Next generation electrophysiology

Interface standard proposal for
very high data rate neurophysiology
and low latency closed-loop experiments

SWC Workshop Next Gen (Open) Ephys: Hardware and Software
May 25-26th 2016, London

Requirements

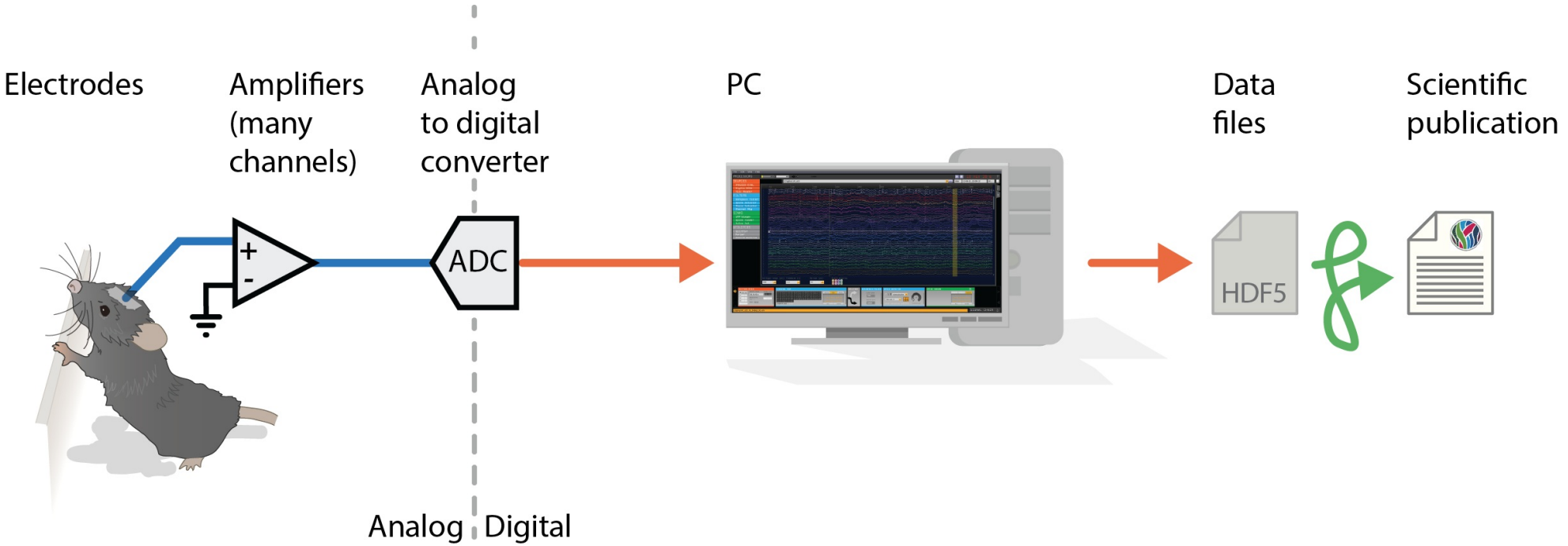
> 1000 channels	NeuroPix, Neuroseeker, Miniscope, etc. ~10 Gbps, more later
< 1ms latency	Closed loop experiments, whole cell, etc.

Ideally also

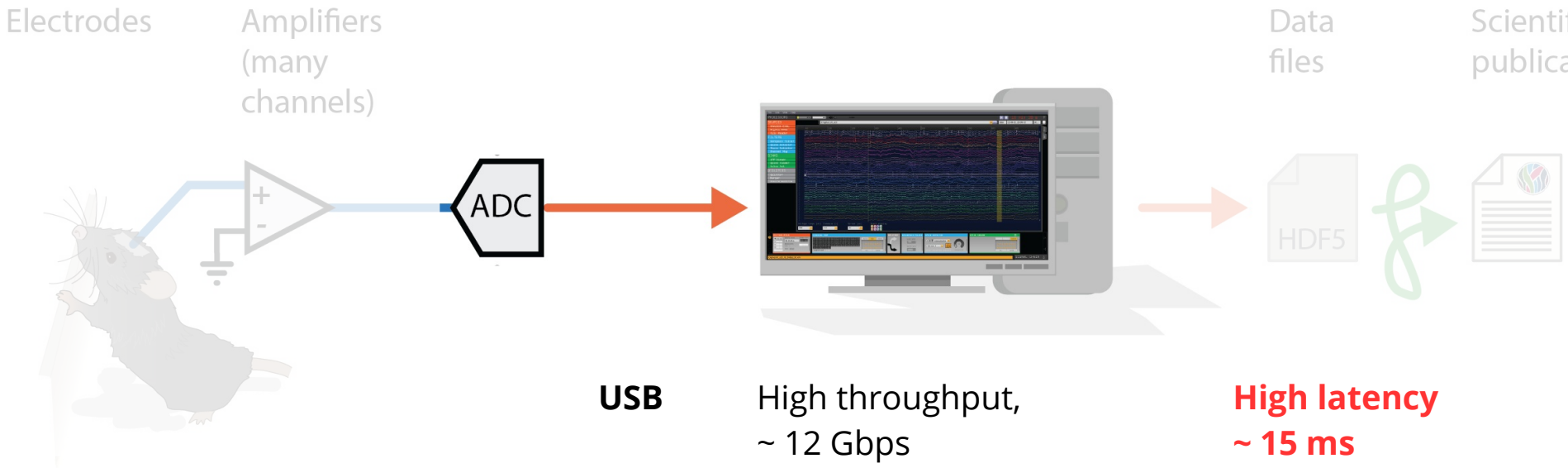
Dynamic clamp capable: < 100μs closed loop latency to user sw

Future proof, modular, not reliant on vendor specific components

What design choices result from this?



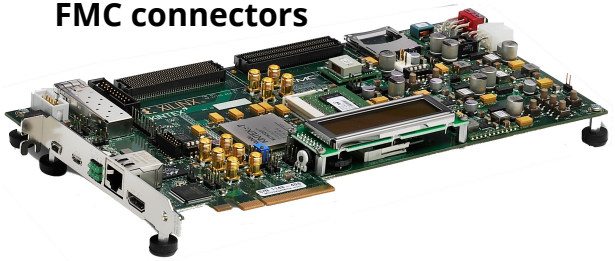
Host PC interface



USB	High throughput, ~ 12 Gbps	**High latency ~ 15 ms**
Ethernet	High throughput, ~12 Gbps	**Low latency ~ 1ms**
PCIe	Highest throughput, ~12 Gbps (Max. ~100)	**Lowest latency ~ 10 µs**

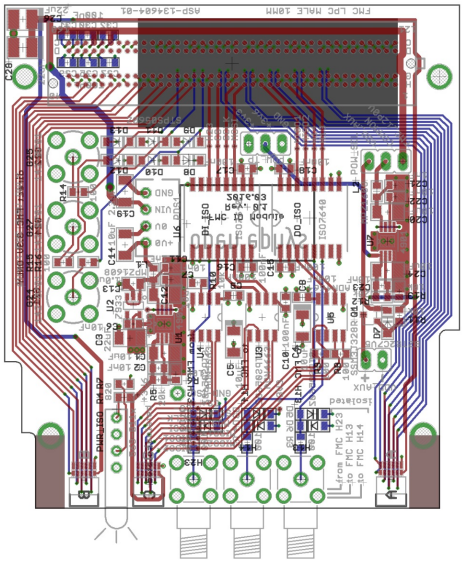
proof of principle: PCIe prototype

FMC connectors



Kintex 7 eval
PCIe 2 x4, $1600

FMC connector



Direct PCIe DMA interface

> 1000 channels @ 30KHz
(approx. max. throughput is
25000 ch. * 16 bit * 30kHz = 12 Gb/s = 1.5 GB/s
Fills 1TB in ~11 minutes)

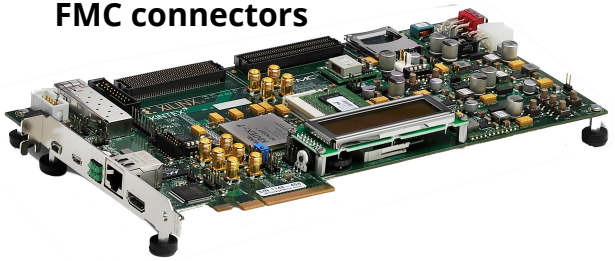
Aaron:
Direct port of Reid's Rhythm firmware & API via Xillybus

Jon & Jakob:
Simple FMC test board
1 isolated intan SPI connector,
1 non-isolated

3 isolated DIO (1 out, 2 in)
3 direct DIO (bidirectional)

