ENCLUSTRA
FPGA SOLUTIONS

Efficient SoC-Based DSP up to GS/s

Oliver Bründler
oliver.bruendler@enclustra.com
CERN 02-Oct-2023

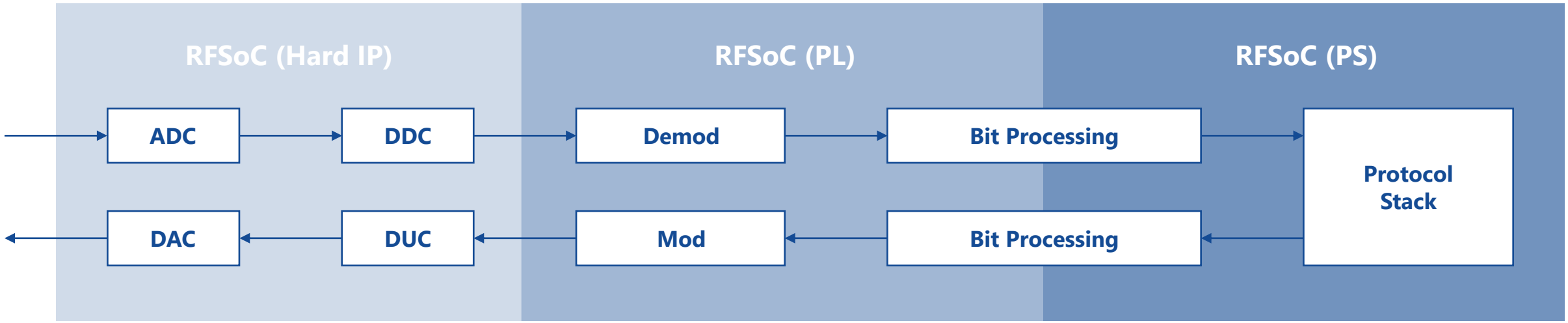
Agenda

- The Evolution of DSP
- Traditional DSP
- Parallel DSP
- Bit-True Development

The Evolution of DSP

The Evolution of DSP Systems

up to 1990s	Implement some part of the DSP chain in software
2000s	Move intensive processing into an FPGA
2010s	Move the CPU into the same package as the FPGA
2020s	Move the ADCs/DACs into the same package as CPU and FPGA

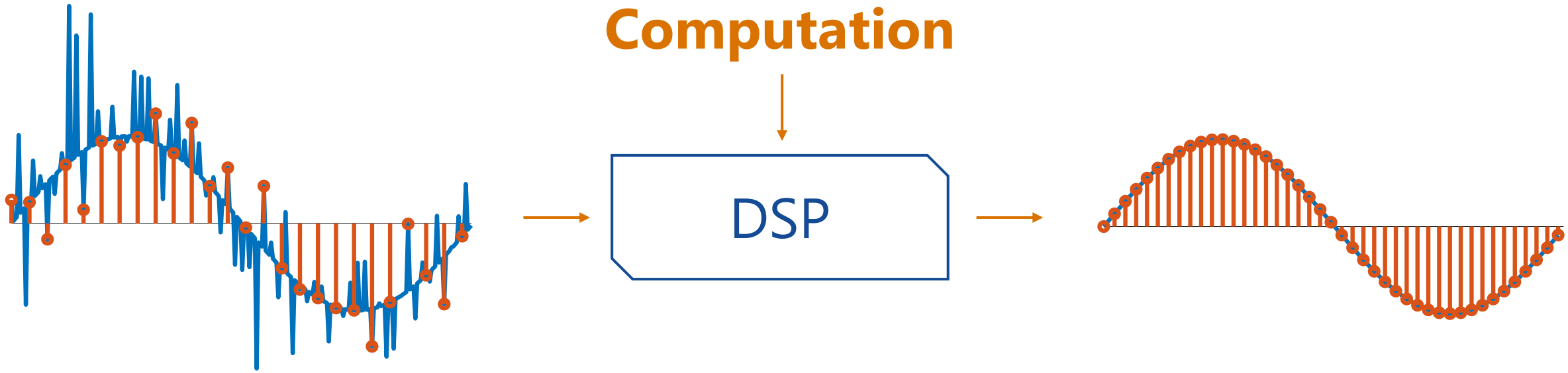


Traditional DSP

Reminder: DSP in a Nutshell

Turn bad signals into good ones

- This requires processing effort (per sample)
- Higher sample rate ⇒ higher effort rate ⇒ more FPGA resources



