Digital Signal processing with Field Programmable Gate Arrays

Beam Instrumentation Workshop 2008
Tahoe City, CA, USA
Javier Serrano, CERN, Geneva, Switzerland

Outline

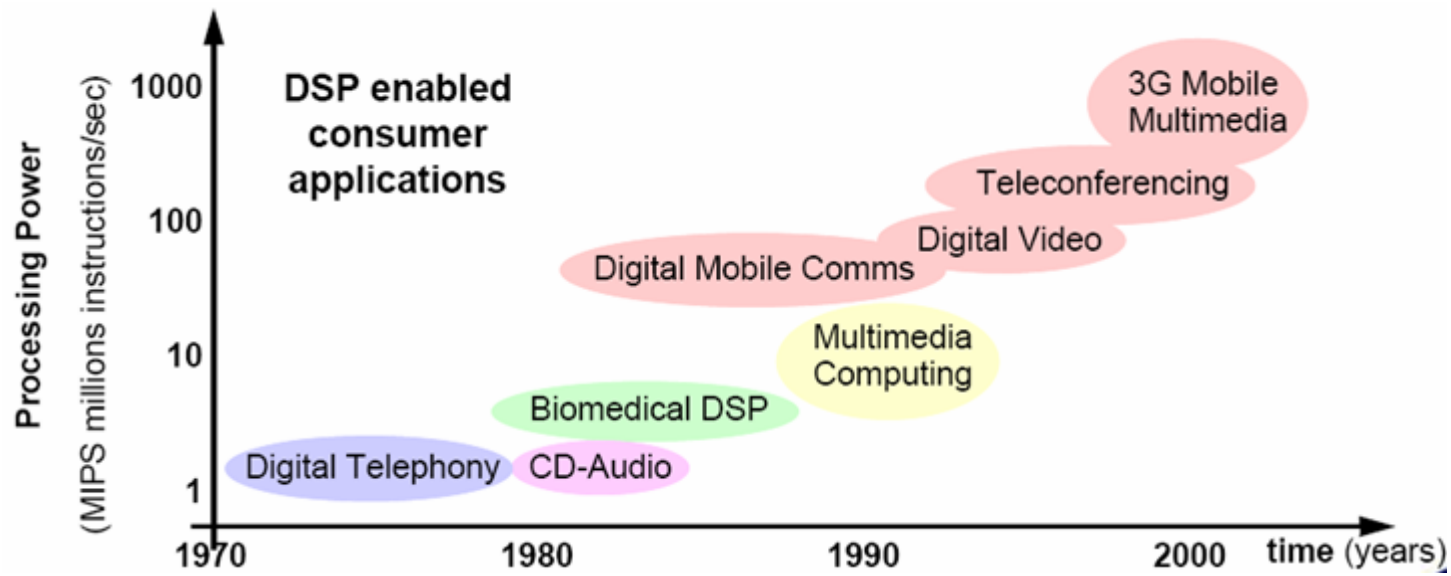
- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