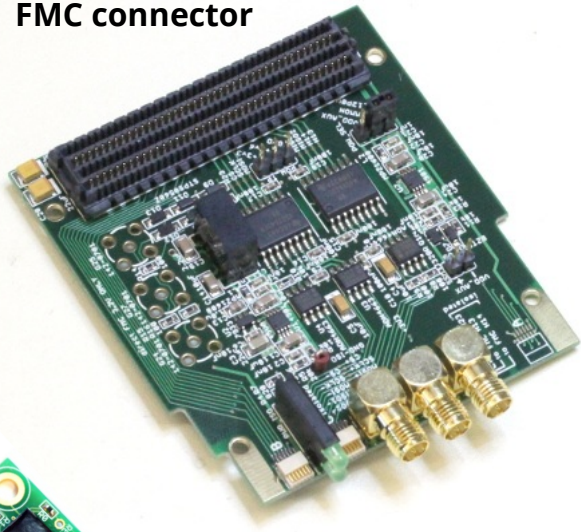
proof of principle: PCIe prototype

FMC connectors



Kintex 7 eval
PCIe 2 x4, $1600

FMC connector



Intan Headstage

Direct PCIe DMA interface

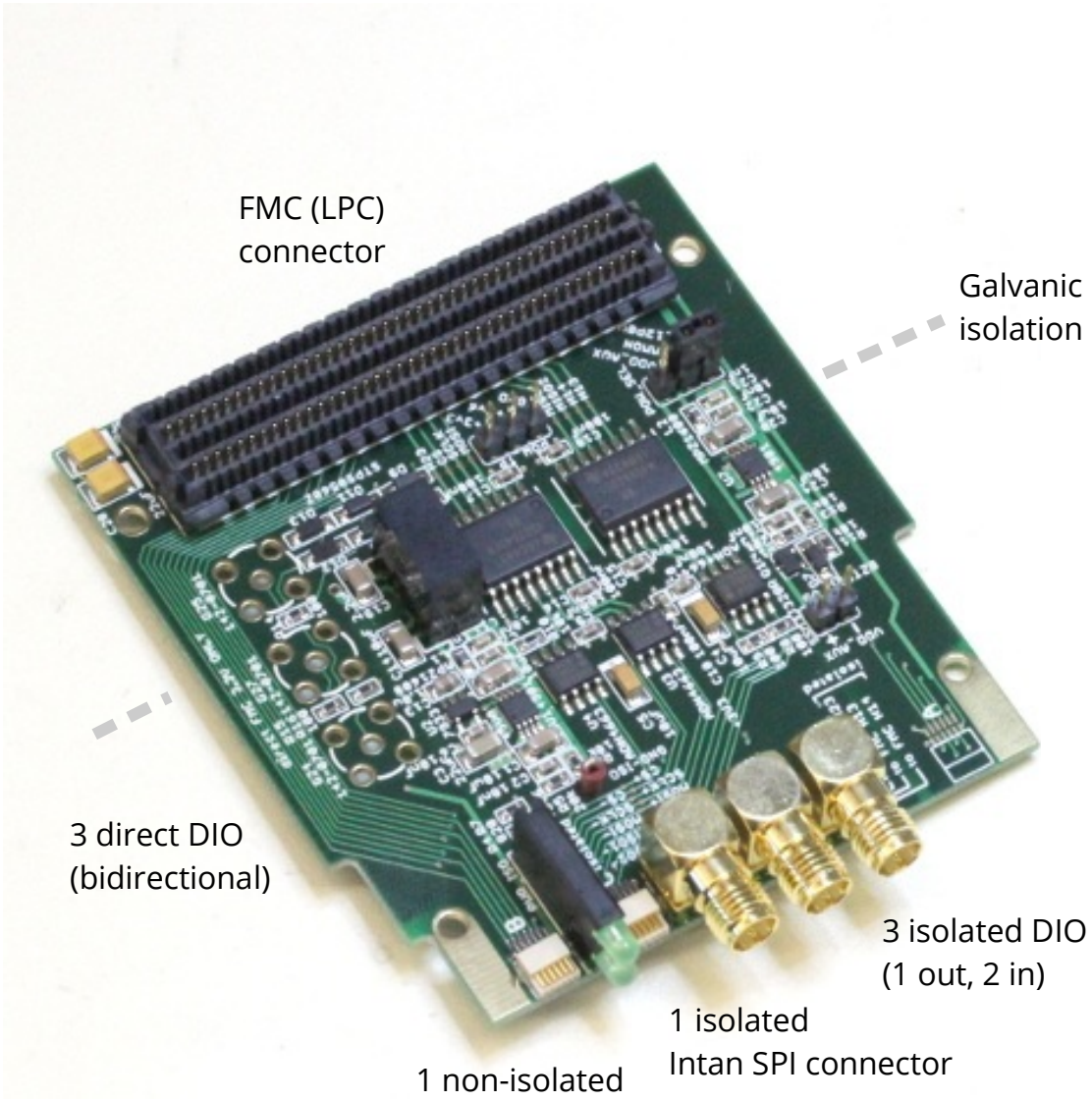
> 1000 channels @ 30KHz
(approx. max. throughput is
25000 ch. * 16 bit * 30kHz = 12 Gb/s = 1.5 GB/s
Fills 1TB in ~11 minutes)

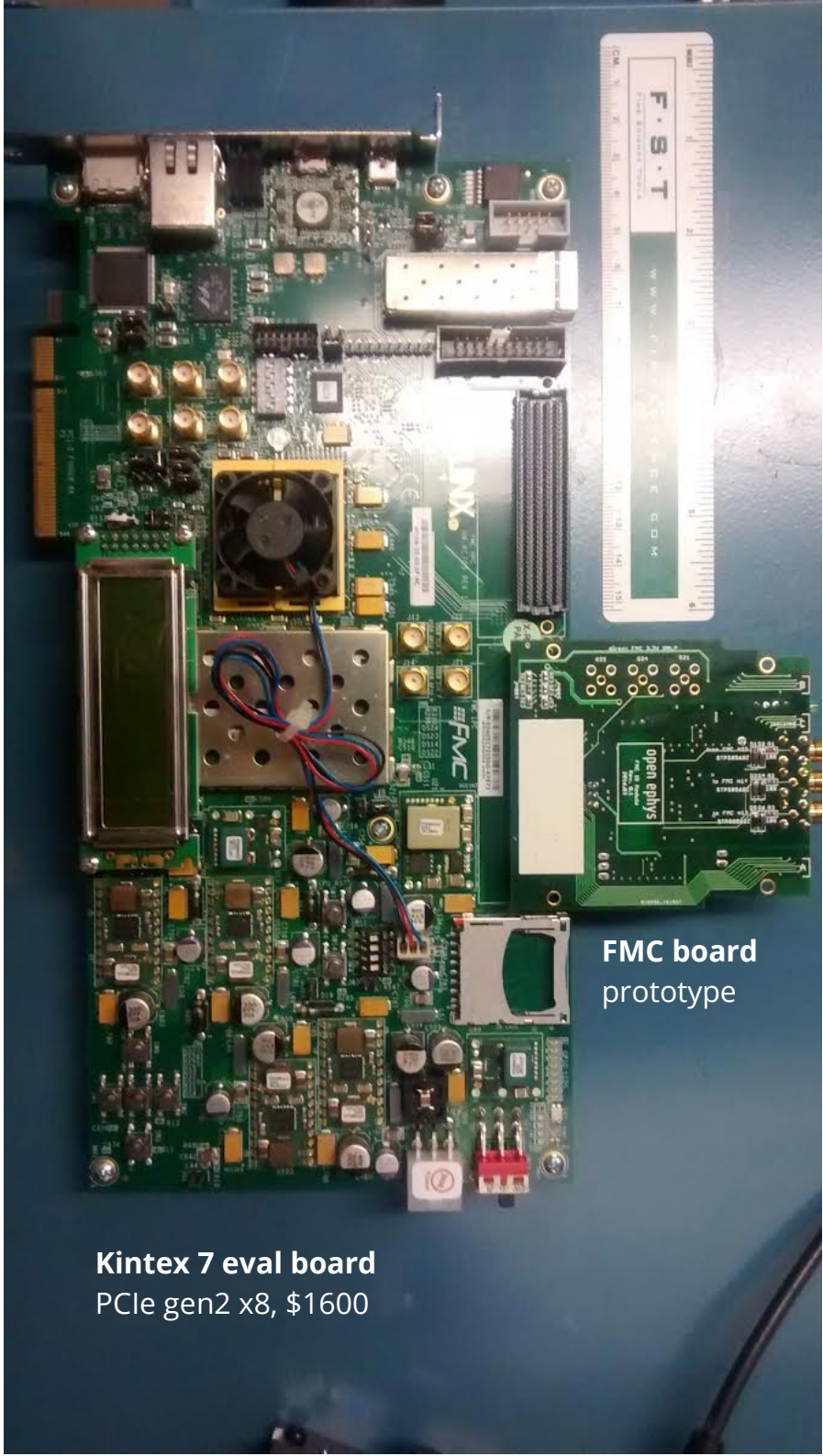
Aaron:
Direct port of Reid's Rhythm firmware & API via Xillybus

Jon & Jakob:
Simple FMC test board
1 isolated intan SPI connector,
1 non-isolated

3 isolated DIO (1 out, 2 in)
3 direct DIO (bidirectional)

PCIe prototype





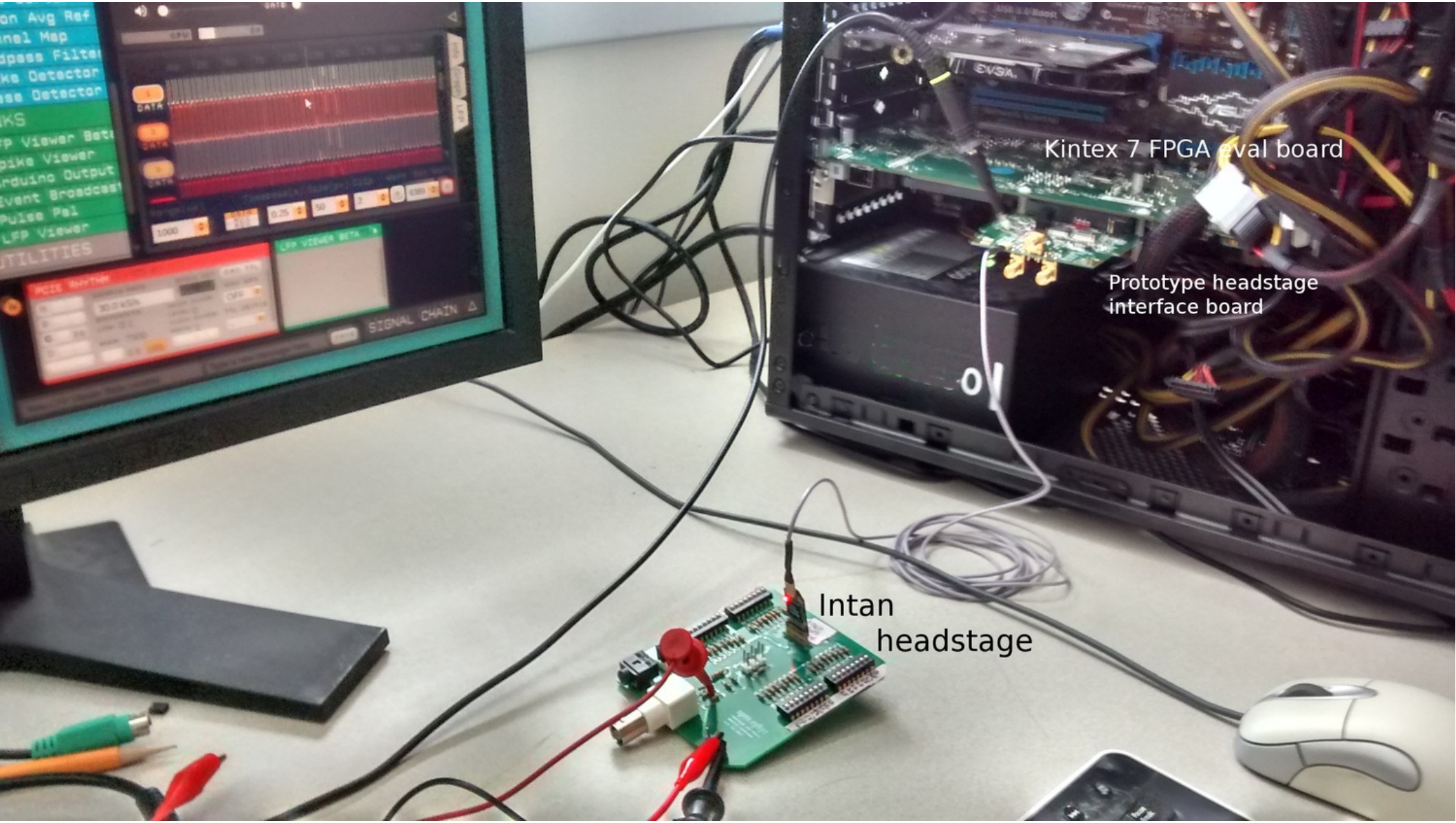
Latency test

```
1444        // do the neural data channels first
1445        for (int dataStream = 0; dataStream < numStreams; dataStream++)
1446        {
1447            int nChans = numChannelsPerDataStream[dataStream];
1448            chanIndex = index + 2*dataStream;
1449            if ((chipId[dataStream] == CHIP_ID_RHD2132) && (nChans == 16)) //RHD2132 16ch. headstage
1450            {
1451                chanIndex += 2 * RHD2132_16CH_OFFSET*numStreams;
1452            }
1453            for (int chan = 0; chan < nChans; chan++)
1454            {
1455                channel++;
1456                thisSample[channel] = float(*(uint16*)(bufferPtr + chanIndex) - 32768)*0.195f;
1457                chanIndex += 2*numStreams;
1458                if (dataStream == 0 && chan == 0) //First channel of the first enabled stream
1459                {
1460                    bool check = (thisSample[channel] > THRESHOLD_CHECK);
1461                    evalBoard->setOuputSigs(check ? 0x0001 : 0x0000); // set pin high/low
1462                }
1463            }
1464        }
1465        index += 64 * numStreams;
1466        //now we can do the aux channels
1467        auxIndex += 2*numStreams;
1468        for (int dataStream = 0; dataStream < numStreams; dataStream++)
1469        {
1470            if (chipId[dataStream] != CHIP_ID_RHD2164_B)
1471            {
1472                int auxNum = (auxSamp+3) % 4;
1473                auxSamp = (++auxSamp) % 4;
1474                if (auxNum < 3)
1475                {
1476                    auxSamples[dataStream][auxNum] = float(*(uint16*)(bufferPtr + auxIndex) - 32768)*0.0000374;
1477                }
1478                for (int chan = 0; chan < 3; chan++)
1479                {
1480                    channel++;
1481                    if (auxNum == 3)
1482                    {
1483                        auxBuffer[channel] = auxSamples[dataStream][chan];
1484                    }
```