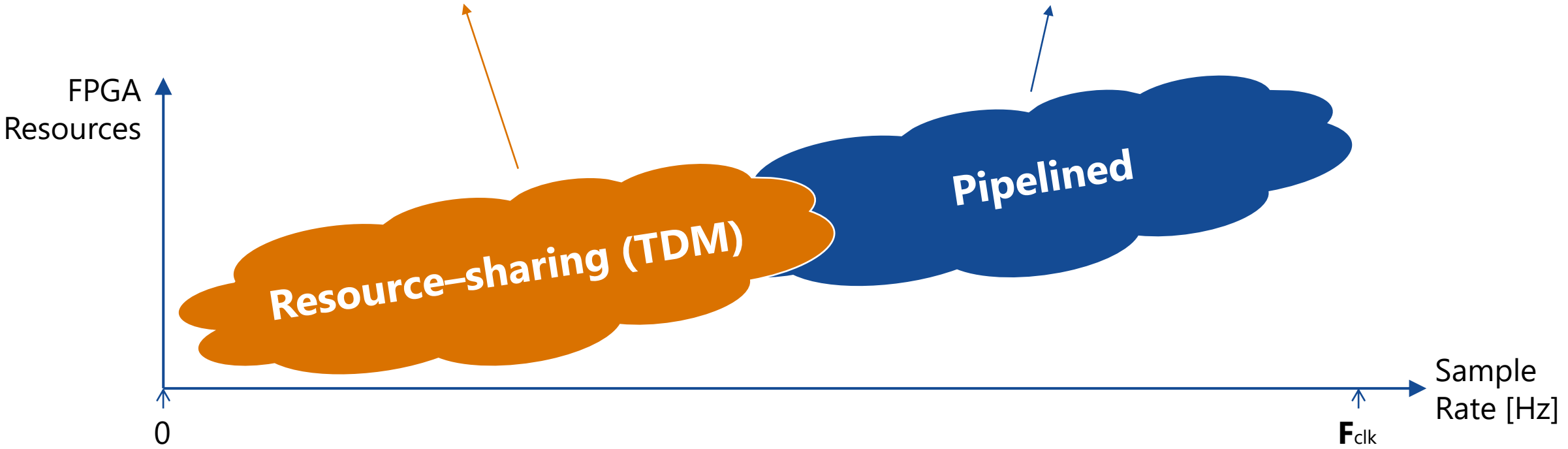
Traditional DSP Trade-offs

Resource-sharing designs

- Lower resource consumption ☺
- Lower processing power ☹

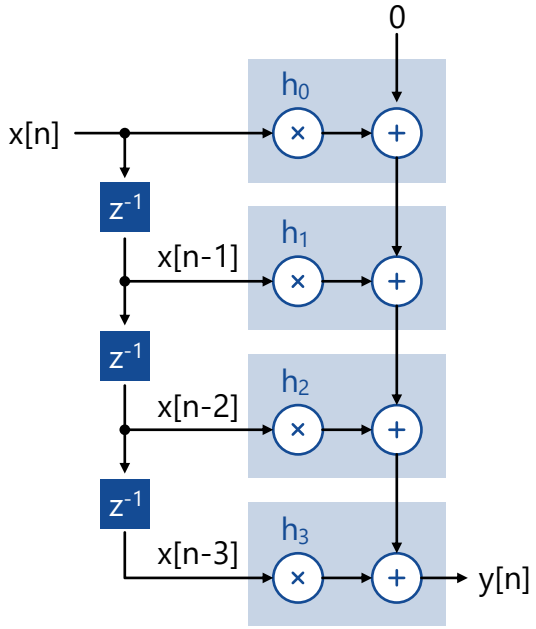
Pipelined designs

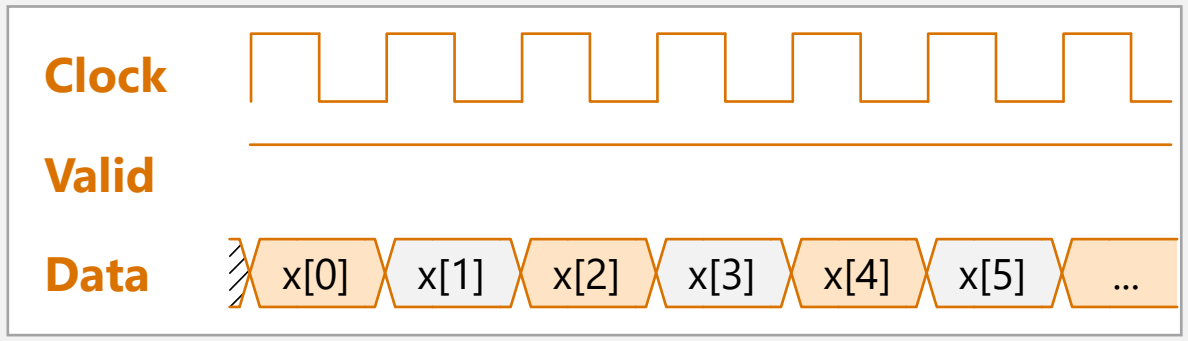
- Higher resource consumption ☹
- Higher processing power ☺



Traditional DSP Example: Pipelined FIR Filter

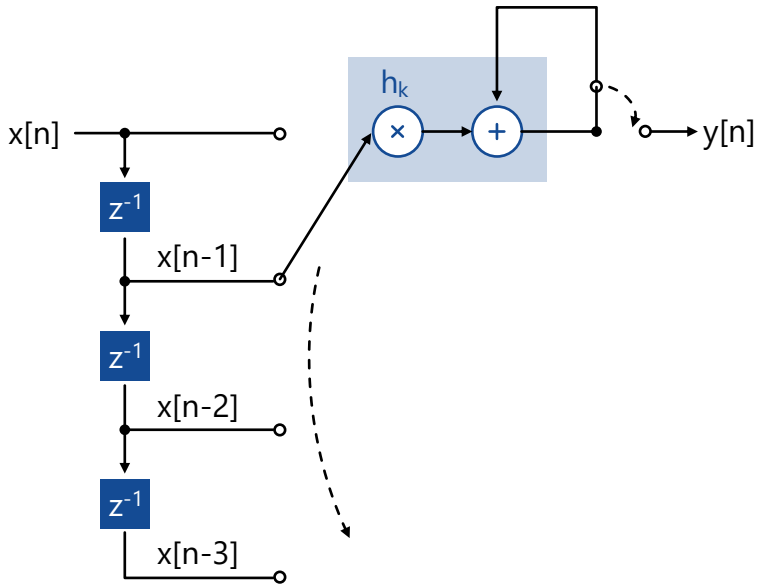
Ready for valid input every clock cycle

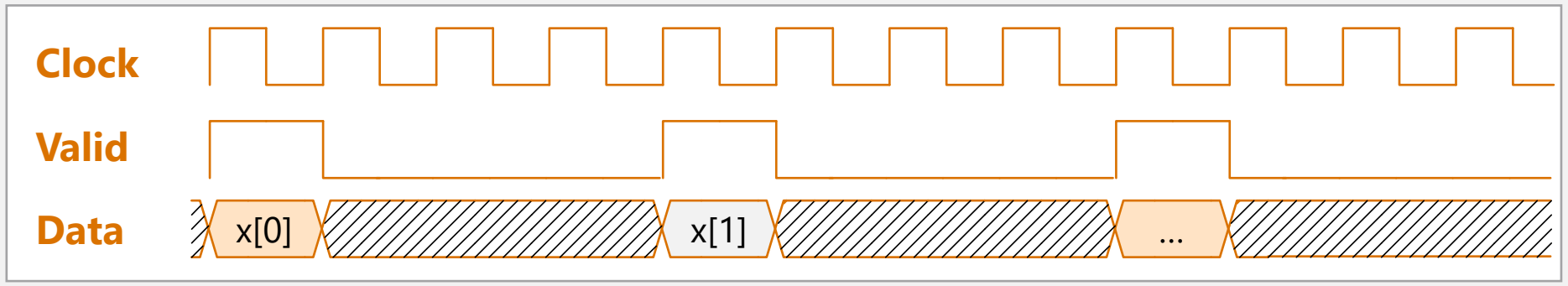




Traditional DSP Example: Resource-Sharing FIR Filter

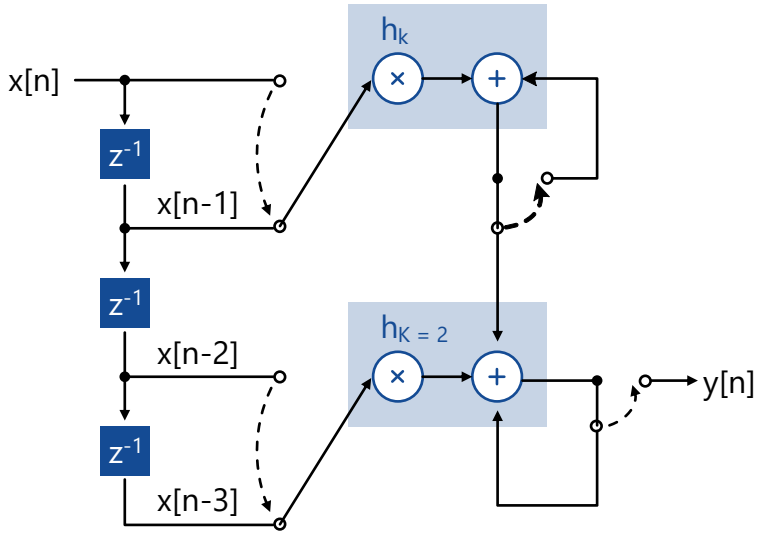
Ready for valid input every
4 clock cycles