Outline

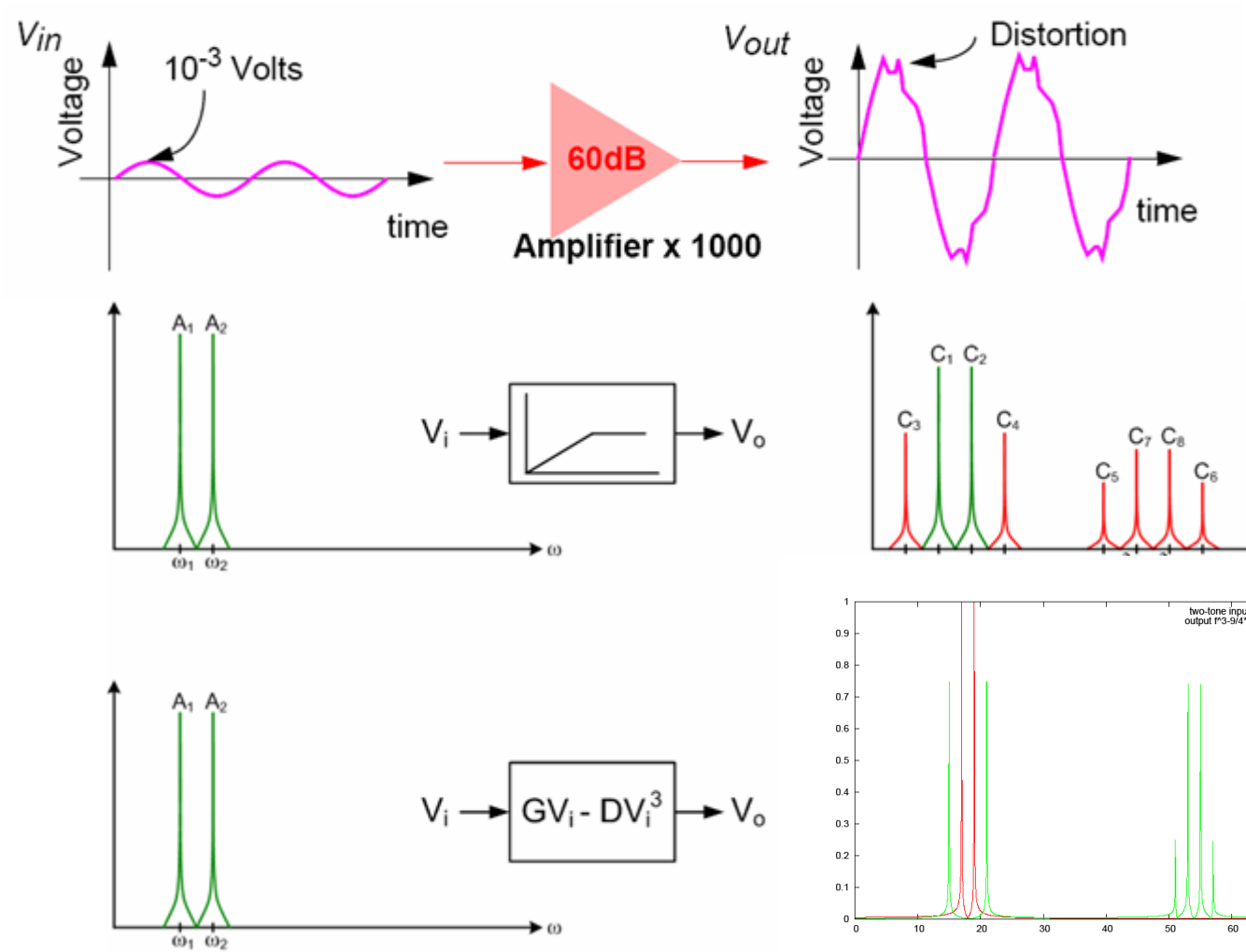
- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

The “DSP Revolution”

- DSP has displaced analog solutions in many consumer and industrial markets. Main reasons:
  - Reliability.
  - Repeatability.
  - Programmability.
  - Performance (e.g. FIR filters have no analog equivalent).

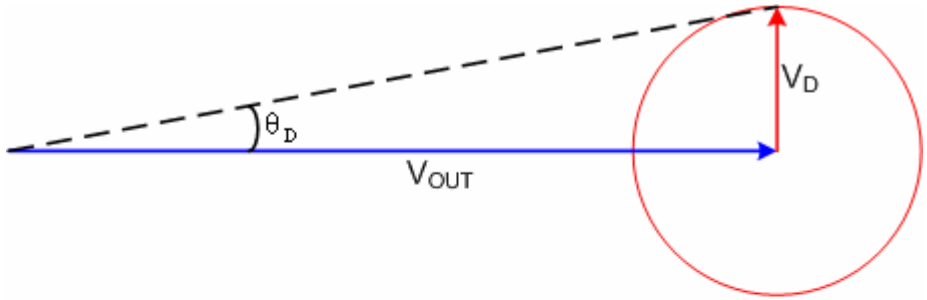


An example: amplifier and mixer distortion



Courtesy Tony Rohlev

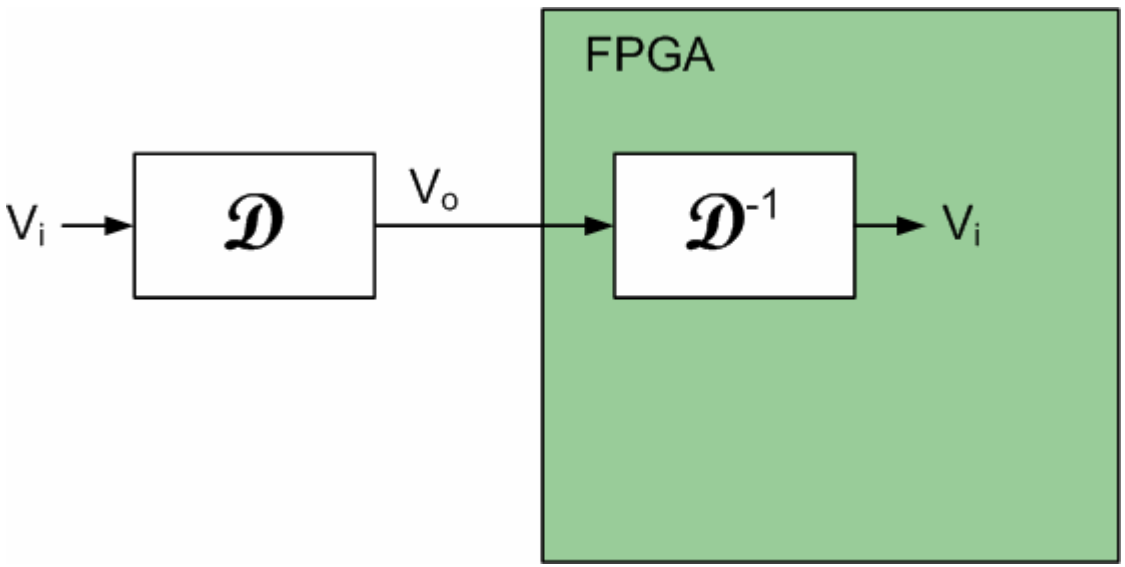
Saturation (IP 3) and its effects on phase accuracy