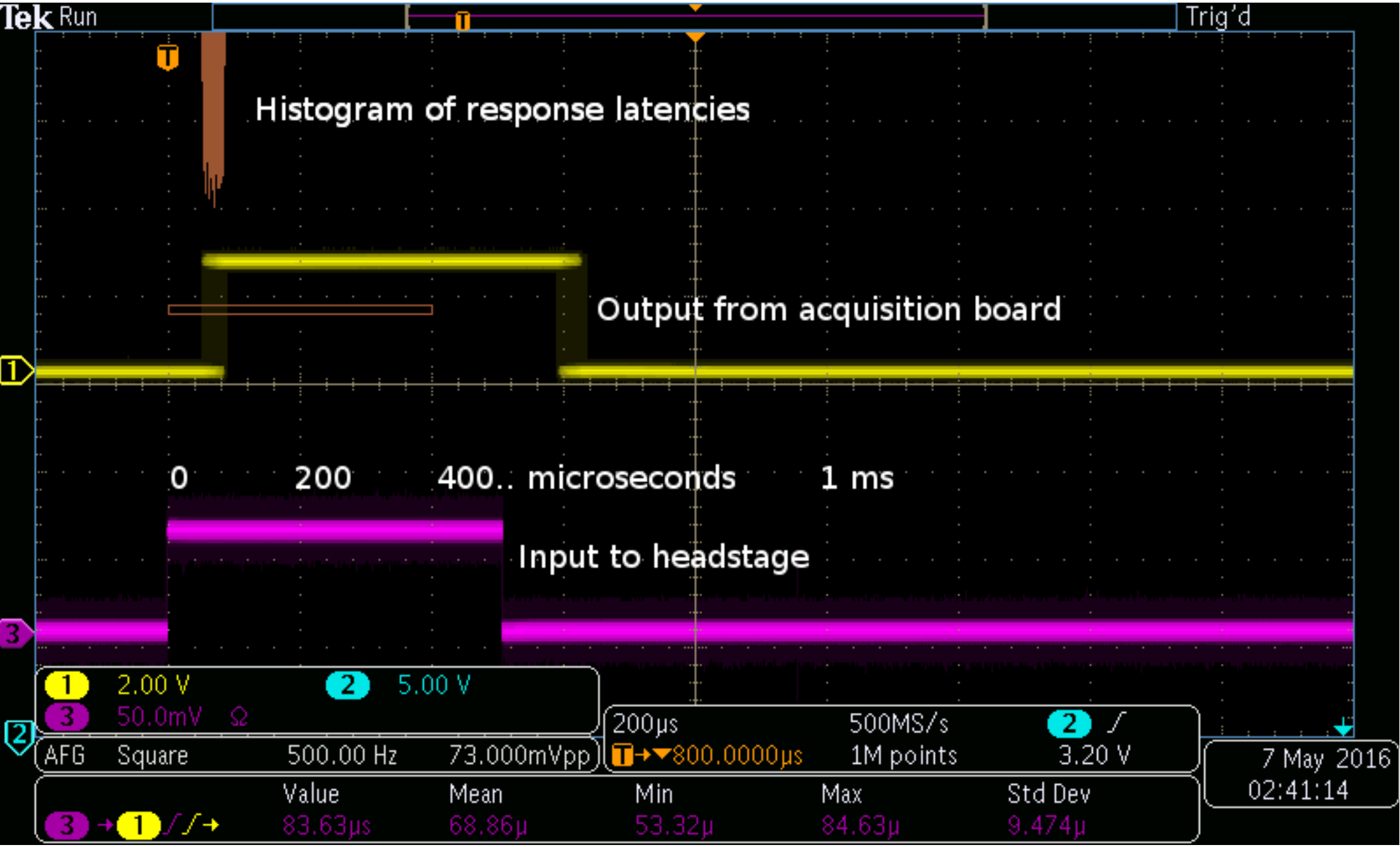
Latency test

```
1444        // do the neural data channels first
1445        for (int dataStream = 0; dataStream < numStreams; dataStream++)
1446        {
1447            int nChans = numChannelsPerDataStream[dataStream];
1448            chanIndex = index + 2*dataStream;
1449            if ((chipId[dataStream] == CHIP_ID_RHD2132) && (nChans == 16)) //RHD2132 16ch. headstage
1450            {
1451                chanIndex += 2 * RHD2132_16CH_OFFSET*numStreams;
1452            }
1453            for (int chan = 0; chan < nChans; chan++)
1454            {
1455                channel++;
1456                thisSample[channel] = float(*(uint16*)(bufferPtr + chanIndex) - 32768)*0.195f;
1457                chanIndex += 2*numStreams;
1458                if (dataStream == 0 && chan == 0) //First channel of the first enabled stream
1459                {
1460                    bool check = (thisSample[channel] > THRESHOLD_CHECK);
1461                    evalBoard->setOuputSigs(check ? 0x0001 : 0x0000); // set pin high/low
1462                }
1463            }
1464        }
1465        index += 64 * numStreams;
1466        //now we can do the aux channels
1467        auxIndex += 2*numStreams;
1468        for (int dataStream = 0; dataStream < numStreams; dataStream++)
1469        {
1470            if (chipId[dataStream] != CHIP_ID_RHD2164_B)
1471            {
1472                int auxNum = (auxSamp+3) % 4;
1473                auxSamp = (++auxSamp) % 4;
1474                if (auxNum < 3)
1475                {
1476                    auxSamples[dataStream][auxNum] = float(*(uint16*)(bufferPtr + auxIndex) - 32768)*0.0000374;
1477                }
1478                for (int chan = 0; chan < 3; chan++)
1479                {
1480                    channel++;
1481                    if (auxNum == 3)
1482                    {
1483                        auxBuffer[channel] = auxSamples[dataStream][chan];
1484                    }
```