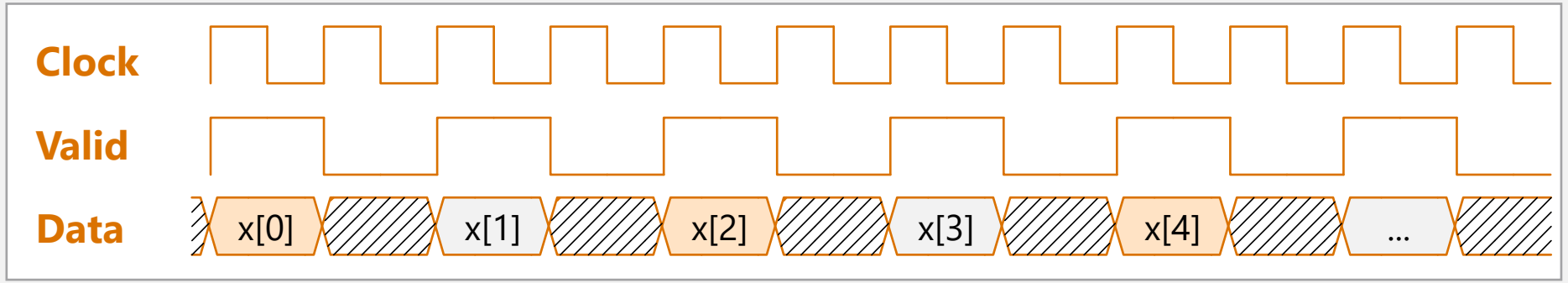




Traditional DSP Example: Semi-Parallel FIR Filter

Ready for valid input every
2 clock cycles



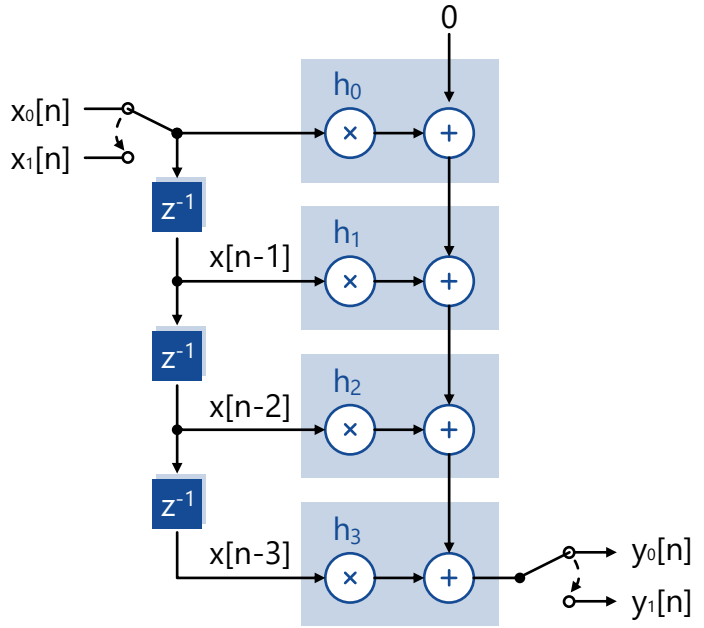


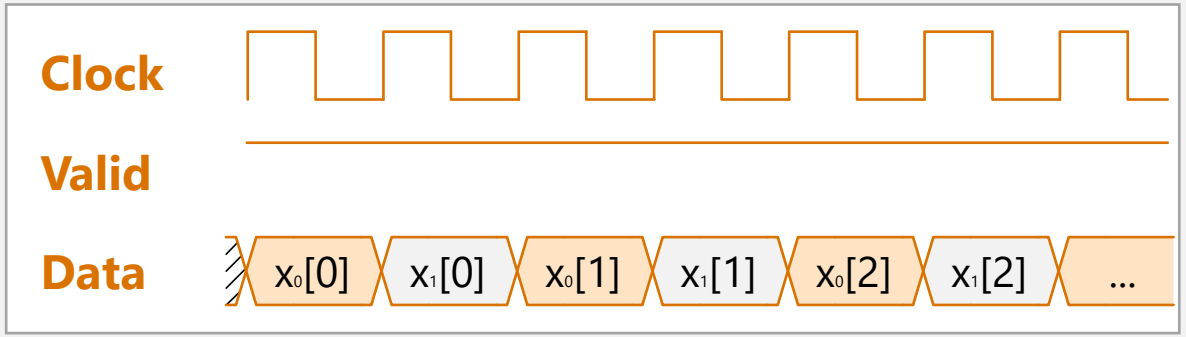
Exactly the same mathematical function requires a totally different implementation at a different sample rate.

Traditional DSP Example: TDM Pipelined FIR Filter

TDM = Time Division Multiplexed

Ready for valid input every
2 clock cycles

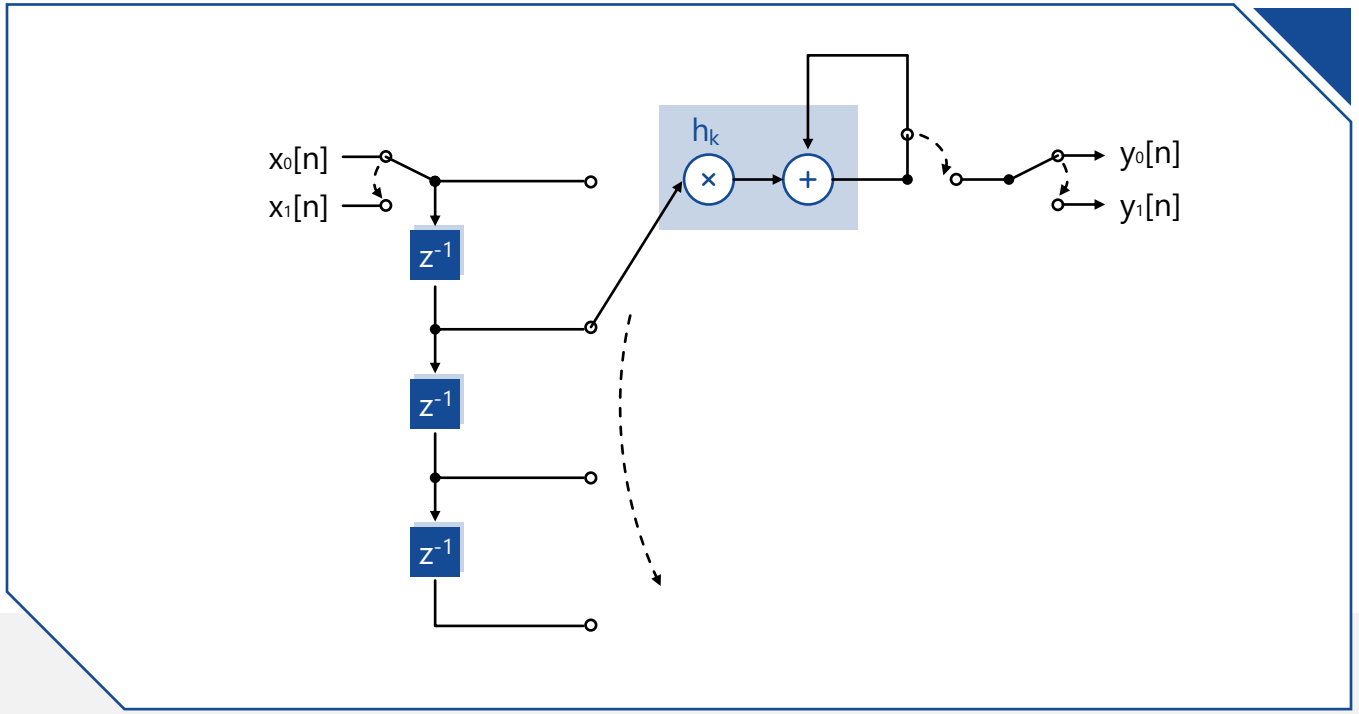


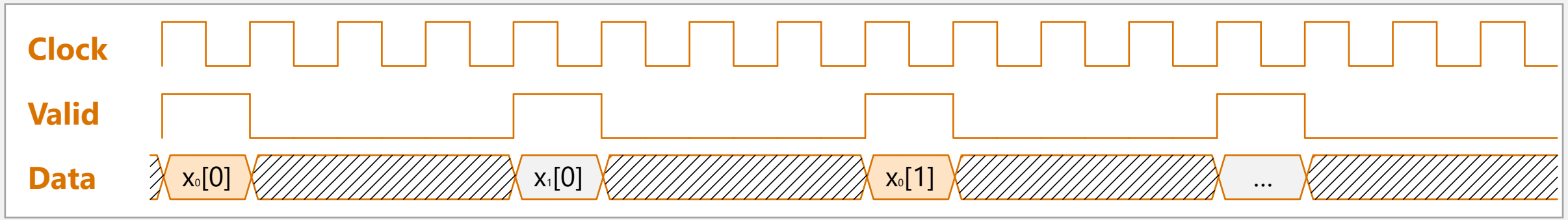


Traditional DSP Example: TDM Resource-Sharing FIR

TDM = Time Division Multiplexed

Ready for valid input every
8 clock cycles





Enclustra's Universal DSP Library

Basic DSP components:

- FIR filters
- CIC filters
- Mixers
- CORDICs
- Uniform/Gaussian/arbitrary noise generators
- Function approximations ($\sqrt{x}$, $1/x$)

