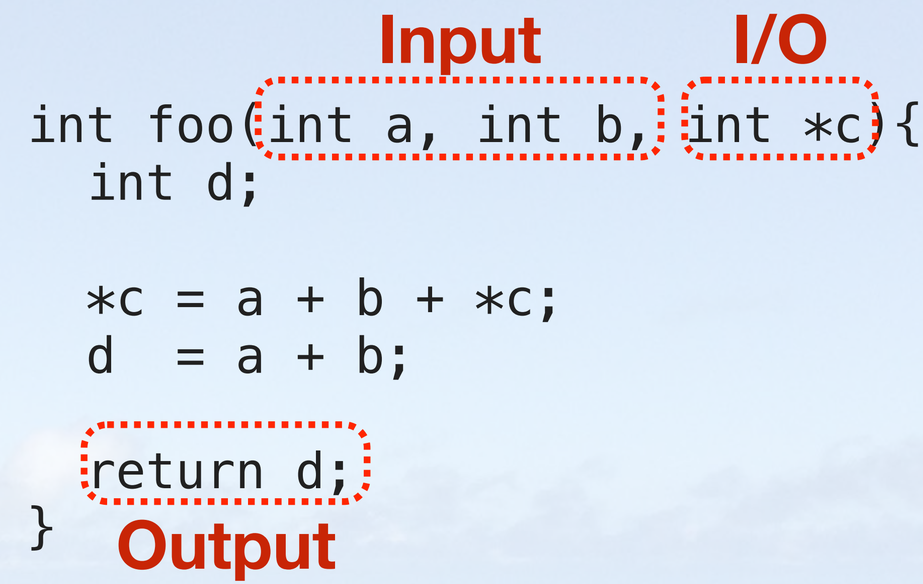
- Most of C/C++ syntax is synthesizable
  - Functions, loops, arrays and structs / classes including C++ templates
  - Double, single and half precision floating point support
  - Pointer and recursion are partially supported
- Hardware-friendly extensions: arbitrary precision variables, FIFO streams, …
- IP library support: linear algebra, DSP, image/video, …

Vivado HLS design flow

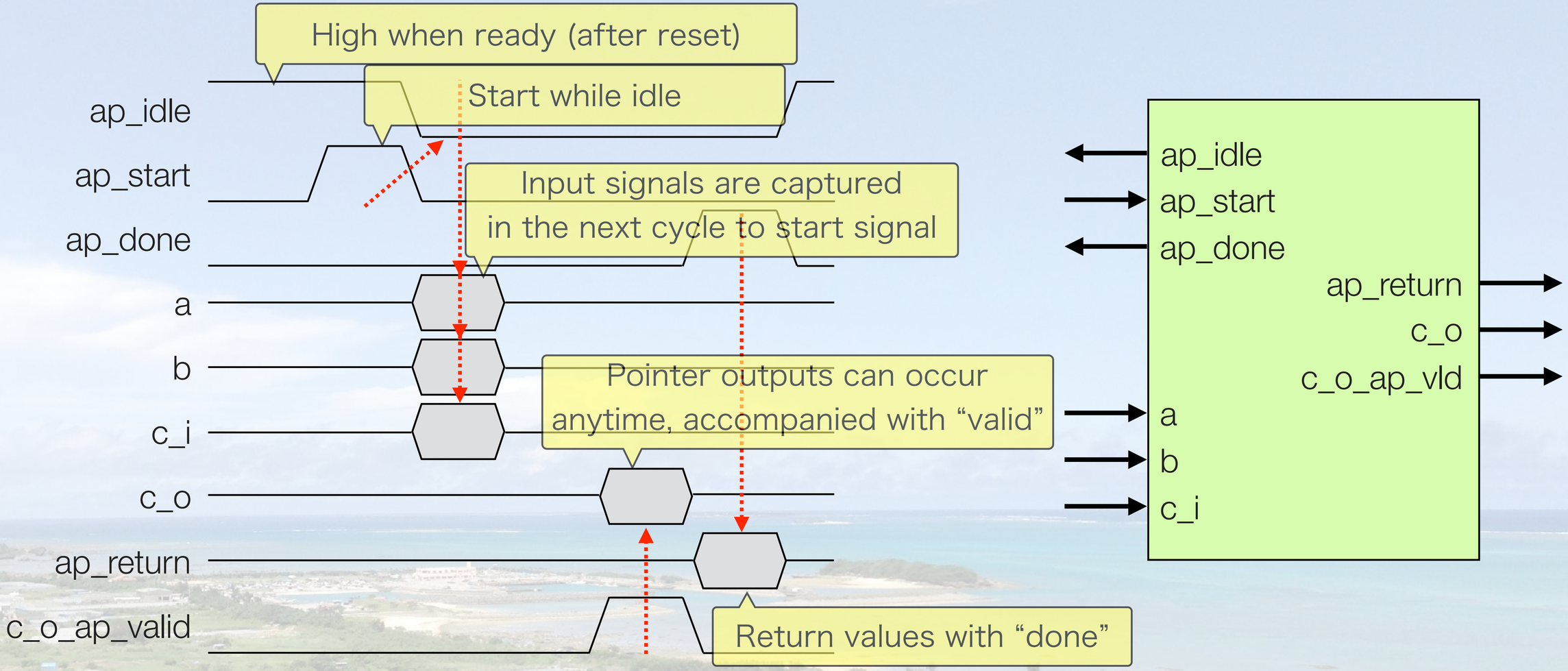
- Write C/C++ code, testbench in main()
- Compile and debug with gcc, g++ or any other standard compiler
- Launch Vivado HLS and synthesize the top-level function: an IP core is generated
  - Do performance / area optimization at this point
- Create Vivado RTL project, with the HLS-generated IP core included