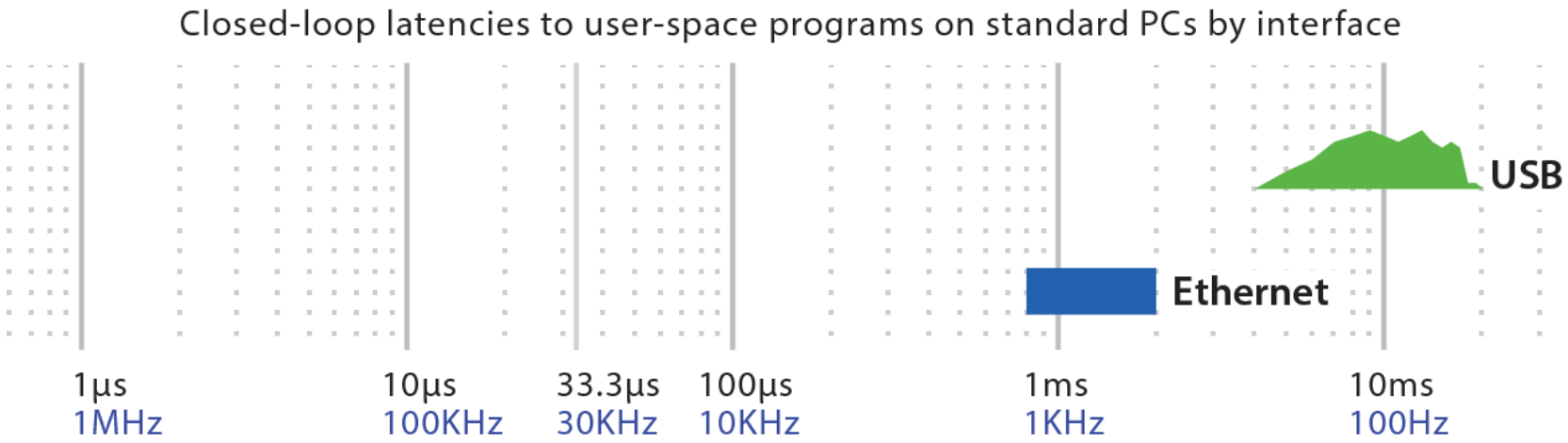
Latency test



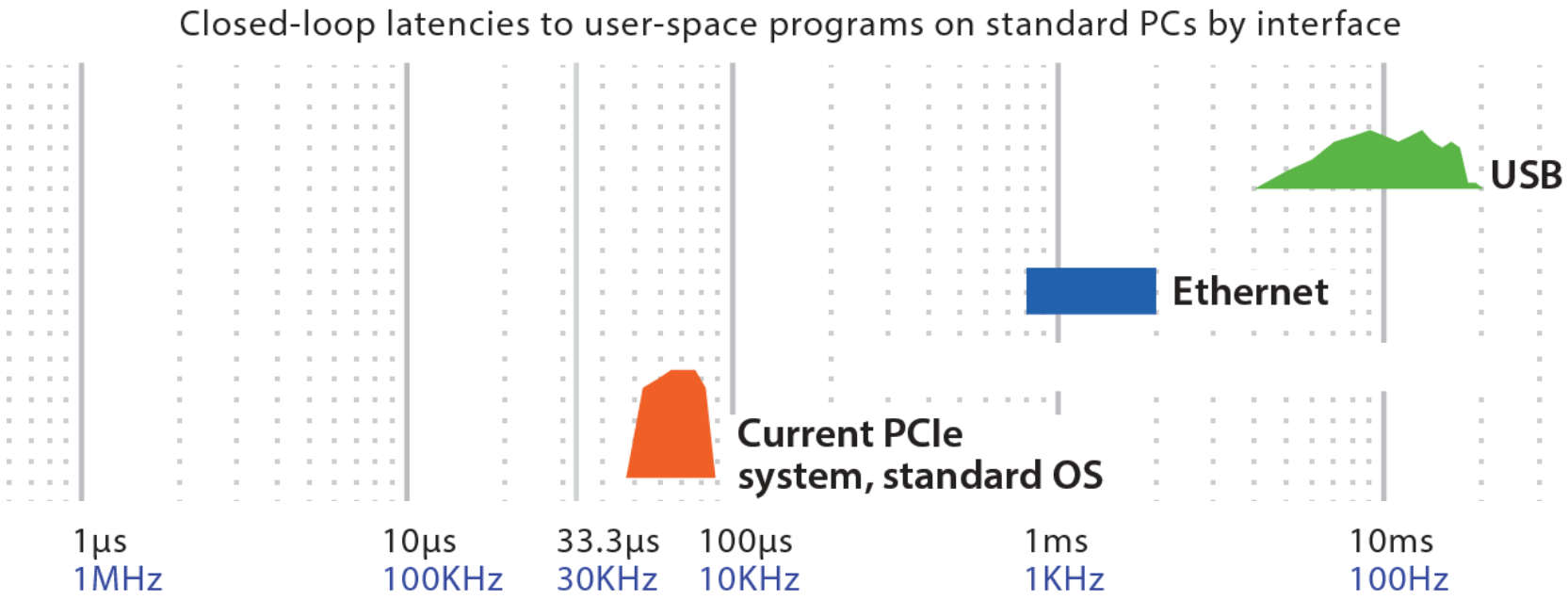
Latency results



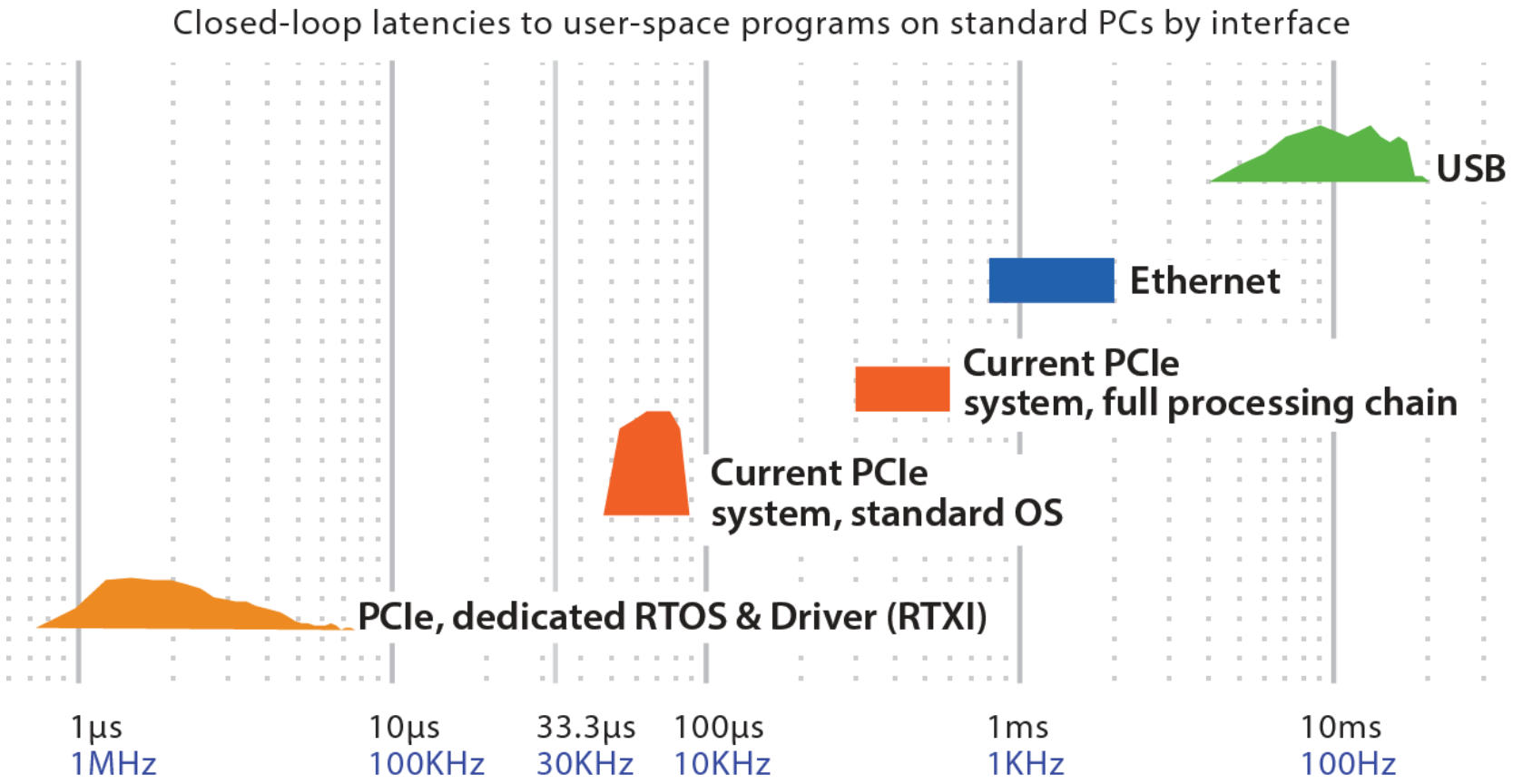
Latency results



Latency results



Latency results

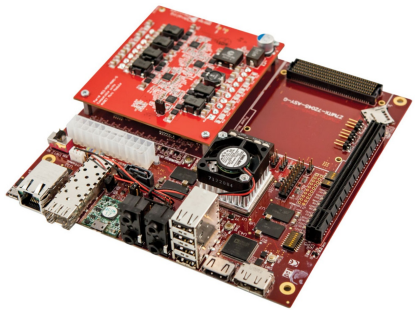


→ PCIe is sufficiently easy to implement,
Provides µs latency, at almost arbitrary channel counts.

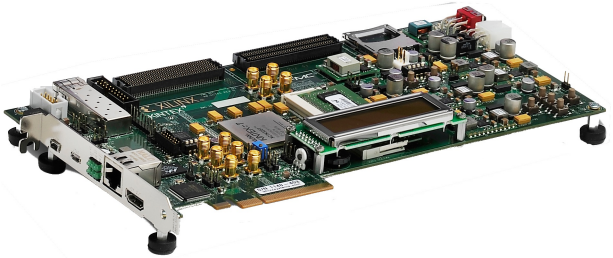
Which FPGA/PCIe card?

→ PCIe is sufficiently easy to implement,
Provides µs latency, at almost arbitrary channel counts.

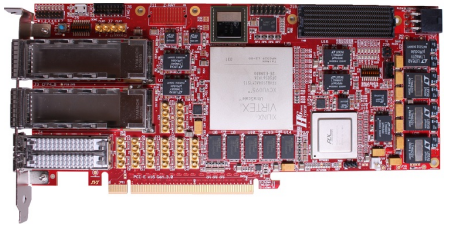
PCIe (FMC) carrier cards are already the industry standard for similar applications.



Zynq board
ARM cortex integration



Kintex 7 eval
PCIe 2.0 x4, $1600



HTG-828
PCIe 3.0 x16

Stratix V dev kit.
PCIe 3.0 x16, dual FPGA, $15000

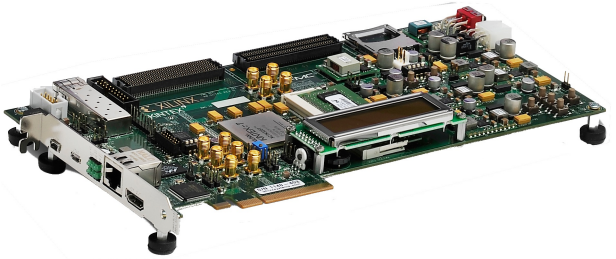
Cern FMC carrier
Open source design

Vadatech VPX514
VPX (VITA-46) card
(not strictly pure PCIe)

→ PCIe is sufficiently easy to implement,
Provides µs latency, at almost arbitrary channel counts.

PCIe (FMC) carrier cards are already the industry standard for similar applications.