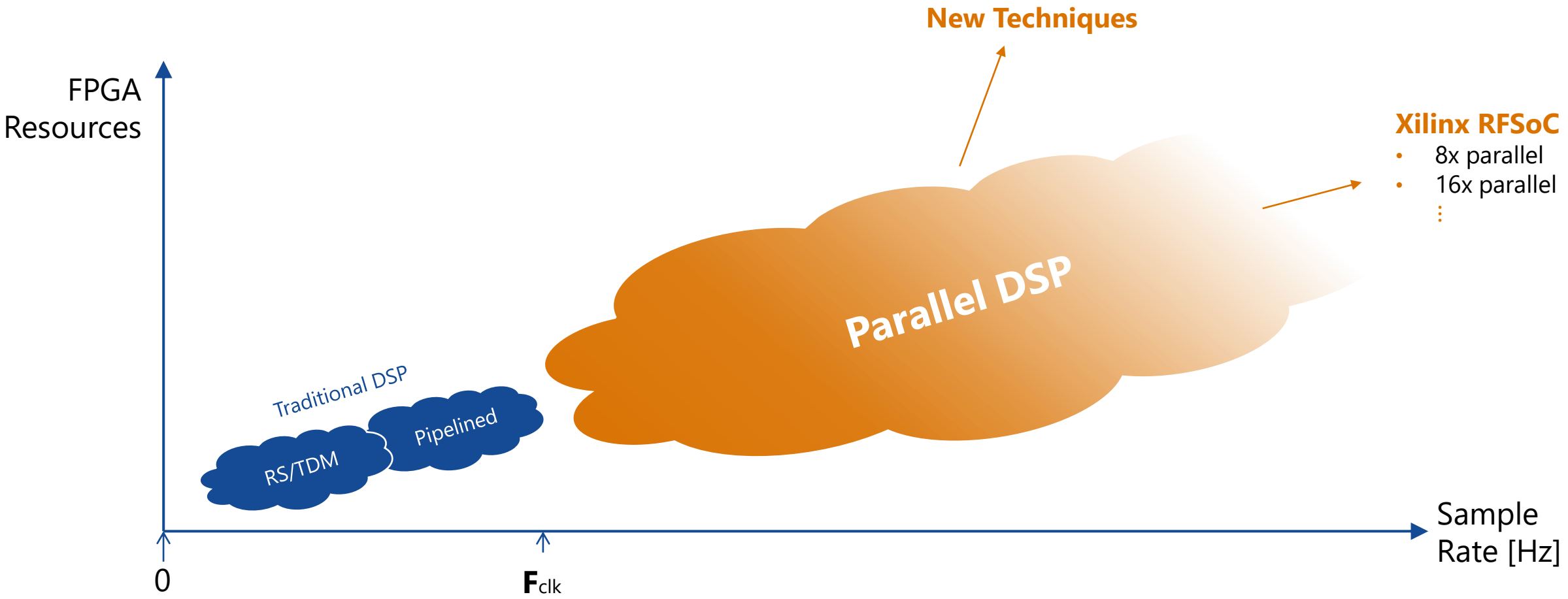
Various implementations:

- Resource-sharing and pipelined
- TDM and parallel channel handling
- Complex and real number handling
- HDL / Vivado-IPI / Python

www.enclustra.com/en/products/ip-cores/universal-dsp-library/

Parallel DSP

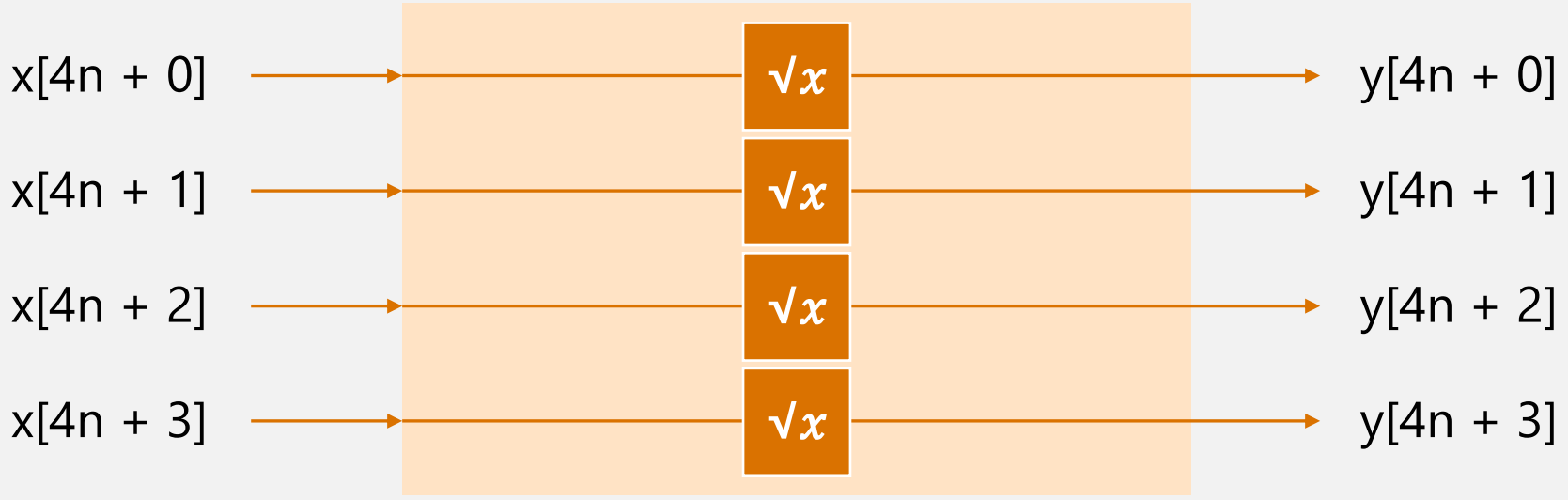
What if Sample Rate > Clock Frequency?



Parallel DSP: Memoryless Functions

Memoryless (per-sample) functions ⇒ no parallelization required

- Basic arithmetic (+, -, ×, ÷, $\sqrt{x}$, log(x))
- CORDIC (rotate, arctan(x), $\sqrt{x}$, sin(x), cos(x), …)



Parallel DSP: Functions with Memory

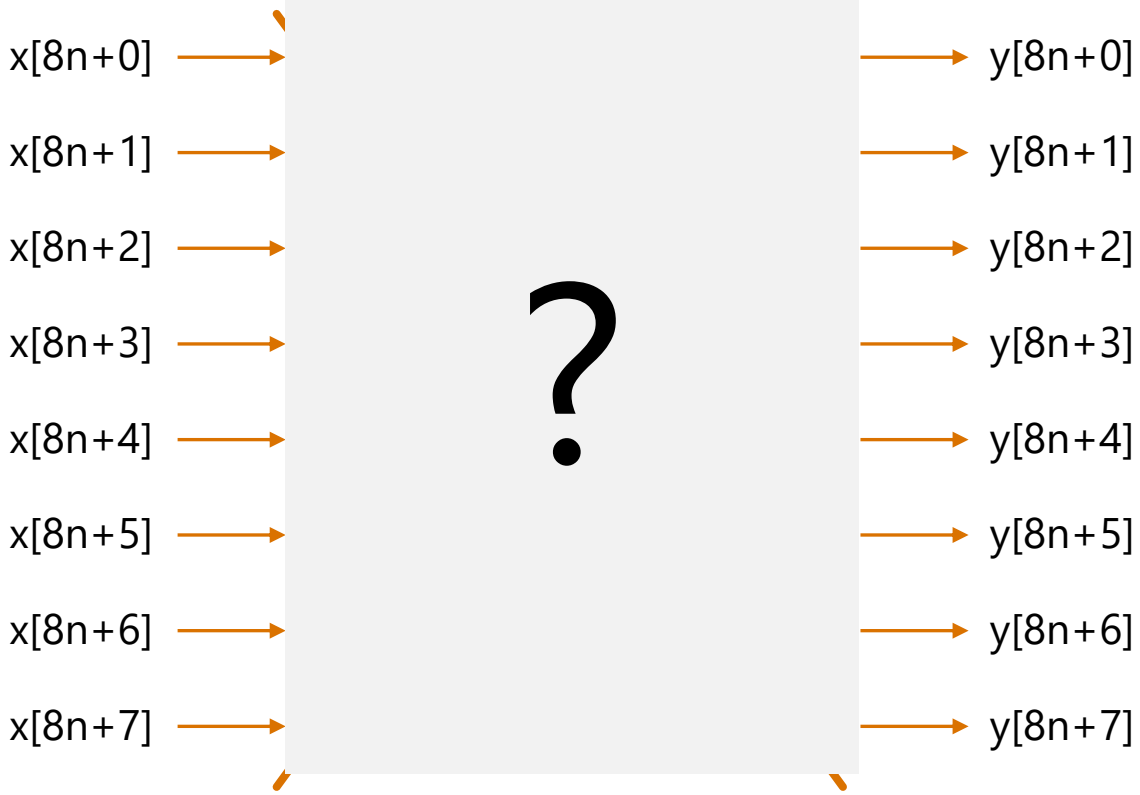
Functions with memory ⇒ algorithm must be parallelized