Module I/O

- Arguments are basically input
- Pointer arguments can be output
- Return values are always output



Basic I/O Protocol



HLS design Examples

- C++ lib: Arbitrary precision types and Stream object
- Interface synthesis
- Performance optimization

Arbitrary Precision Types

- Available types:
  - Integer (signed, unsigned)
  - Fixed-point
- All basic operators in C/C++ are supported

```
#include “ap_int.h”

…

ap_int<3>  three_bit_signed;
ap_uint<7> seven_bit_unsigned;

ap_uint<1000> very_wide;

…

// bit range operations
int a = very_wide.range(131, 100);
very_wide(63, 32) = b;
```

Stream object: hls::stream

- Synthesized as:
  - FIFO buffer (in a function)
  - Shift register (if the length is fixed)
  - FIFO interface (as a function argument)

```
#include “hls_stream.h”

…

hls::stream<int> int_stream;

…

int_stream.write(3);

…

int a = int_stream.read()
```

Interface synthesis: stream I/O (1)

- Example: vector sum
- Pass-by-reference is synthesized to
  - a FIFO interface
  - or an AXI4-Stream port

```
int vec_sum(hls::stream<int>& in){
  int sum;

  for (int i=0; i<10; i++)
    sum += in.read();

  return sum;
}
```

Interface synthesis: stream I/O (2)

- Example: vector add
- Multiple stream ports are OK
  - Birectional port is NOT allowed
- Port handshake (throttling) is fully automated!

```
void vec_add(hls::stream<int>& a,
             hls::stream<int>& b,
             hls::stream<int>& s){

  for (int i=0; i<10; i++)
    s.write(a.read() + b.read());
}
```

Interface synthesis: Memory I/O

- Arrays in function argument can be memory controller interface
  - BRAM and AXI (DDR / HBM)
  - Multiple arrays can be combined into single large memory
  - RAM interface width can be configured

```
void foo ( int array_on_ram[1000], … );
```