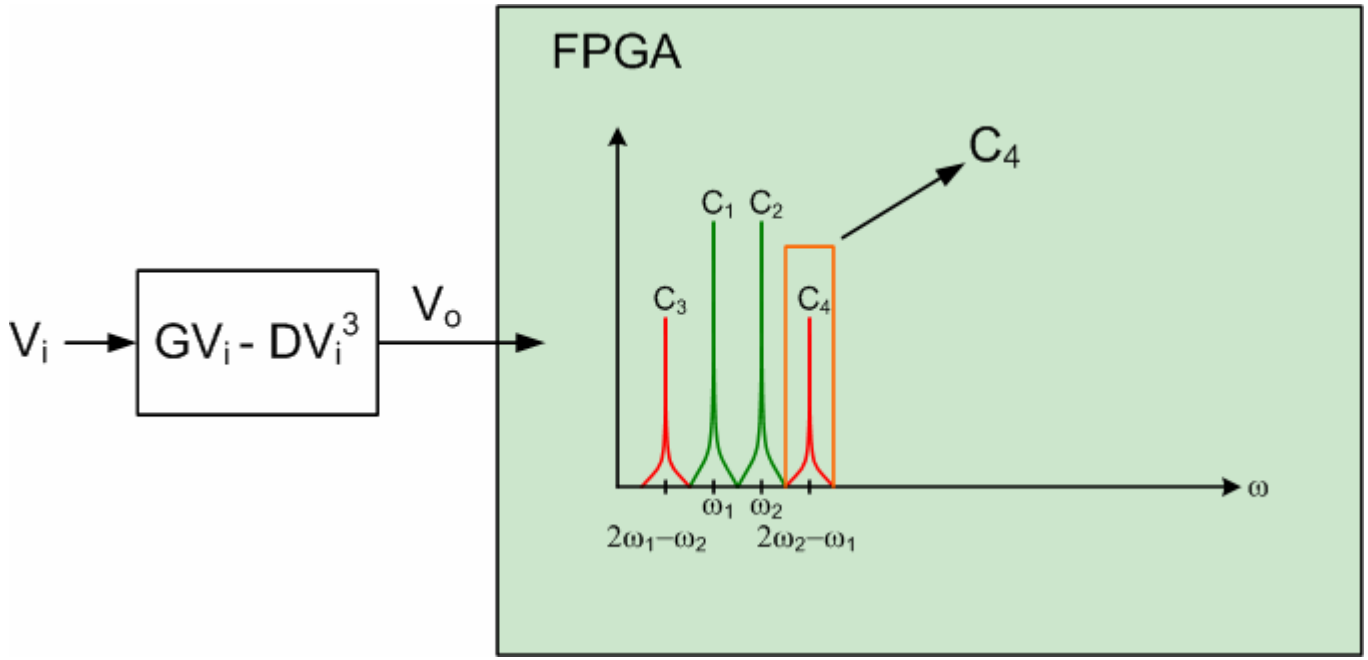
θ_{Dpk} = Distortion Phase (radians pk)
V_{OUT} = Output Signal Voltage (V pk)
V_D = Distortion Voltage (V pk)

$$\theta_{Dpk} = \tan^{-1}\left(\frac{V_D}{V_{OUT}}\right)$$

Strategy: model and “invert”



Finding coefficients D and G



$$C_1 = GA_1 - \frac{3}{4}DA_1^3 - \frac{3}{2}DA_1A_2^2$$
$$C_2 = GA_2 - \frac{3}{4}DA_2^3 - \frac{3}{2}DA_1^2A_2$$
$$C_3 = -\frac{3}{4}DA_1^2A_2$$
$$C_4 = -\frac{3}{4}DA_1A_2^2$$

$$D = -\frac{4C_3}{3A_1^2A_2} = -\frac{4C_4}{3A_1A_2^2}$$

$$G - \frac{C_1}{A_1} - \frac{C_4}{A_1}\left(\frac{A_1^2}{A_2^2} + 2\right) - \frac{C_2}{A_2} - \frac{C_3}{A_2}\left(\frac{A_2^2}{A_1^2} + 2\right)$$

Extracting the original V_i

Direct?

$$V_i(V_o) = \frac{1}{\sqrt[3]{D}}\left(\sqrt[3]{-\frac{V_o}{2}+\sqrt{\frac{V_o^2}{4}-\frac{G^3}{27D}}}+\sqrt[3]{-\frac{V_o}{2}-\sqrt{\frac{V_o^2}{4}-\frac{G^3}{27D}}}\right)$$

NO.