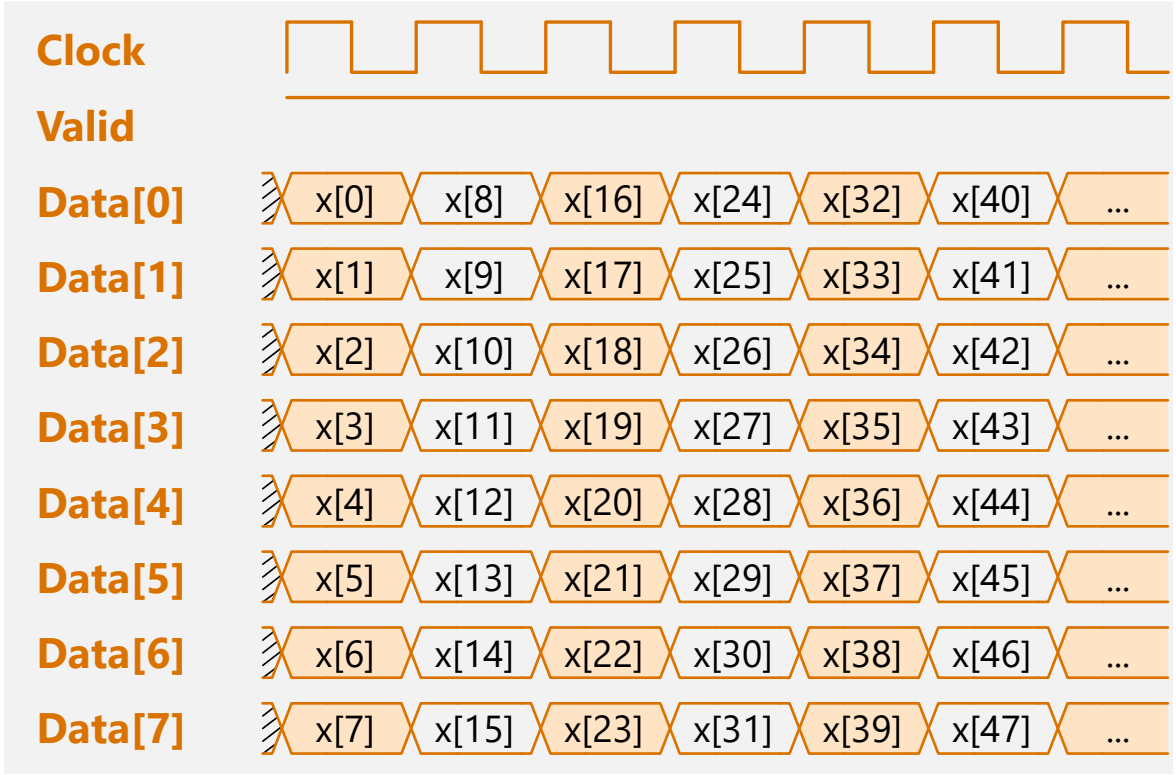
- Pseudorandom Number Generators
- Numerically Controlled Oscillators (NCOs)
- FIR Filters
- FFTs
- …

Parallel DSP Example: 8-Parallel FIR Filter

Ready for 8 valid inputs every clock cycle

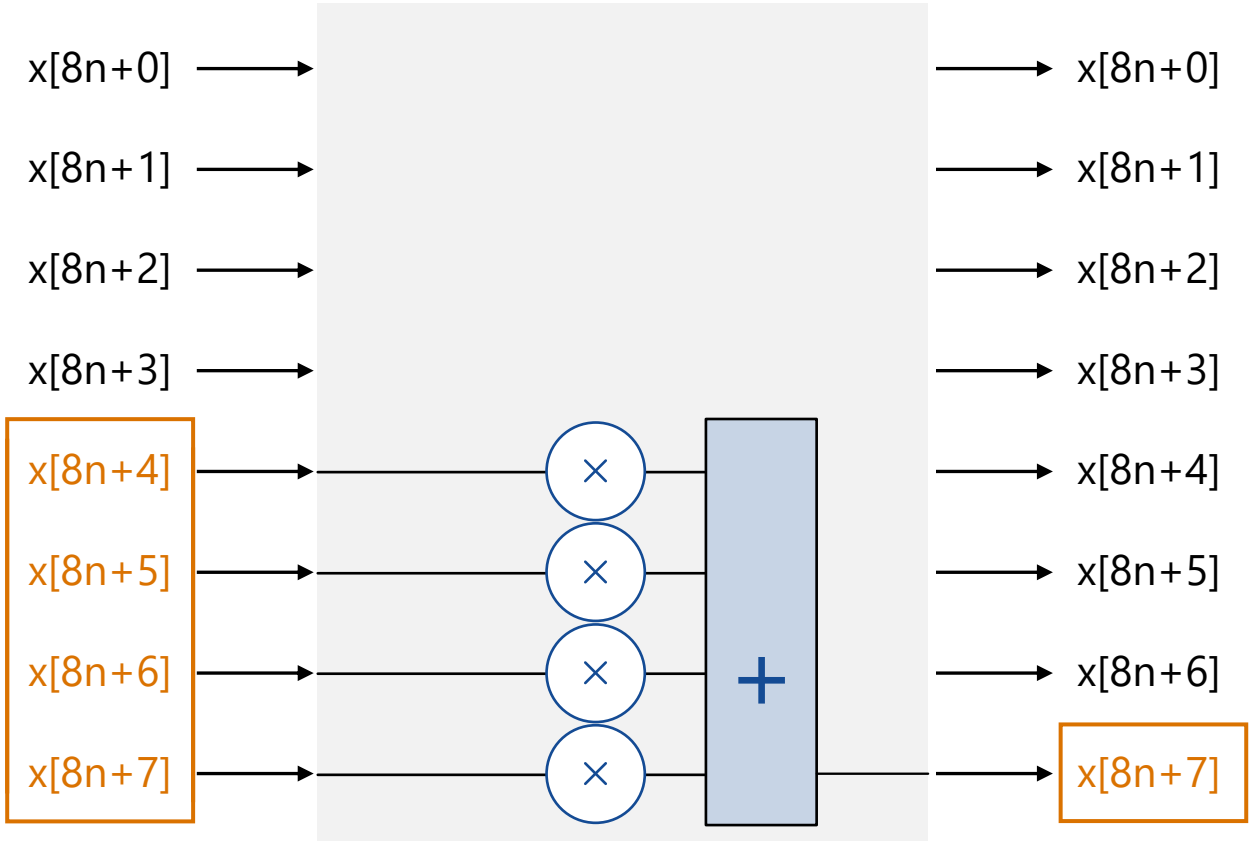
How should we arrange the 32 multipliers?