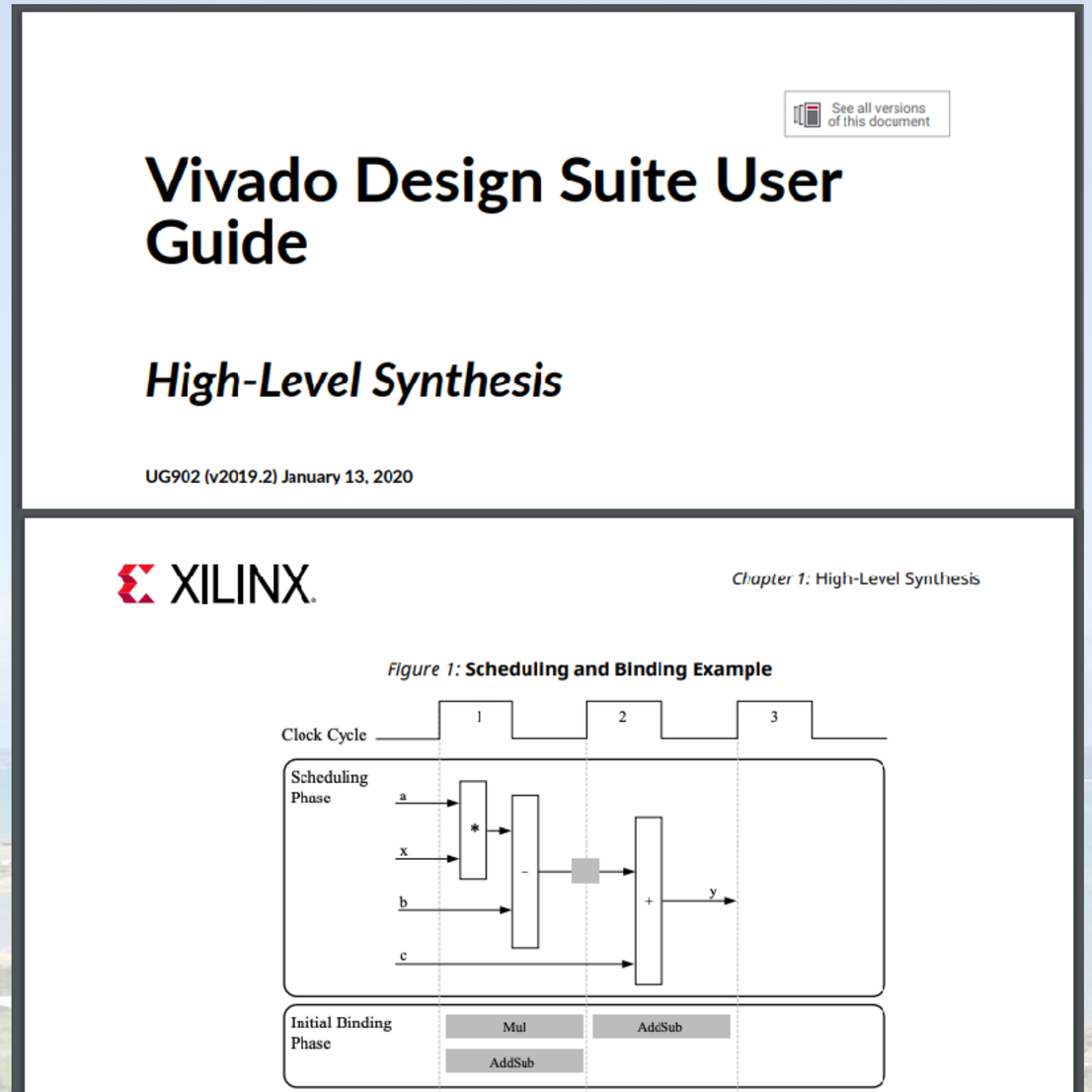
Performance tuning

- Directive-based tuning
  - Loop pipeline, unroll and merge
  - Array reshaping
  - and many other features

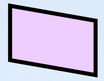
```
int vec_sum(hls::stream<int>& in){
  int sum;

  for (int i=0; i<10; i++)
#pragma HLS PIPELINE
    sum += in.read();

  return sum;
}
```