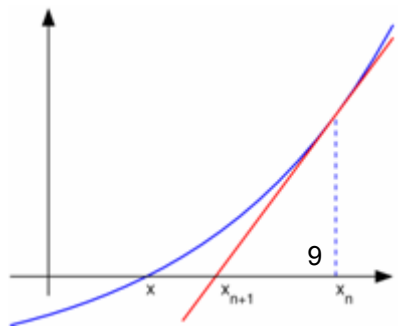
Taylor Expansion?

$$V_i(x) = \sum_{n=0}^{\infty} \frac{1}{n!}\frac{d^n V_i(c)}{dV_o^n}(x-c)^n$$

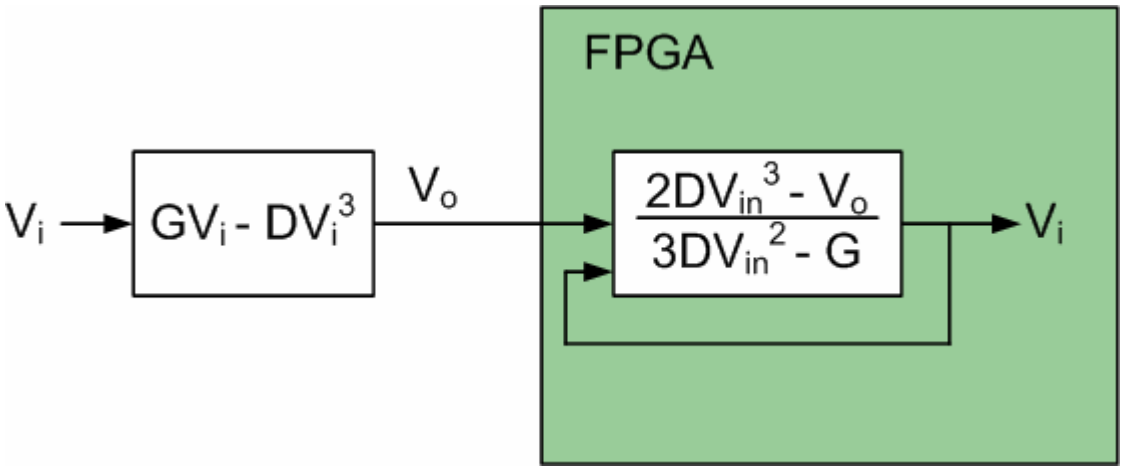
PLEASE NO!

Newton's Method?

$$V_{i_{n+1}} = V_{i_n} - \frac{f(V_{i_n})}{f'(V_{i_n})} = \frac{2DV_{i_n}^3 - V_o}{3DV_{i_n}^2 - G}$$