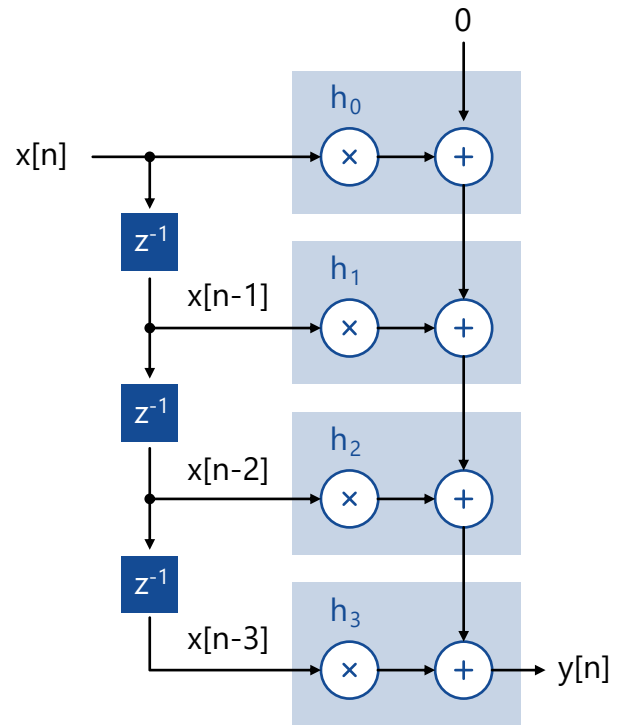


Parallel DSP Example: 8-Parallel FIR Filter

Start with the last output:

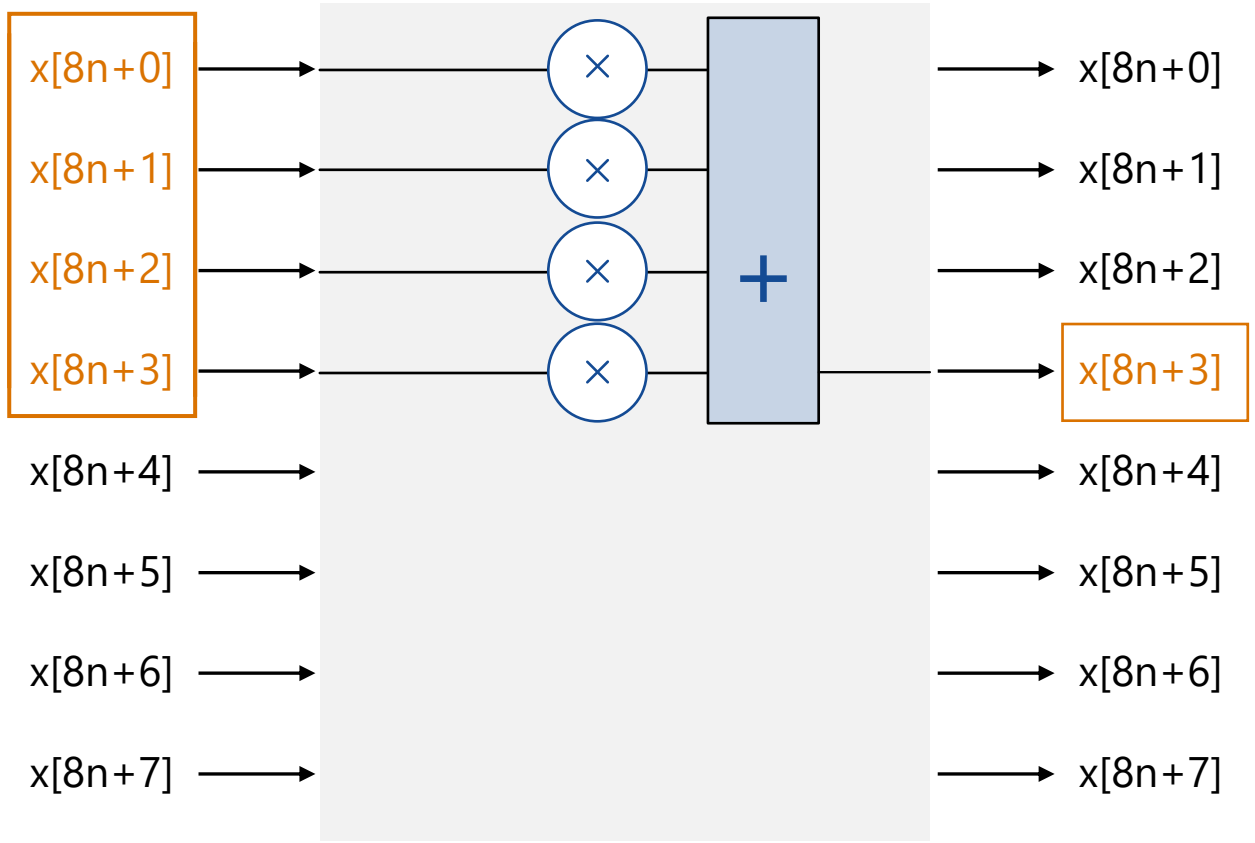


Reminder:



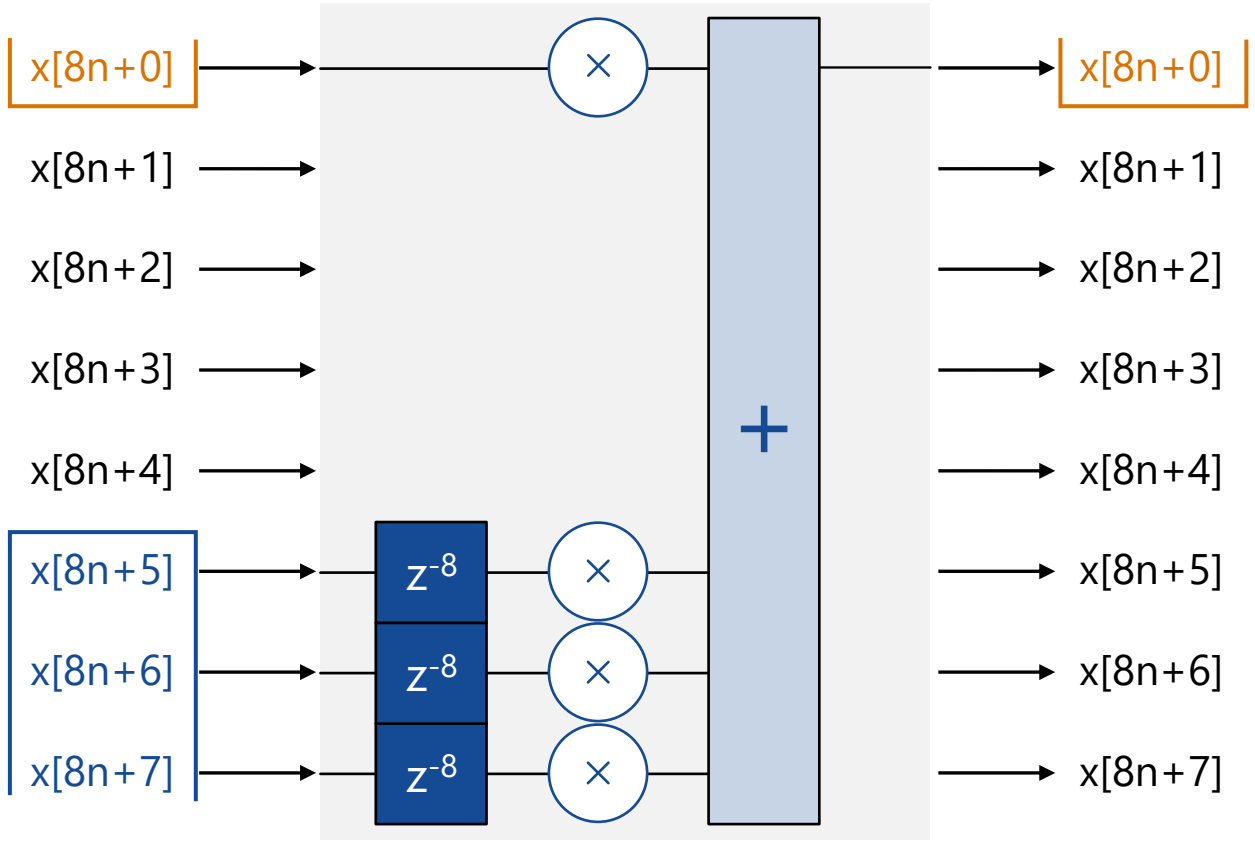
Parallel DSP Example: 8-Parallel FIR Filter