Zynq board
Arm cortex integration



Kintex 7 eval
PCIe 2.0 x4, $1600

HTG-828
PCIe 3.0 x16

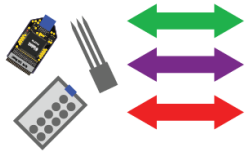
Stratix V dev kit.
PCIe 3.0 x16, dual FPGA, $15000

Cern FMC carrier
Open source design

Vadatech VPX514
VPX (VITA-46) card
(not strictly pure PCIe)

Design choices for PCIe systems

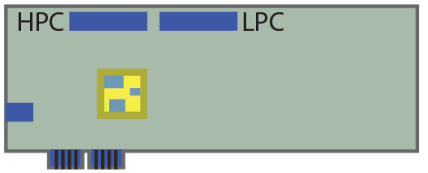
Data sources



Diverse data sources
Need different
hardware interfaces
and firmware

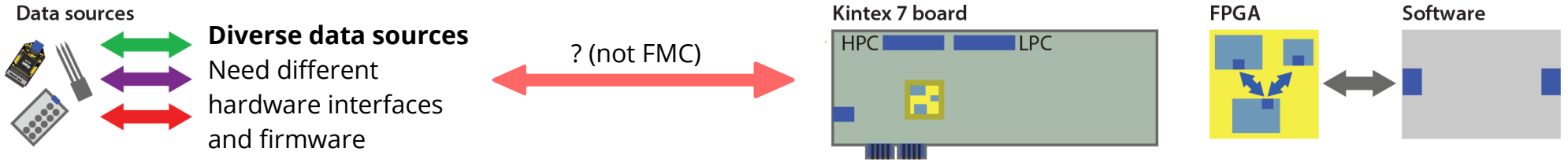
? (not FMC)

Kintex 7 board



FPGA eval board
Provides PCIe interface
Is future-proof

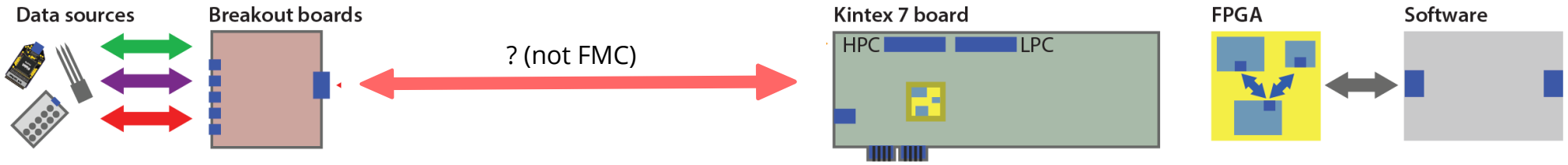
Design choices for PCIe systems



FPGA eval board
Provides PCIe interface
Is future-proof

Firmware modules
Different probes will need different firmwares, but API interface can stay the same.

Design choices for PCIe systems

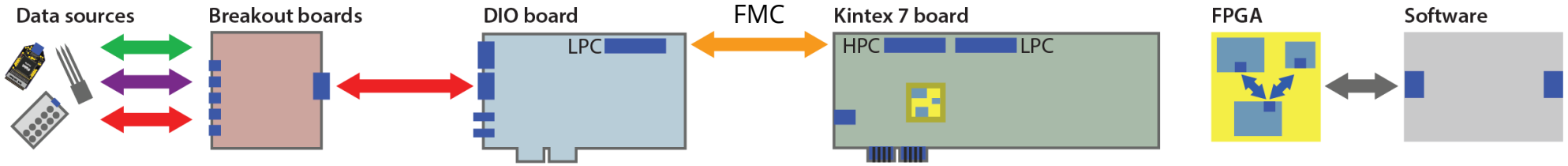


Breakout boards
We will not want to connect probes directly to the PC, so some form of breakout boards will be needed.

FPGA eval board
Provides PCIe interface
Is future-proof

Firmware modules
Different probes will need different firmwares, but API interface can stay the same.

Design choices for PCIe systems



Breakout boards
We will not want to connect probes directly to the PC, so some form of breakout boards will be needed.

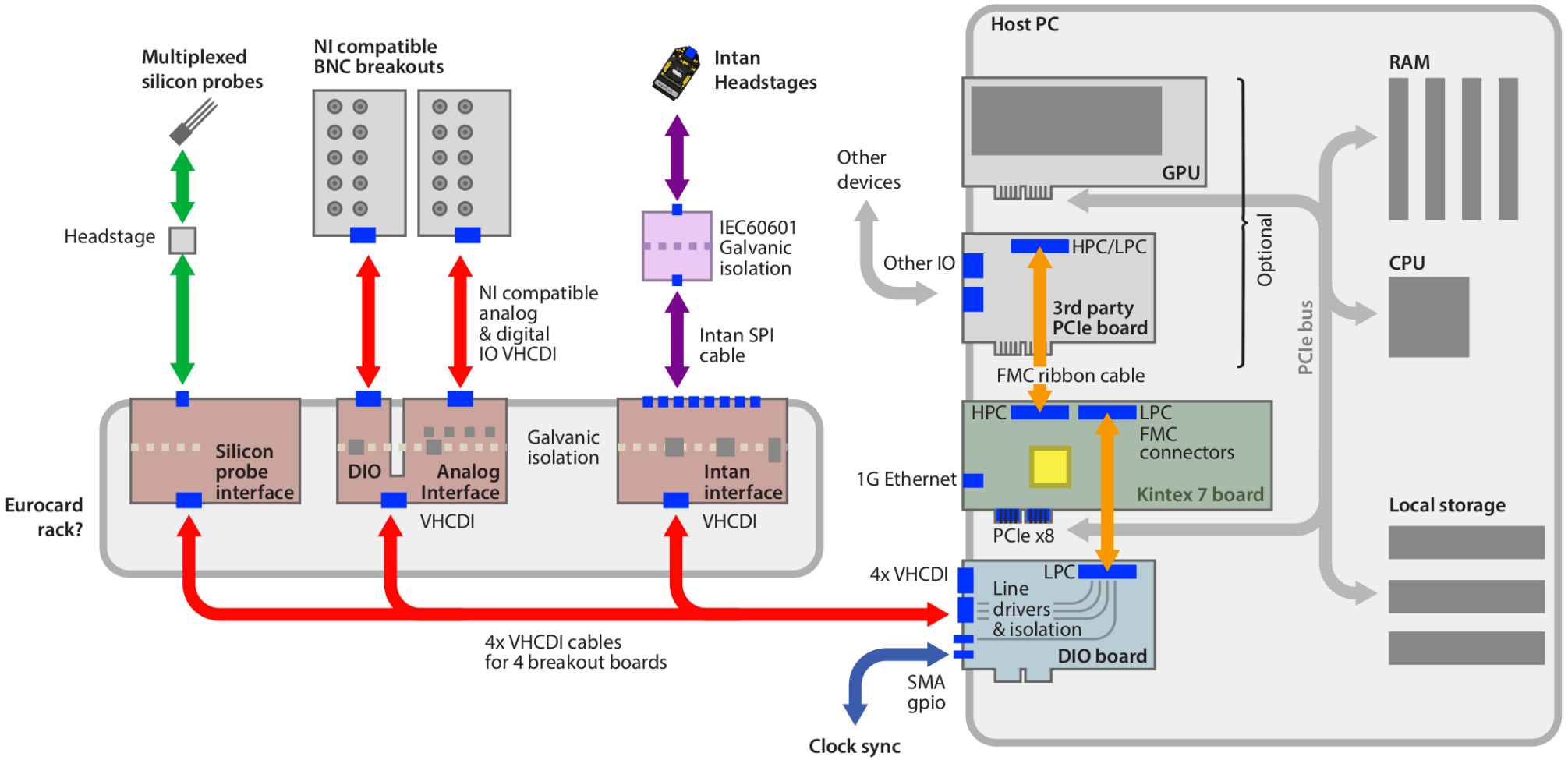
Not always needed?

DIO board
Intermediate board between FPGA board and breakout boards. FMC is great but not useful for routing outside PC case.

FPGA eval board
Provides PCIe interface
Is future-proof

Firmware modules
Different probes will need different firmwares, but API interface can stay the same.

Design choices for PCIe systems

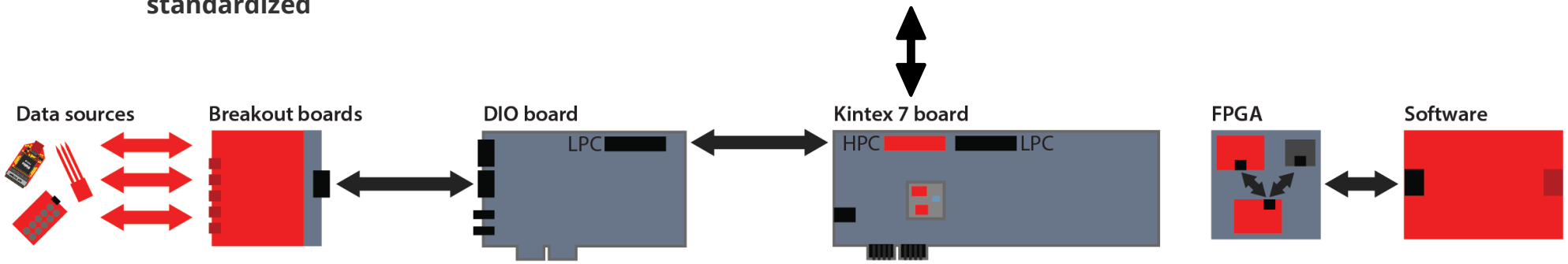


What decisions result from this now?

Design choices for PCIe systems

custom
standardized

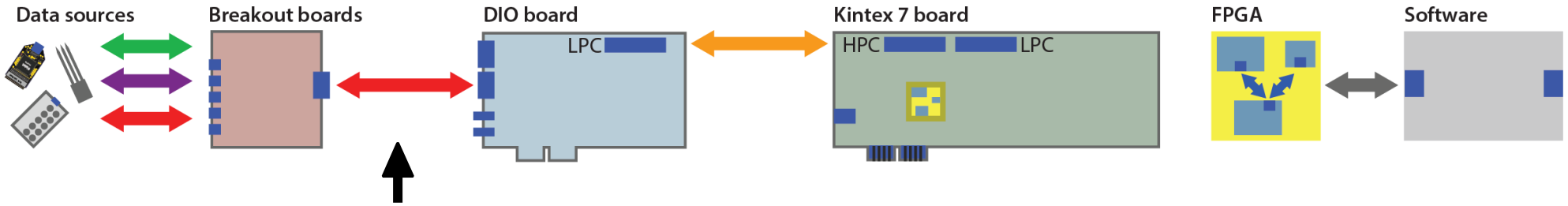
FMC connector
To non-standard
board (microscopes etc.)



Breakout boards
Implement different
HW interfaces.

Firmware modules
Different probes will
need different
firmwares, but API
interface can stay the
same.

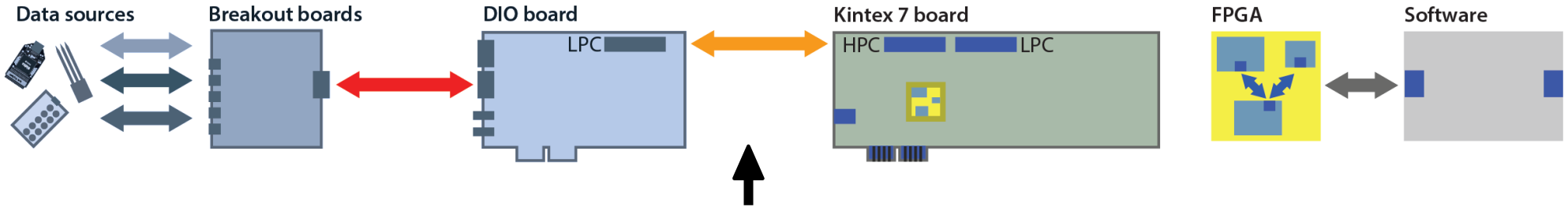
Design choices for PCIe systems



Breakout boards need to connect to PC

Which connectors/cables?