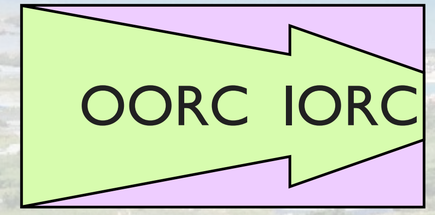
Xilinx UG902: the HLS cookbook

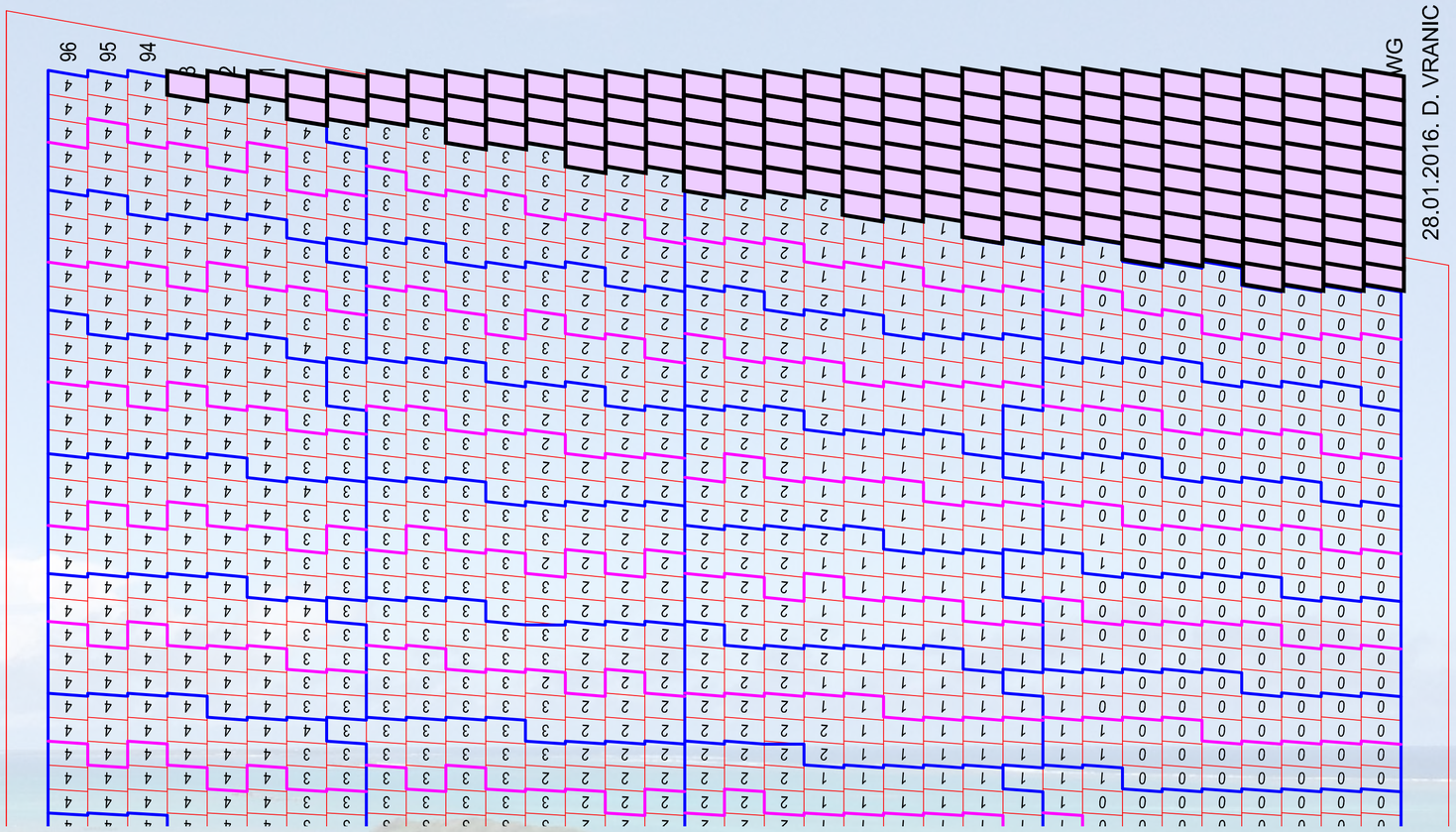
- Useful even for Intel HLS users!
  - Describes how HLS works in detail
  - Available in English and Japanese



Cluster finder: test version

- Add “zero” pads 
- Reshape ROC rectangle
  - 152 rows x 138 cols





Performance requirement

- 2 U50 cards for 16,000ch
  - 8,000ch at 5Msps = 8,000ch in 80clk at 400MHz
  - 100ch / clk = 1,000 bit / clk = not too wide
    - 14 parallel BRAM/URAM is sufficient, no need of HBM
- 3D stencil operation of 5x5 (spatial) x5 (temporal)
  - I/O stream width = 1,380bit (= 138 pads in OROC3)

2D version

- 1,380 bit stream-in and out
- 5x5 average calculation:
  cluster math still not implemented
- Fully pipelined in just <6% of SLR
  - Clk estimation: 450+ MHz
- Just a straightforward C++ code,
  took 1 day to make it work

Detail

Instance

Loop

	Latency (cycles)			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- row_loop	156	156	6	1	1	152	yes