Similarly:

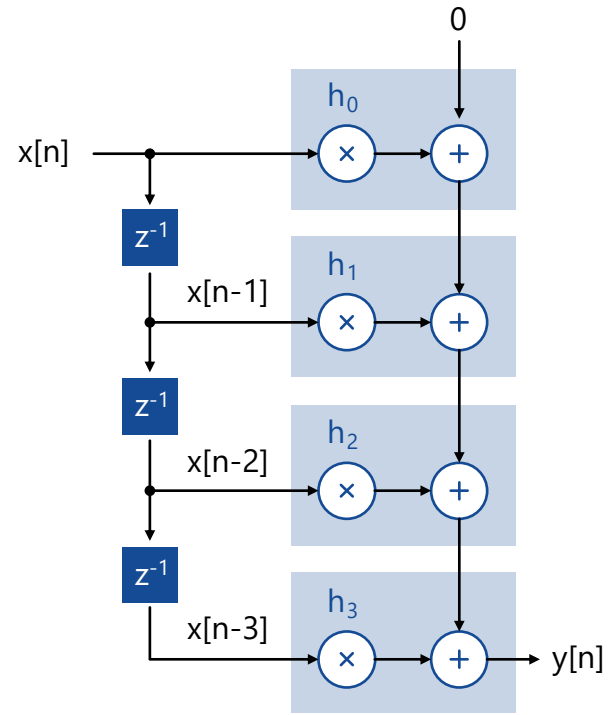


Parallel DSP Example: 8-Parallel FIR Filter

Be careful when wrapping the previous data beat:



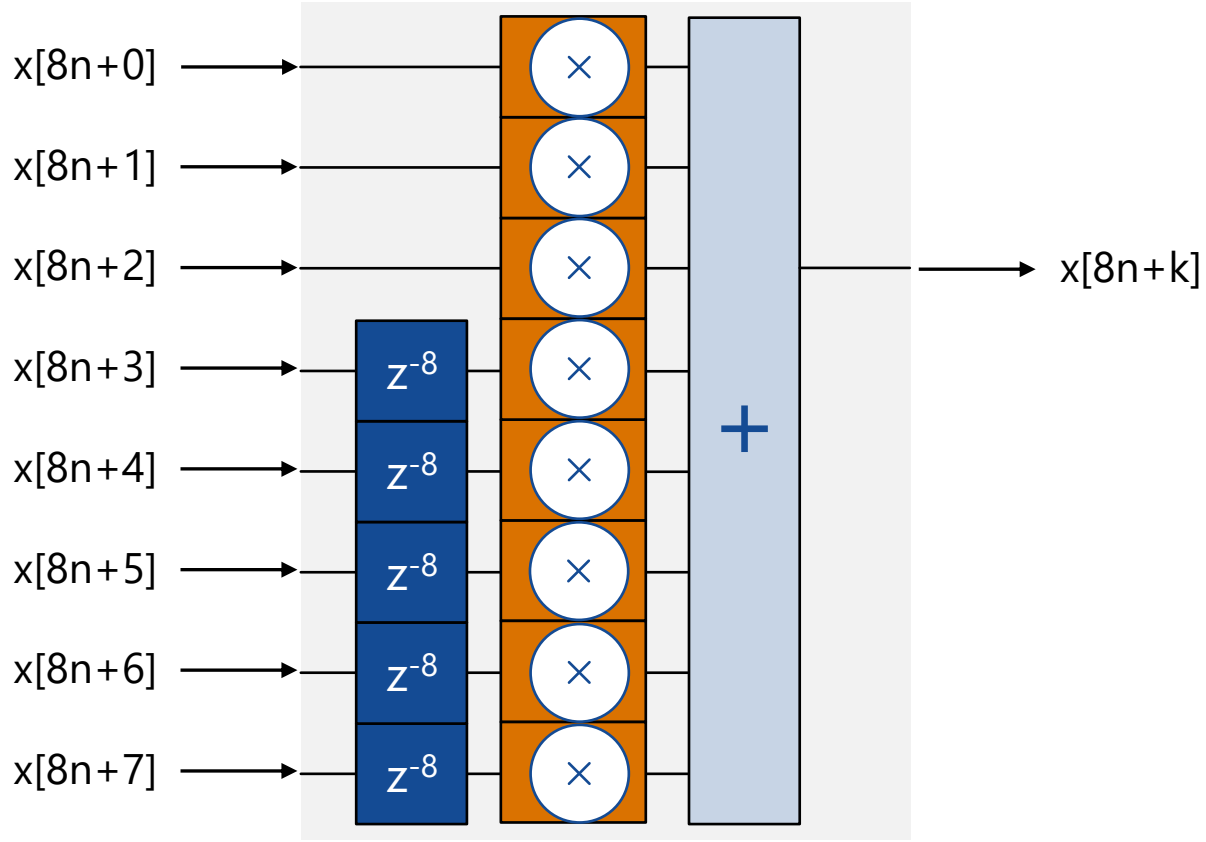
Reminder:



Parallel DSP Example: 8-Parallel FIR Filter

The filter length (N) may be larger than the parallelization factor (P)

In general, **each output** requires P=8 pipelined FIR subfilters



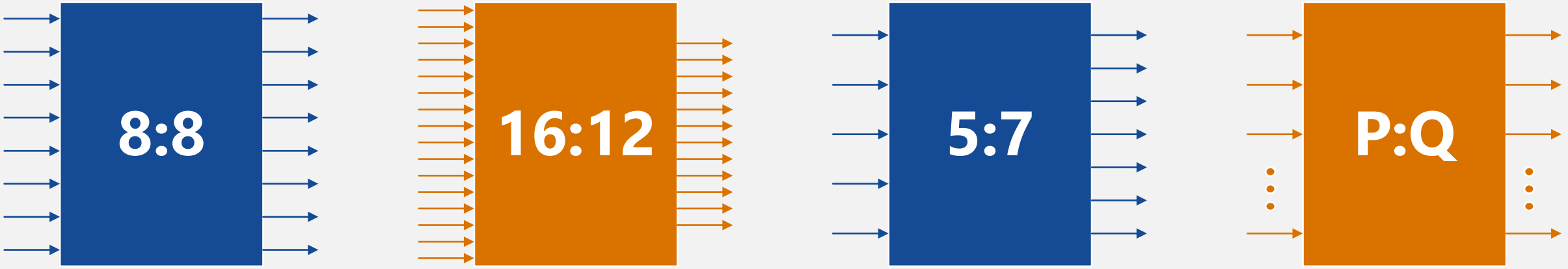
Exactly the same mathematical function requires a totally different implementation at a different sample rate.

Parallel Polyphase FIR Filters

This result can be generalized to all parallel polyphase resampling filters (P inputs → Q outputs)

The main building block is a traditional pipelined FIR filter

- Resource optimization (exploiting symmetry) requires new techniques



Bit-True Development

Why Do We Need Fixed-Point Numbers?