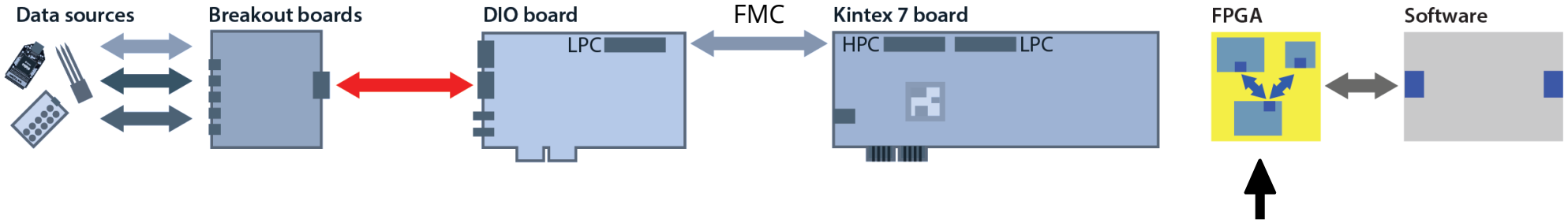
Design choices for PCIe systems



FMC connectors
(ribbon cables for PCIe format cards)

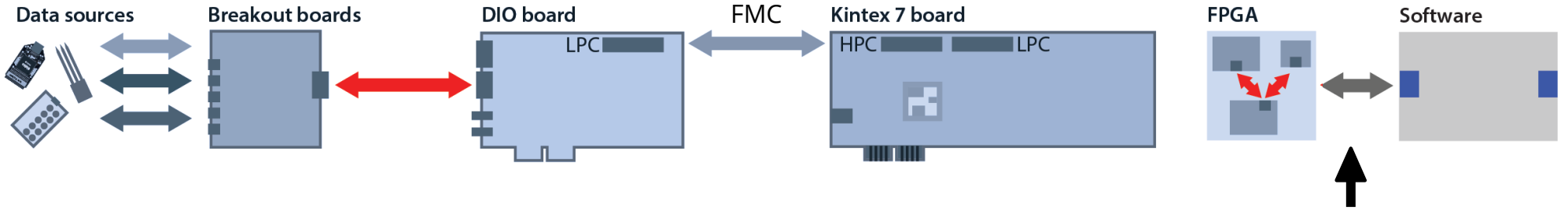
Vita 57 FMC daughter card mezzanine connectors - already a strong standard
Same as used at CERN, all kinds of Xilinx, Altera, etc. carriers and modules, Neuroseeker(?), NeuroPixel

Design choices for PCIe systems



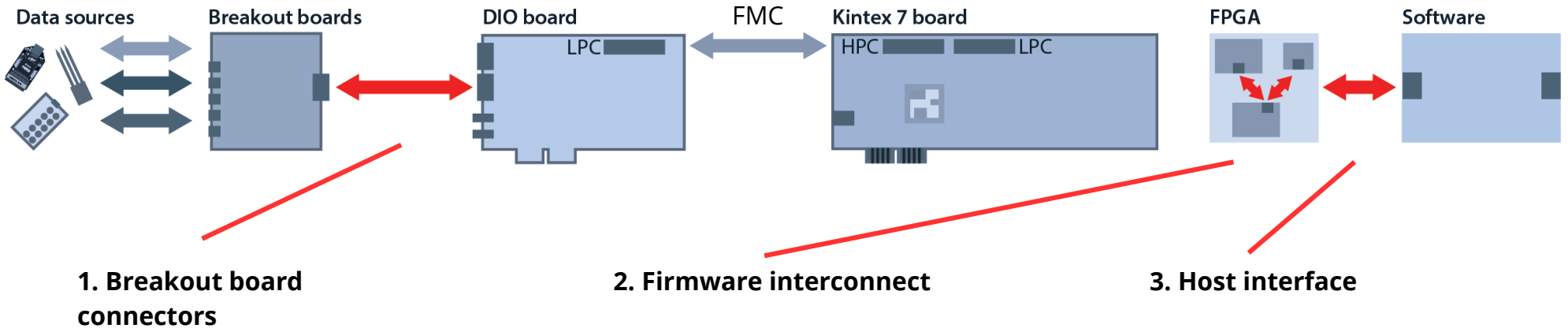
Firmware interconnect
Wishbone
IP cores?

Design choices for PCIe systems

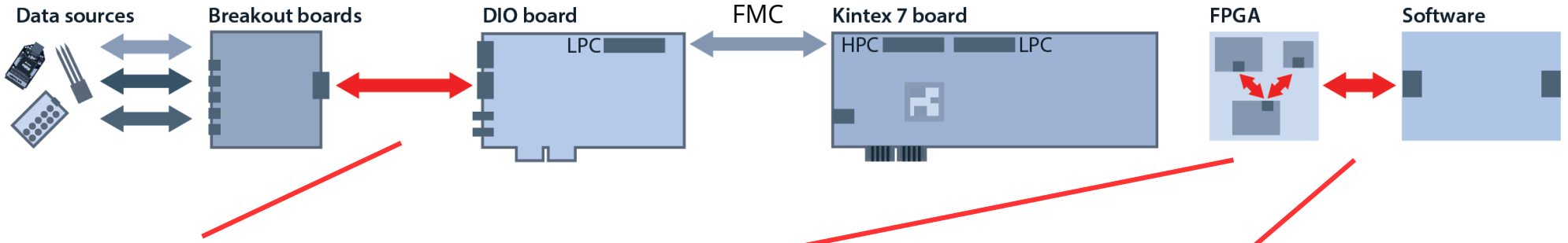


Host interface
PCIe (which method?)
optional Ethernet ?

Design choices for PCIe systems



Design choices for PCIe systems

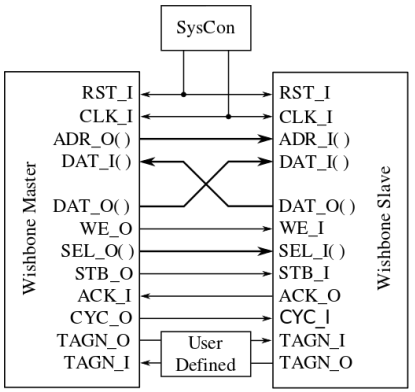


1. Breakout board connectors

Just break out raw FMC pins
VHCDI / SCSI cables

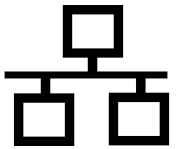
2. Firmware interconnect

Wishbone based specification?

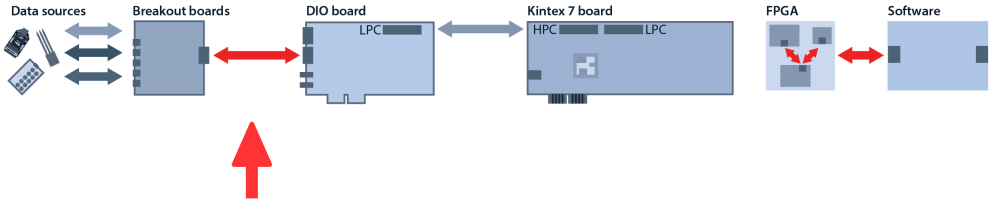


3. Host interface

Extendable PCIe based DMA
Plus ethernet for broadcast



Breakout board connectors



Breakout board connectors

Just break out raw FMC pins
to VHCDI / SCSI cables

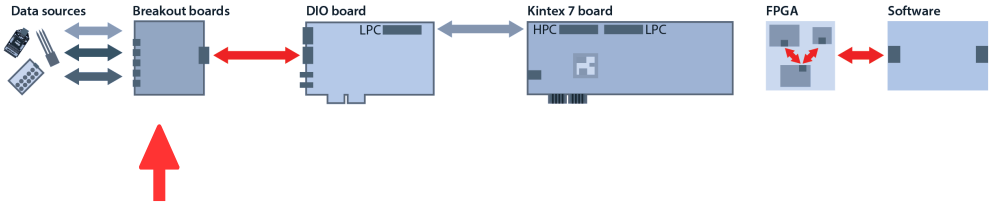
Space for 4 conectors
& ~3 SMAs for clock etc.

68 pins
Twisted pairs
$25 - $400 options (shielded or not etc.)
~28GA → ~600mA/wire

Specify minimum number of pins
- VCC (3.3V & 12V) /GND
- i2c bus for eeprom device id

Use LPC or HPC? Likely HPC since we want a decent pin count per connector.