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	138	-	-	-
Expression	-	-	0	28261	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	84	-
Register	0	-	11116	32	-
Total	0	138	11116	28377	0
Available	2688	5952	1743360	871680	640
Available SLR	1344	2976	871680	435840	320
Utilization (%)	0	2	~0	3	0
Utilization SLR (%)	0	4	1	6	0

2D version: code

* See the terminal…

Optimizations

- Stencil line buffer: `ap_uint<10> buf[5][COLS];`
  - 2D array of 5 rows, split into independent registers for parallel access

```
#pragma HLS ARRAY_PARTITION variable=buf complete dim=2
#pragma HLS ARRAY_PARTITION variable=buf complete dim=1
```

  - Innermost loop unrolled, all 138 pads calculated at once

```
    for (int c=0; c<COLS; c++){
#pragma HLS UNROLL
```

3D version

- Just piled-up the 2D version
- 1,380x5 bit stream-in and out
  - not synthesized in 3hr…
- Half-size version: 720x5 bit in/out
  - in <13% of SLR, 450+MHz
  - 2 modules in parallel is possible

Latency

Summary

Latency (cycles)		Latency (absolute)		Interval (cycles)		
min	max	min	max	min	max	Type
167	167	0.417 us	0.417 us	167	167	none

Detail

Instance

Loop

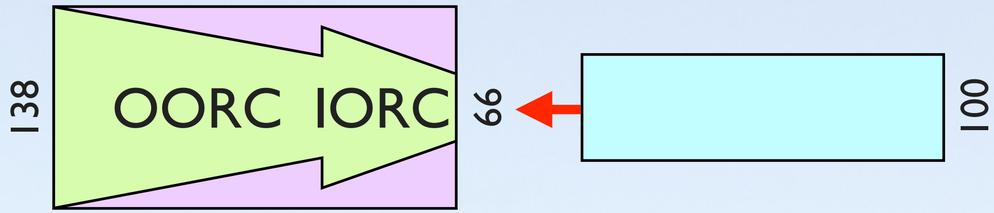
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	288	-	-	-
Expression	-	-	0	53617	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	102	-
Register	0	-	36891	6944	-
Total	0	288	36891	60663	0
Available	2688	5952	1743360	871680	640
Available SLR	1344	2976	871680	435840	320
Utilization (%)	0	4	2	6	0
Utilization SLR (%)	0	9	4	13	0

Cluster Finder I/O

- Without “zero” pads
  - Requires pad mapping ROM and shifters
- Feeding 400Gbps (400MHz x 100pads x 10bit)
  - Fixed 100 pad width to ROC width
  - Mapping is statically determined



Pad remapping

- Pre-calculated static mapping
  - Stored in 2D array, as a ROM

```
Word 0
 Mapped 0–65 for row 0 0–65 push
 Mapped+ 66–99 for row 1 0–33
Word 1
 Mapped 0–31 for row 1 34–65 push
 Mapped 32–97 for row 2 0–65 push
 Mapped+ 98–99 for row 3 0–1
Word 2
 Mapped 0–65 for row 3 2–67 push
 Mapped+ 66–99 for row 4 0–33
Word 3
 Mapped 0–33 for row 4 34–67 push
 Mapped+ 34–99 for row 5 0–65
Word 4
 Mapped 0–1 for row 5 66–67 push
 Mapped 2–71 for row 6 0–69 push
 Mapped+ 72–99 for row 7 0–27a
```

I/O stream converter

- Significant LUT consumption… :(
  - Because of shifters to handle 100-to-138 conversion
  - Optimization may be possible, but don't know how to do in HLS
  - Implementation is possible anyway

Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	38994	-
FIFO	-	-	-	-	-
Instance	-	-	-	-	-
Memory	7	-	20	54	-
Multiplexer	-	-	-	93	-
Register	0	-	10045	160	-
Total	7	0	10065	39301	0
Available	2688	5952	1743360	871680	640
Available SLR	1344	2976	871680	435840	320
Utilization (%)	~0	0	~0	4	0
Utilization SLR (%)	~0	0	1	9	0