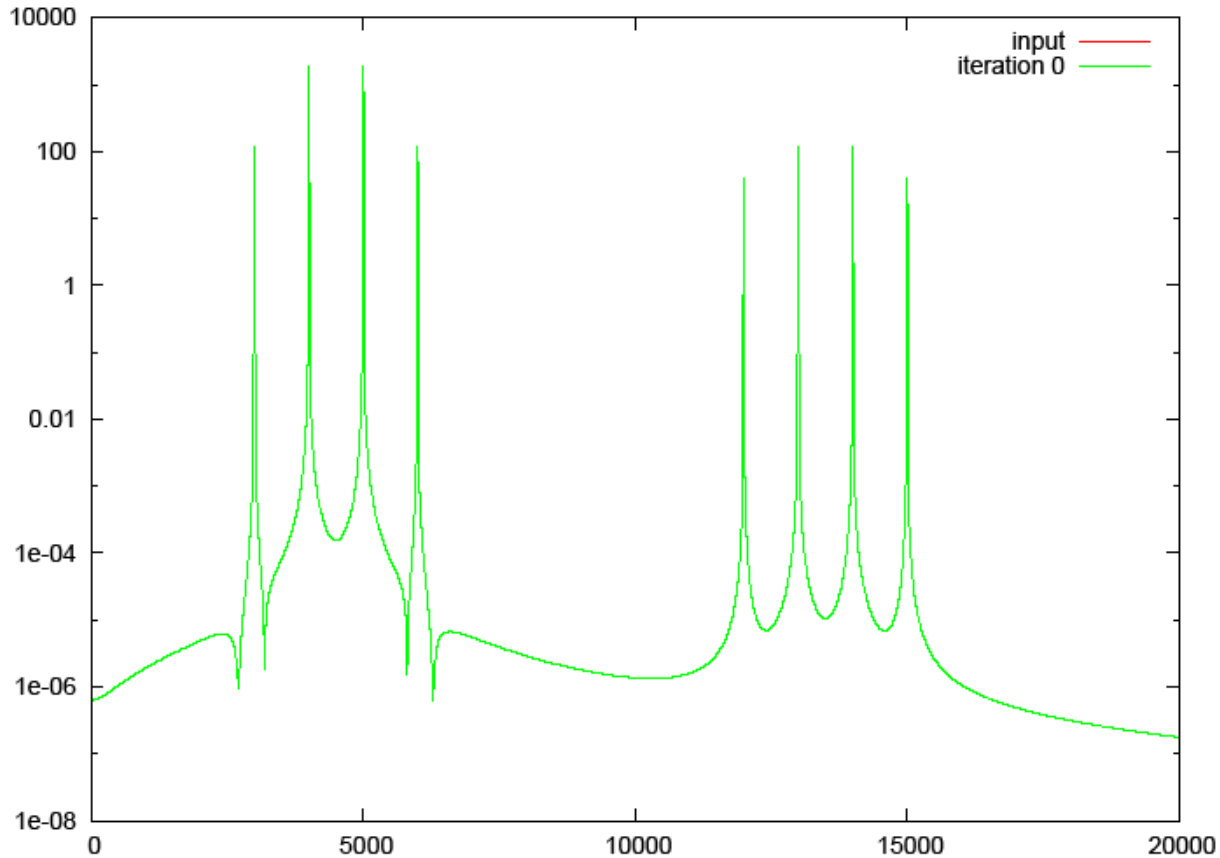


FPGA implementation

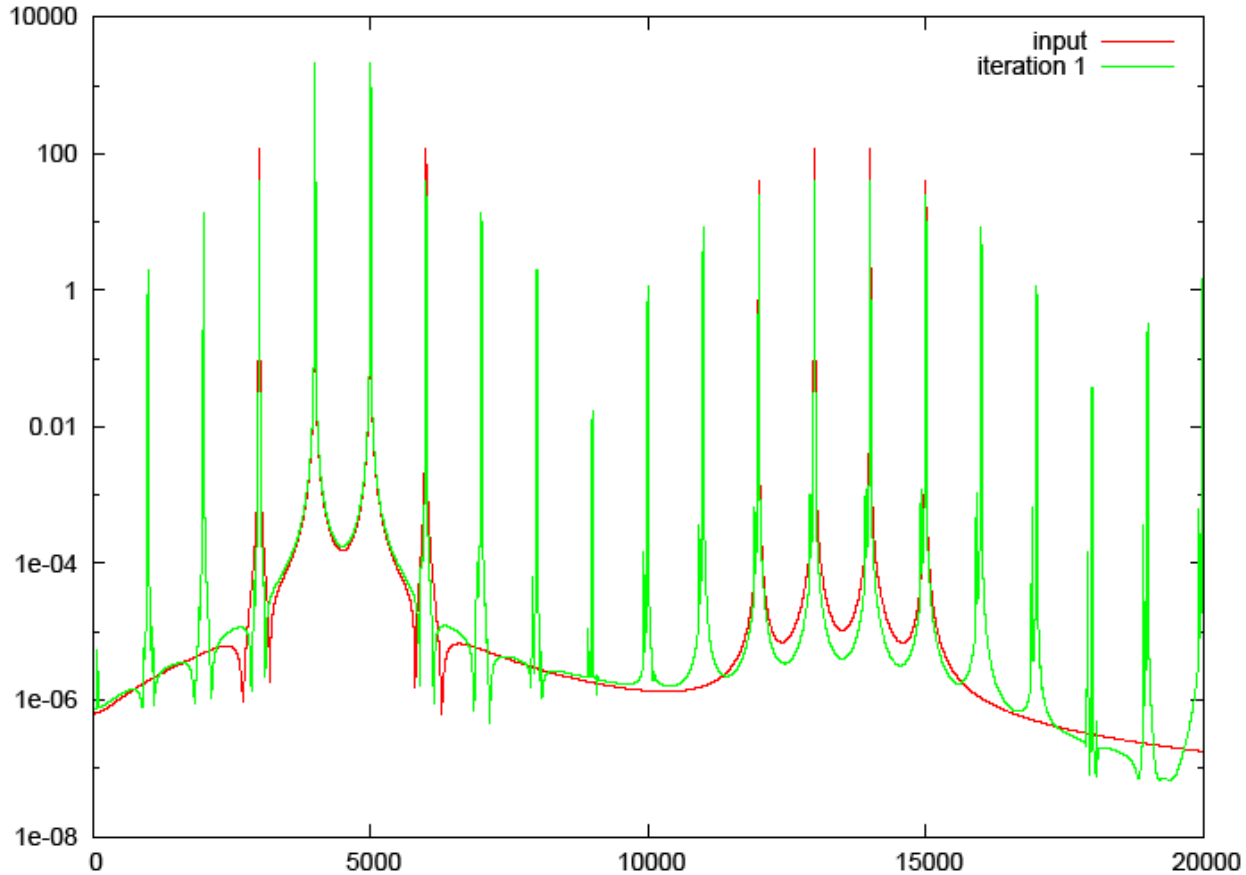


Or unroll the loop and increase throughput…