Breakout board format

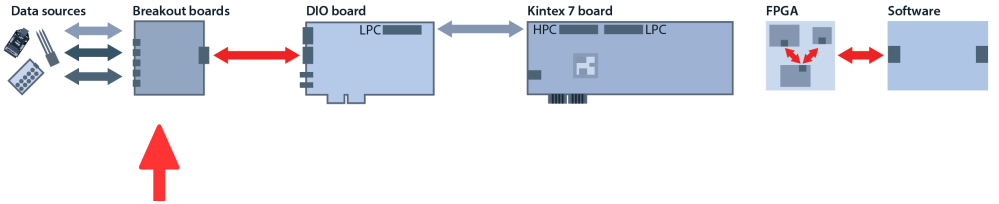


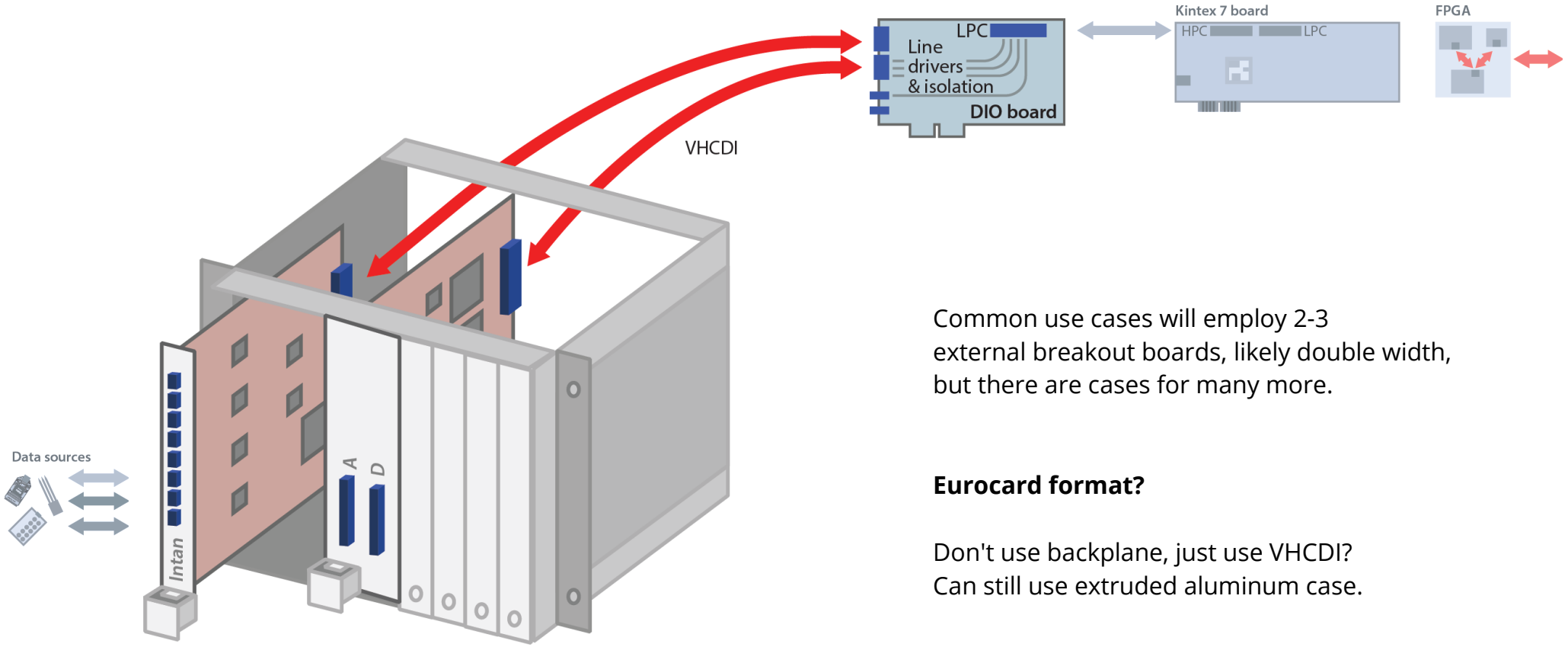
Common use cases will employ 2-3 external breakout boards, likely double width, but there are cases for many more.

Eurocard format?

Don't use backplane, just use VHCDI?
Can still use extruded aluminum case.

Breakout board format



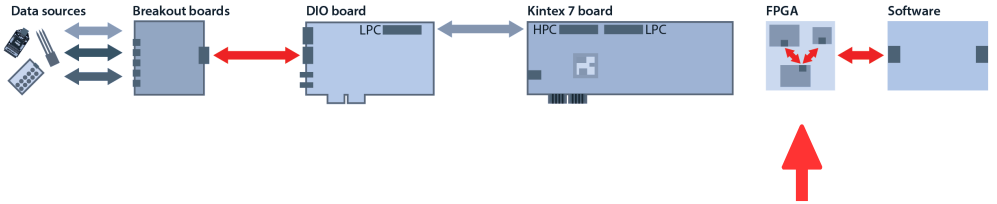


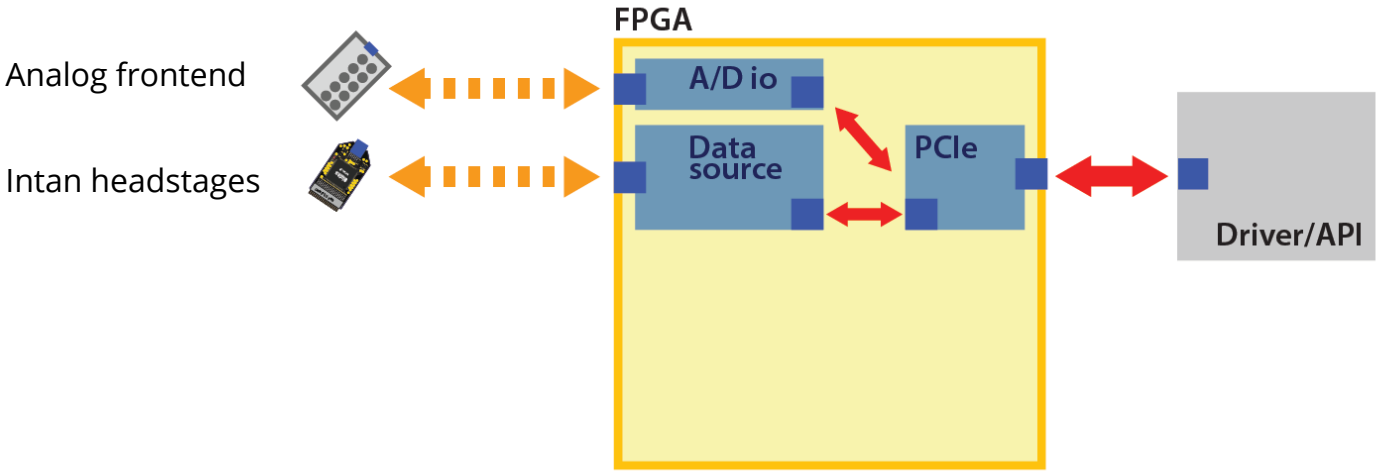
Common use cases will employ 2-3 external breakout boards, likely double width, but there are cases for many more.

Eurocard format?

Don't use backplane, just use VHCDI?
Can still use extruded aluminum case.

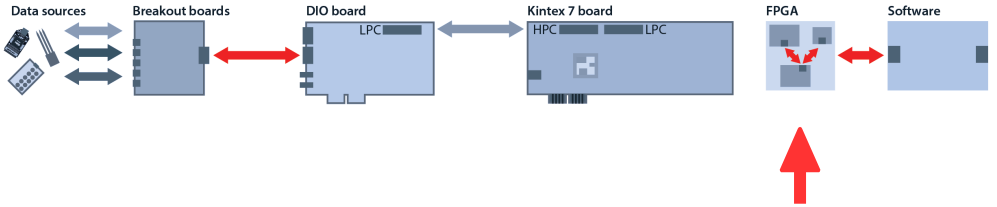
IP core interfaces

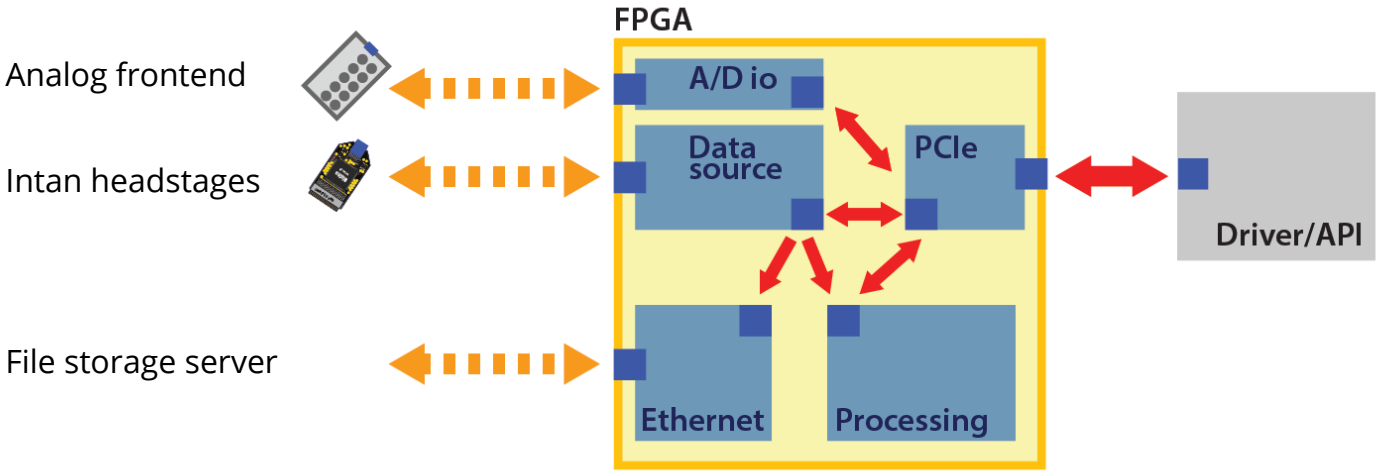




Core application:
Data source Firmware
Aux input (A/D io)
PCIe interface

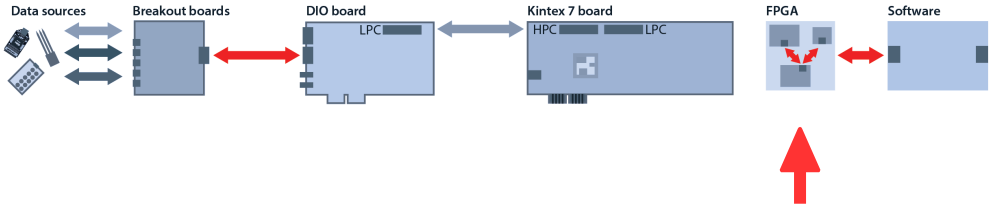
IP core interfaces

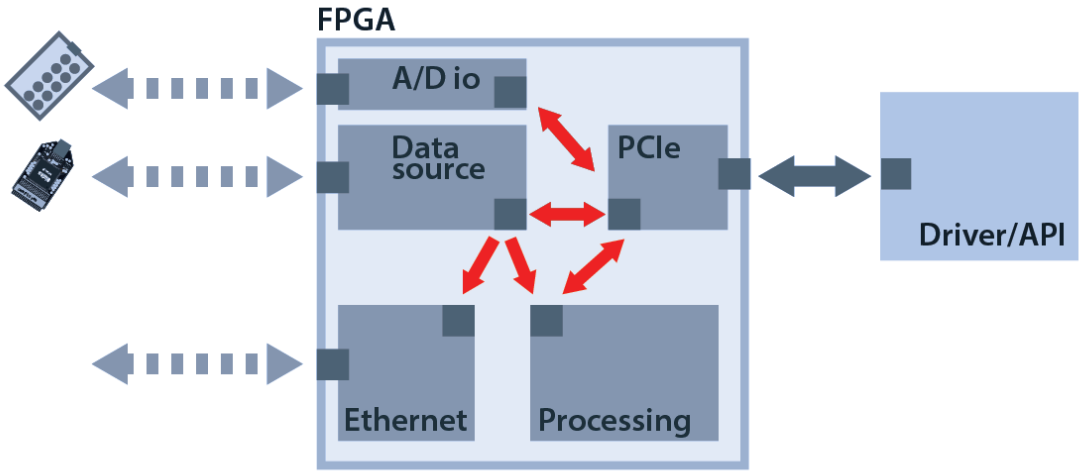




Extended application:
Data source Firmware
Aux input (A/D io)
PCIe interface
FPGA processing
Ethernet streaming