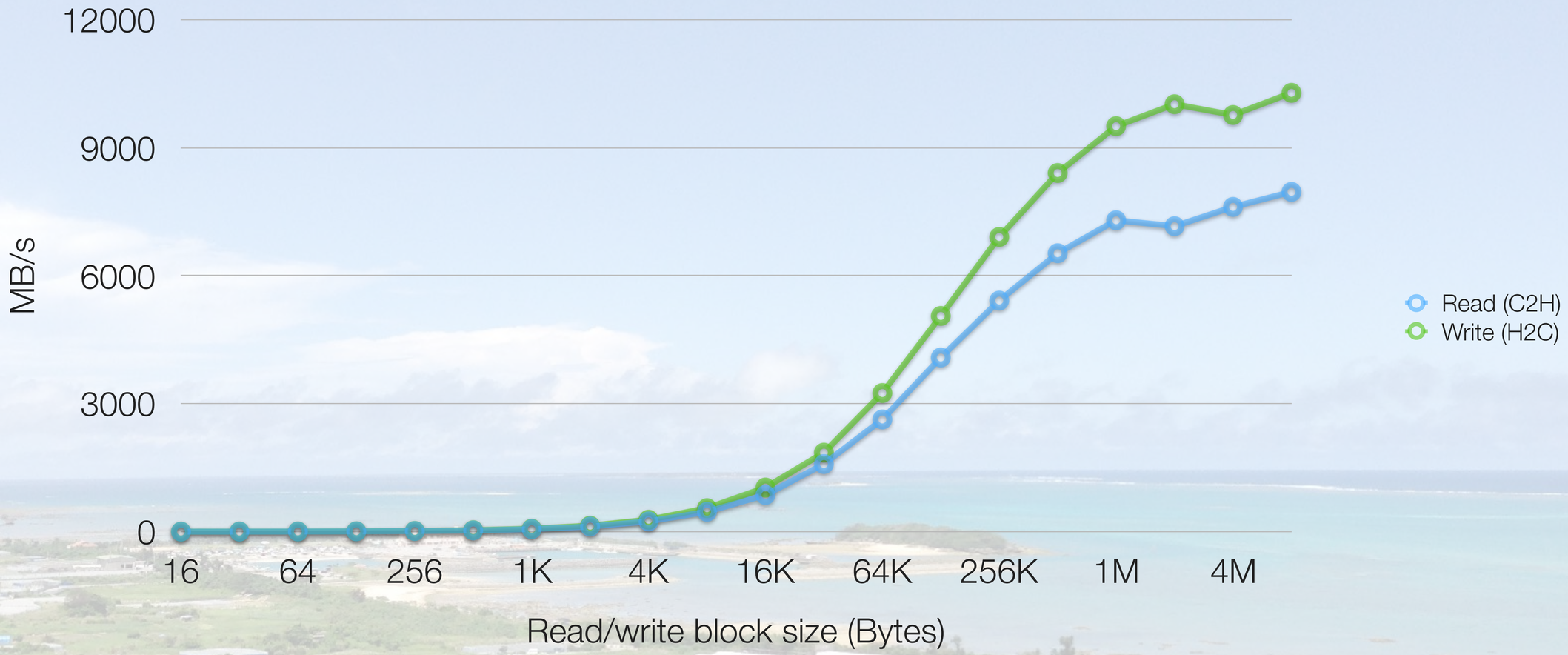
XDMA Performance test

- Core i3-6100 3.7GHz
  - DDR4-2133 dual channel
  - PCIe Gen3 x16
  - Ubuntu Linux 18.04 (kernel 4.15)
- DMA to/from 8MB BlockRAM

With Xilinx's reference code:

```
Write Performance
** Average BW = 8388608, 10296.580078

Readback and Compare
** Average BW = 8388608, 4462.804199

Read Performance
** Average BW = 8388608, 9191.814453
```

XDMA Performance test (my code)



Current situation & Problems

- Clustering on U50 is possible, if I/O interfaces are implemented
- I/O between CRU:
  - 100GbE, or should we design our own protocol?
  - 5 CRU - 4 serial links on U50: Possible PCIe connector hack
- Cluster finding is not actually implemented: circuit may be larger

Data rate problem

- CRU: Arria10 GX = 12.5Gbps transceivers, supports 10GbE soft MAC
- U50: 28Gbps transceiver x4, supports 100GbE hard MAC
- 5x CRU vs 1x U50… mmm

100GbE switch?