Fixed point operations are cheap in hardware

FPGA architectures are optimized for fixed-point

Example: Typical Multiply-Add Costs

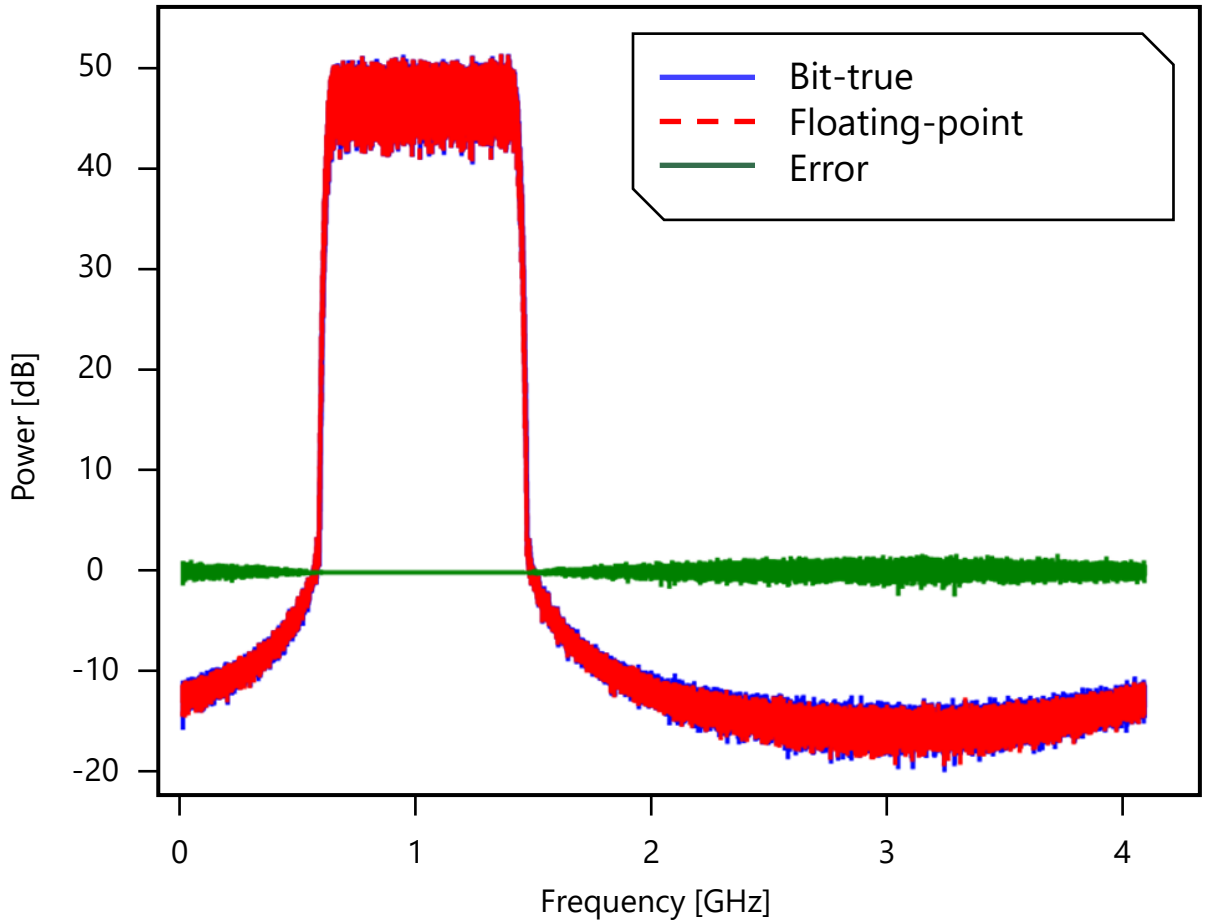
Data type	Data widths	DSPs	LUTs	Latency
Floating-point	Double-precision	10	1798	26
Floating-point	Single-precision	2	702	16
Fixed-point	24×24	2	48	5
Fixed-point	24×18	1	0	4

Fixed-Point Format Design

How does the developer choose the number formats?

Use full precision, if feasible

Error analysis in software:



Bit-True Development: *en_cl_fix*

The **free** open-source *en_cl_fix* library handles all fixed-point math

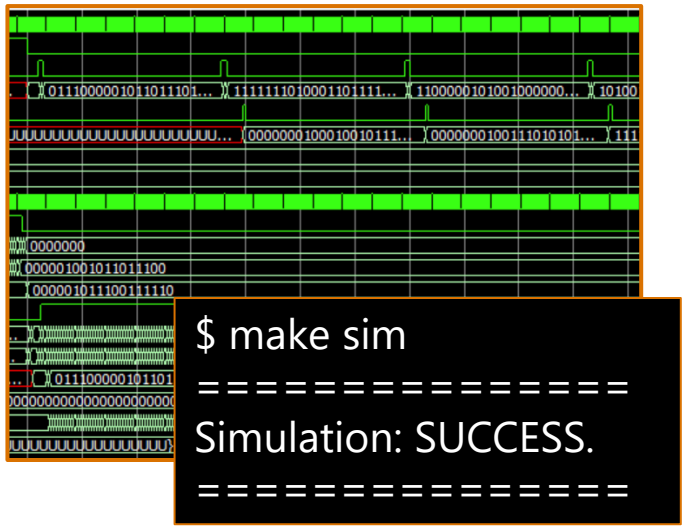
- Currently supports: VHDL, MATLAB and Python
- On the roadmap: Verilog

```python
# Python: c = a+b
c = cl_fix_add(a, a_fmt, b, b_fmt, c_fmt, FixRound.ConvEven_s, FixSaturate.Sat_s)
```

```vhdl
-- VHDL: c = a+b
c <= cl_fix_add(a, a_fmt, b, b_fmt, c_fmt, ConvEven_s, Sat_s);
```

www.github.com/enclustra/en_cl_fix

Bit-True Development: Step-by-Step



(1) Design

Floating-Point Model (Python)

Design Parameters

Performance Analysis

Bit-True Model (Python)

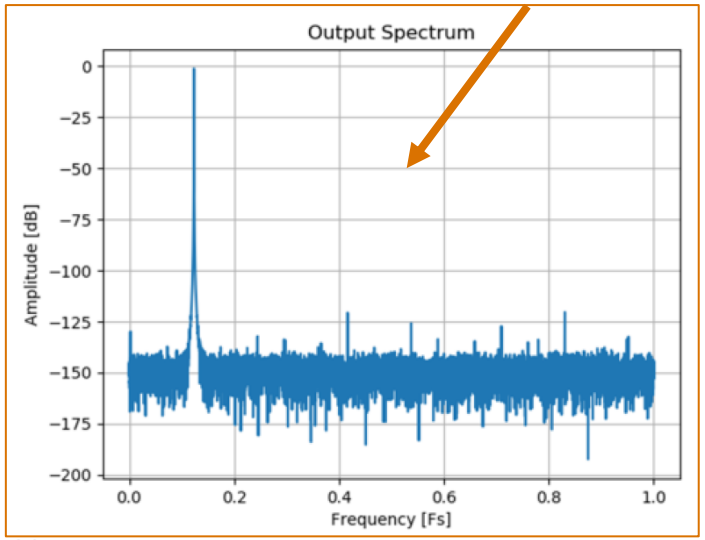
(3) Verify

RTL Implementation (HDL)

(2) Implement

HDL Testbench (e.g. Vunit)

Cosimulation data



Conclusion

- The Evolution of DSP
- Traditional DSP
- Parallel DSP
- Bit-True Development

Call To Action

Make Use of Enclustra Materials

- Open source *en_cl_fix* Library (GitHub)
- DSP to FPGA Whitepaper (Coming Soon)
- *Universal DSP Library* (Website)
  - Free evaluation license!
- Don't hesitate to contact us
  - info@enclustra.com

Design Services

- You have a project – Get in touch!
  - HW / FW / SW development
  - System design
  - Consulting
  - At any stage of the project
- info@enclustra.com

Our Mission is getting your Design Working!

Everything FPGA.

Thank you for your attention