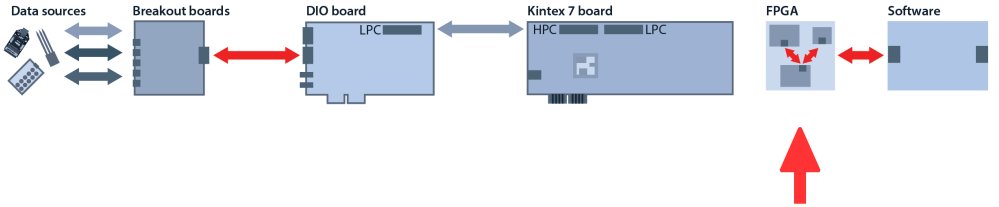
IP core interfaces

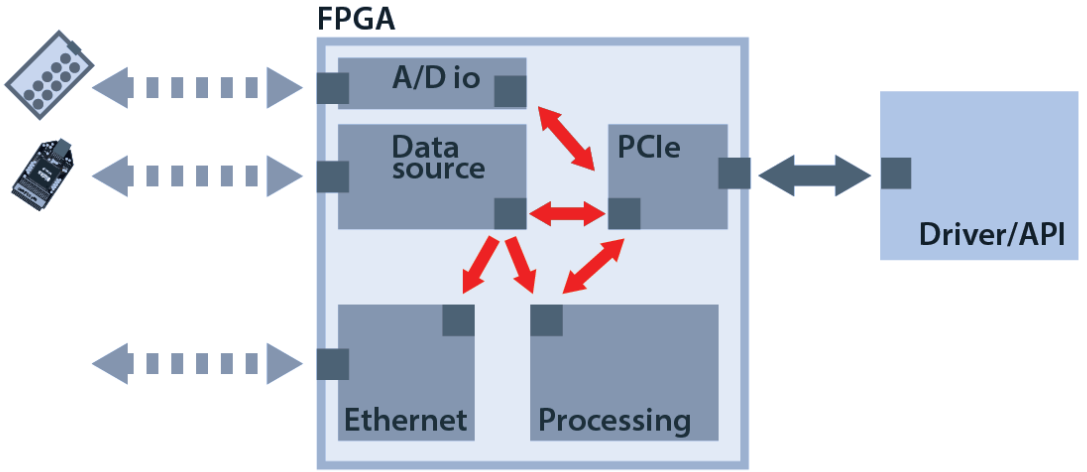


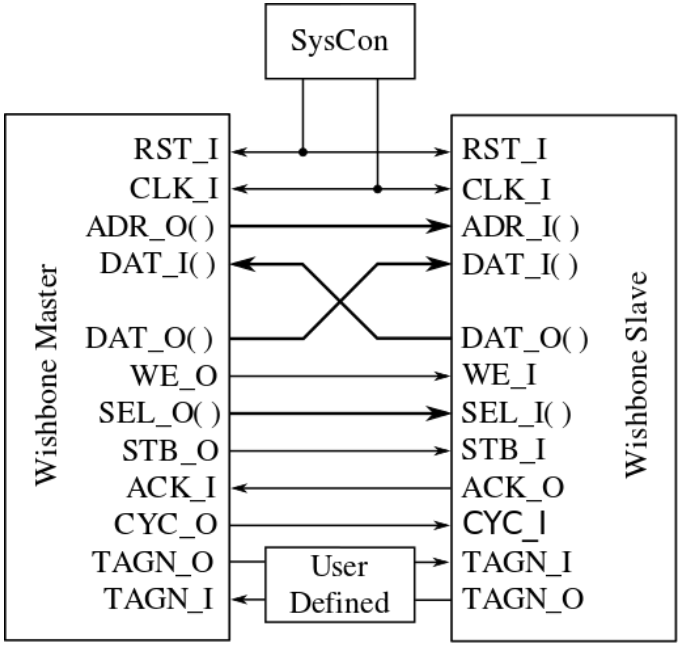


Interface between modules
Wishbone, which specifications?

IP core interfaces



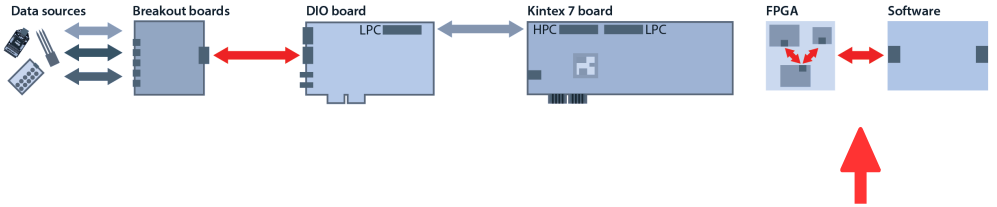


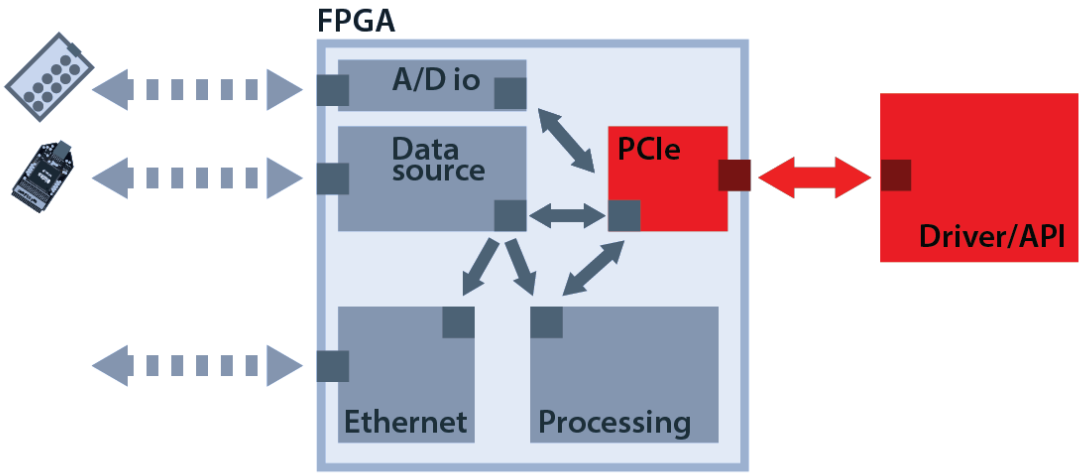


Interface between modules
Wishbone interconnect?

Data format and config register access is specified very similarly to the API.

IP core interfaces





Host PC interface
Firmware side & Drivers

Currently use xillybus
Move to something with non-restrictive driver that can be used commercially.

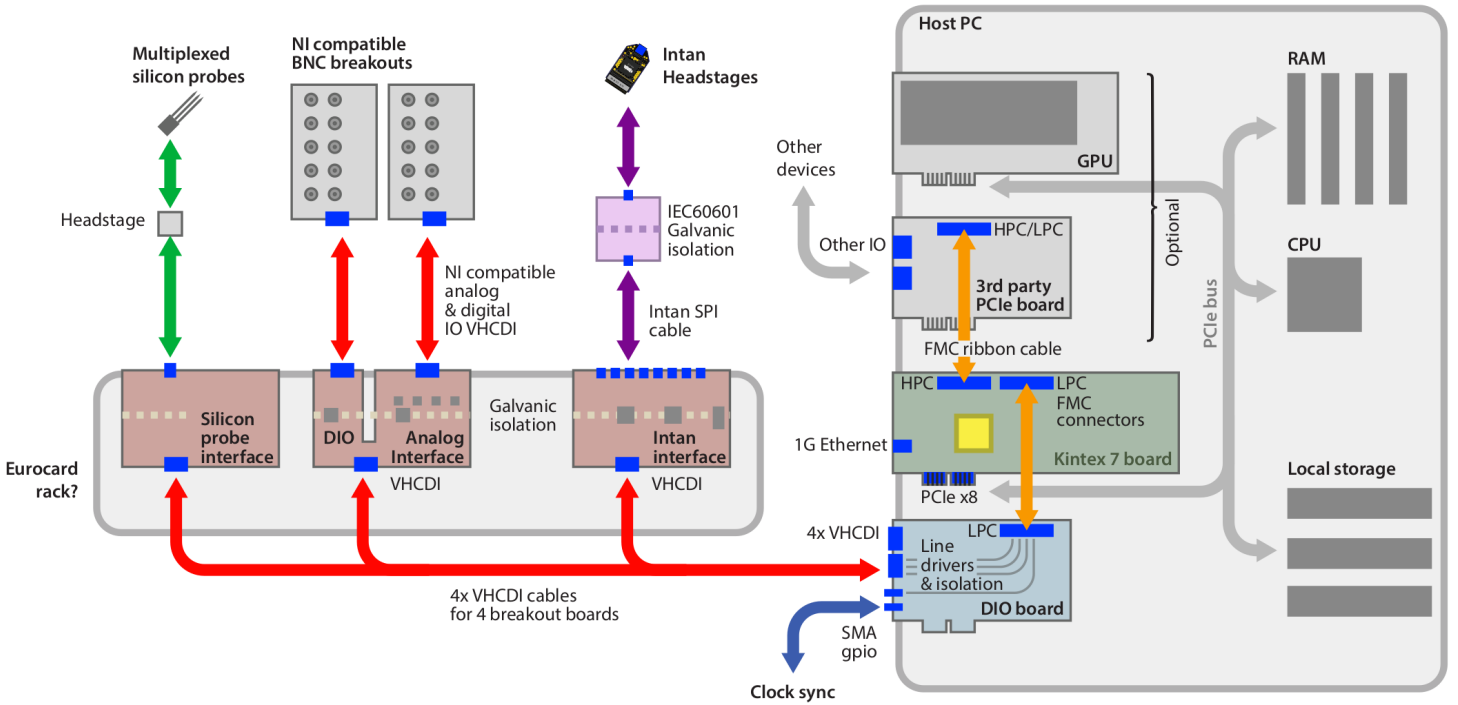
Conclusion

PCIe DMA interface is
- sufficiently developer friendly
- highest performance
- most future-proof

Proposed system can mostly use generic industry standards

Maintains compatibility with almost all existing standards

Minimizes amount of work required for new data sources to close to theoretical minimum.



Interface specifications:
1. Breakout board standards
2. IP interconnect
3. PCIe ↔ PC interconnect