- Not super-expensive
- But comes with:
  - Unpredictable delay and jitter among data links…
  - Don’t love Ethernet MAC :(

www.nttxstore.jp

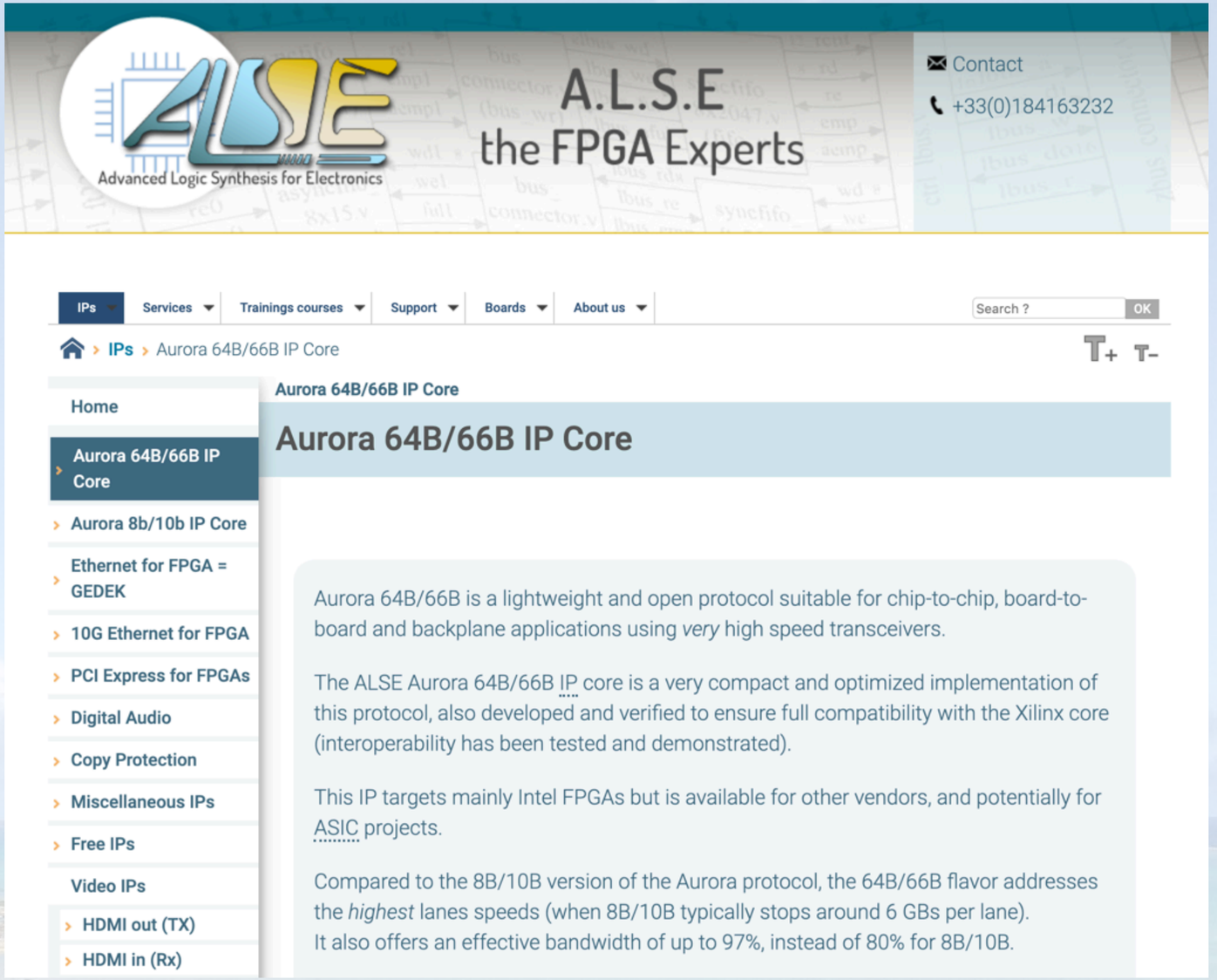
Possible hack on PCIe edge

- With Gen4 PCIe, 8 transceivers on the PCIe are not used
  - Designing custom bifurcation adapter…?
    - Special board to host SFP modules
  - Direct link to 5 or more CRUs
  - Need Intel-Xilinx high-speed link protocol

PCIe extender cable:
thermaltake.com

Aurora 64b66b

- Xilinx version of SerialLite
  - Provides FIFO-style simple transceiver channel
  - Intel version is available commercially
  - Makes I to X link with no effort
  - But don’t know the price



Interlaken

- Available both in Arria10 GX and U50
  - but not sure about the protocol