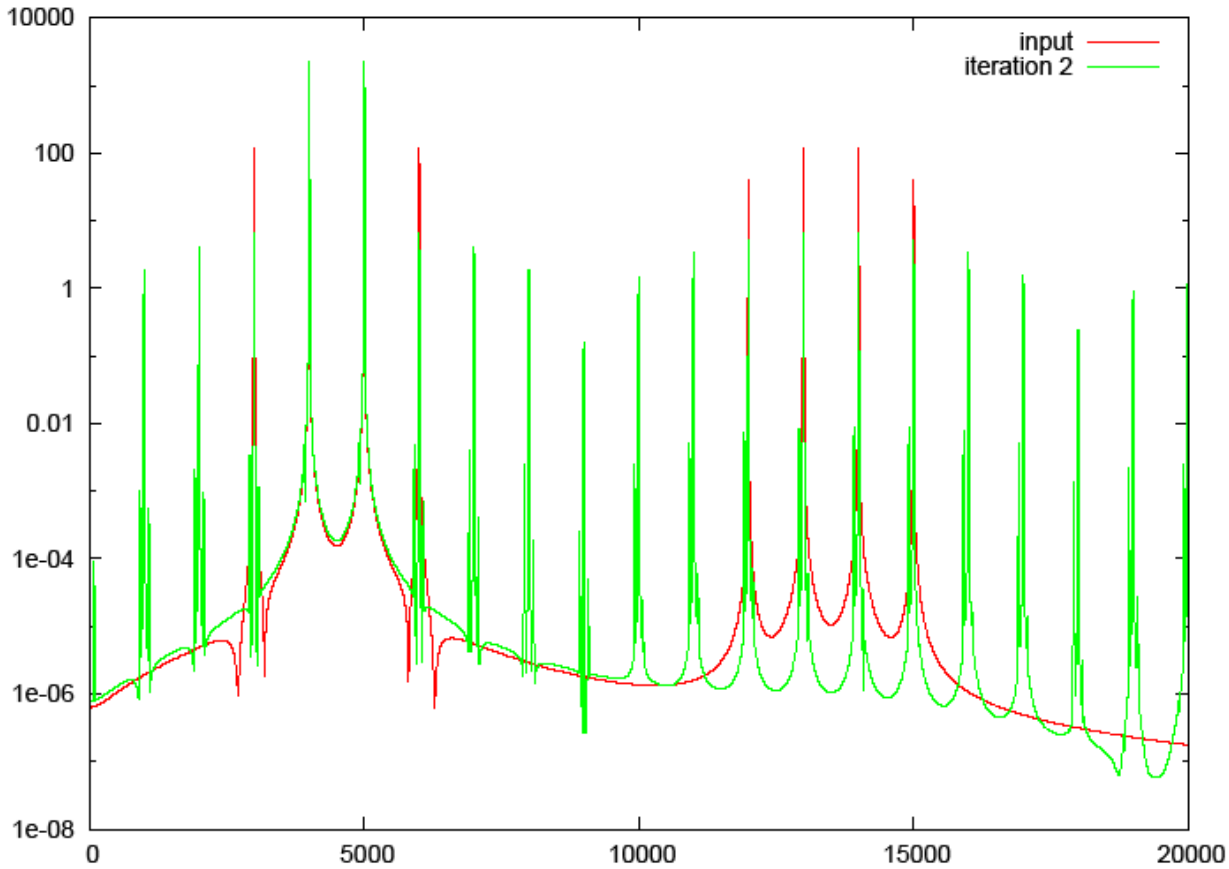
Simulation


input
iteration 0
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

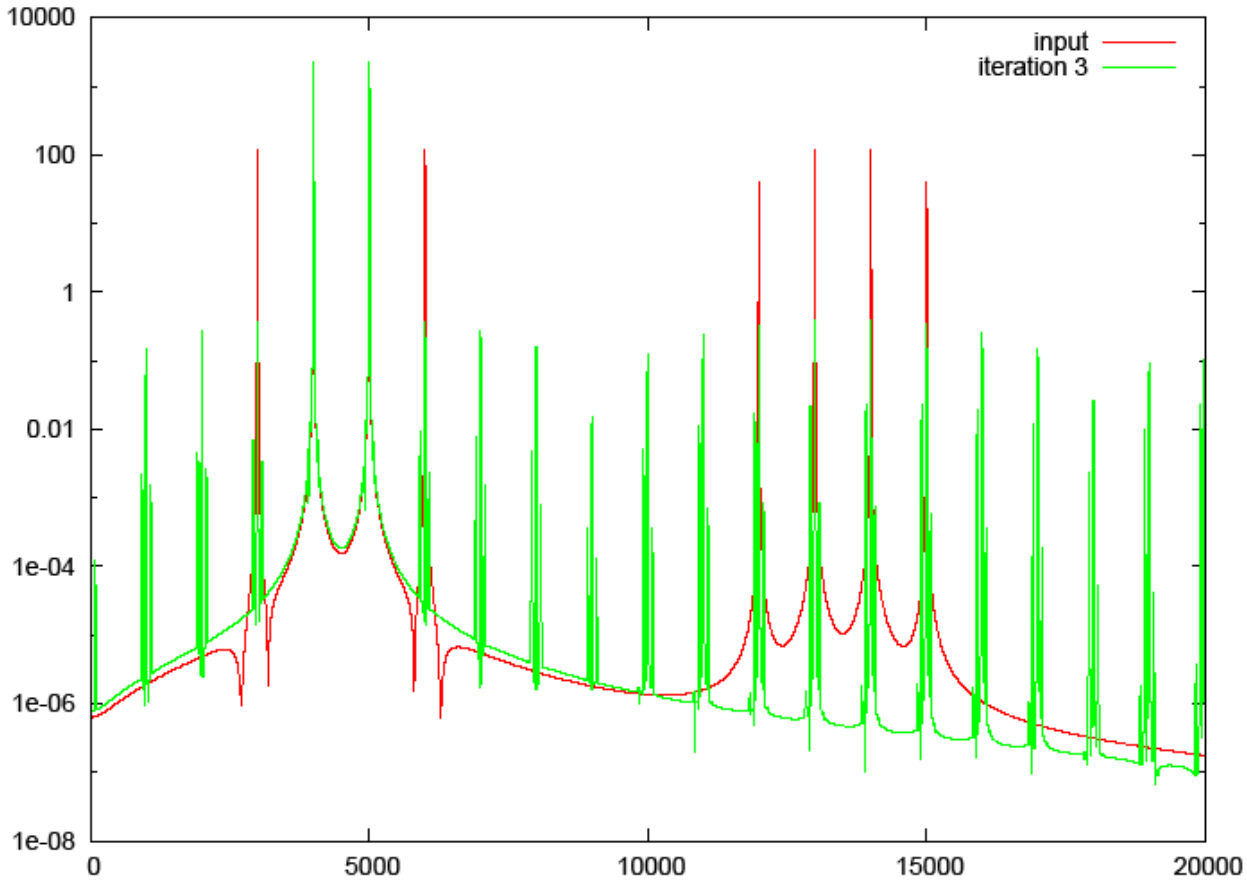
Simulation


input
iteration 1
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

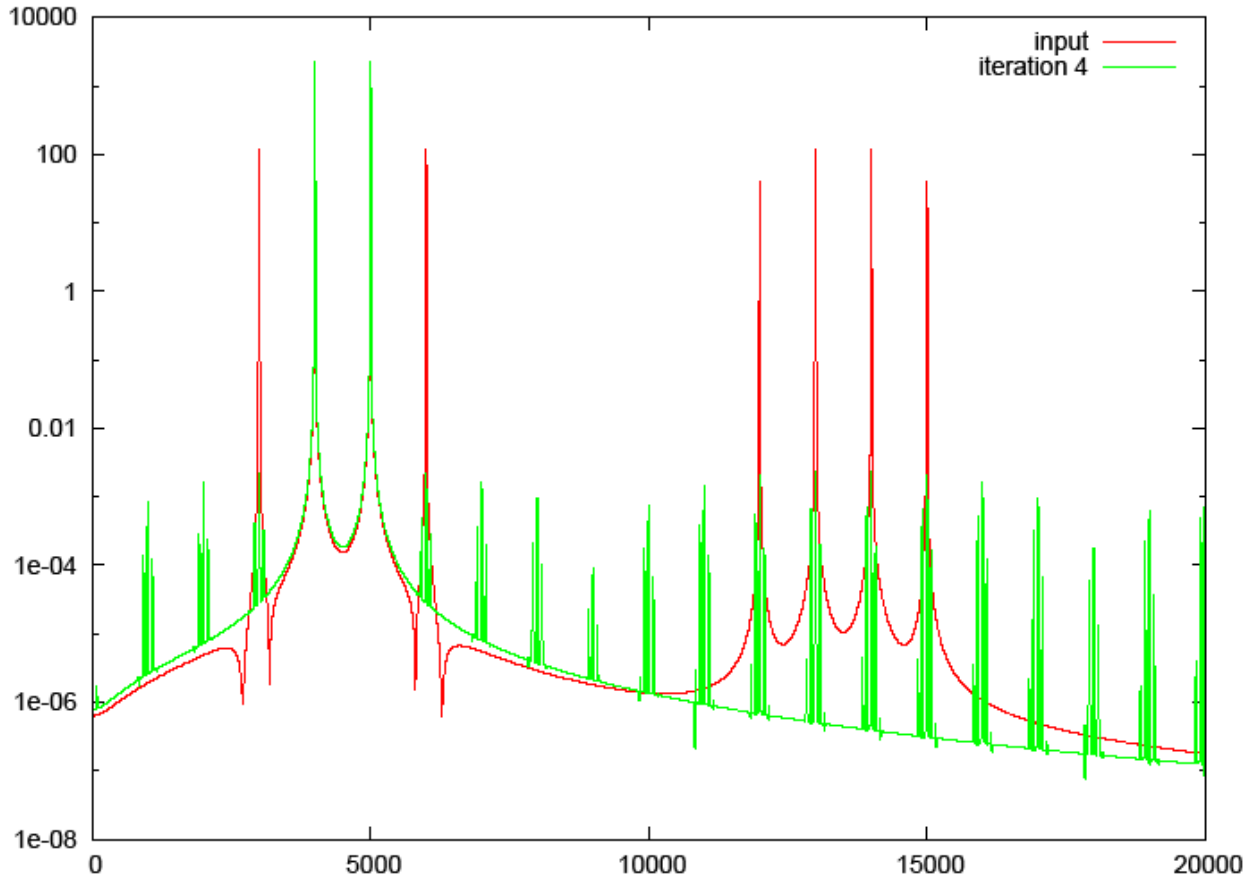
Simulation


input
iteration 2
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

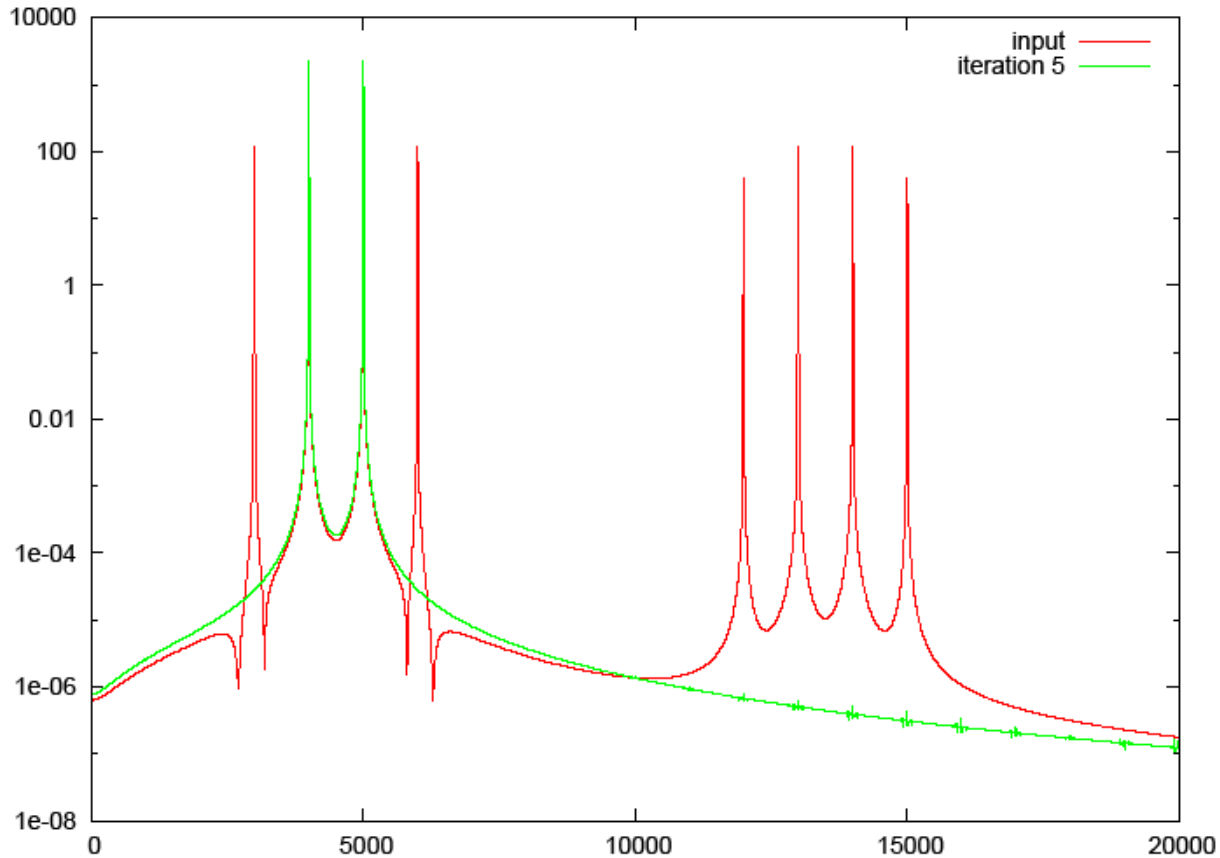
Simulation


input
iteration 3
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

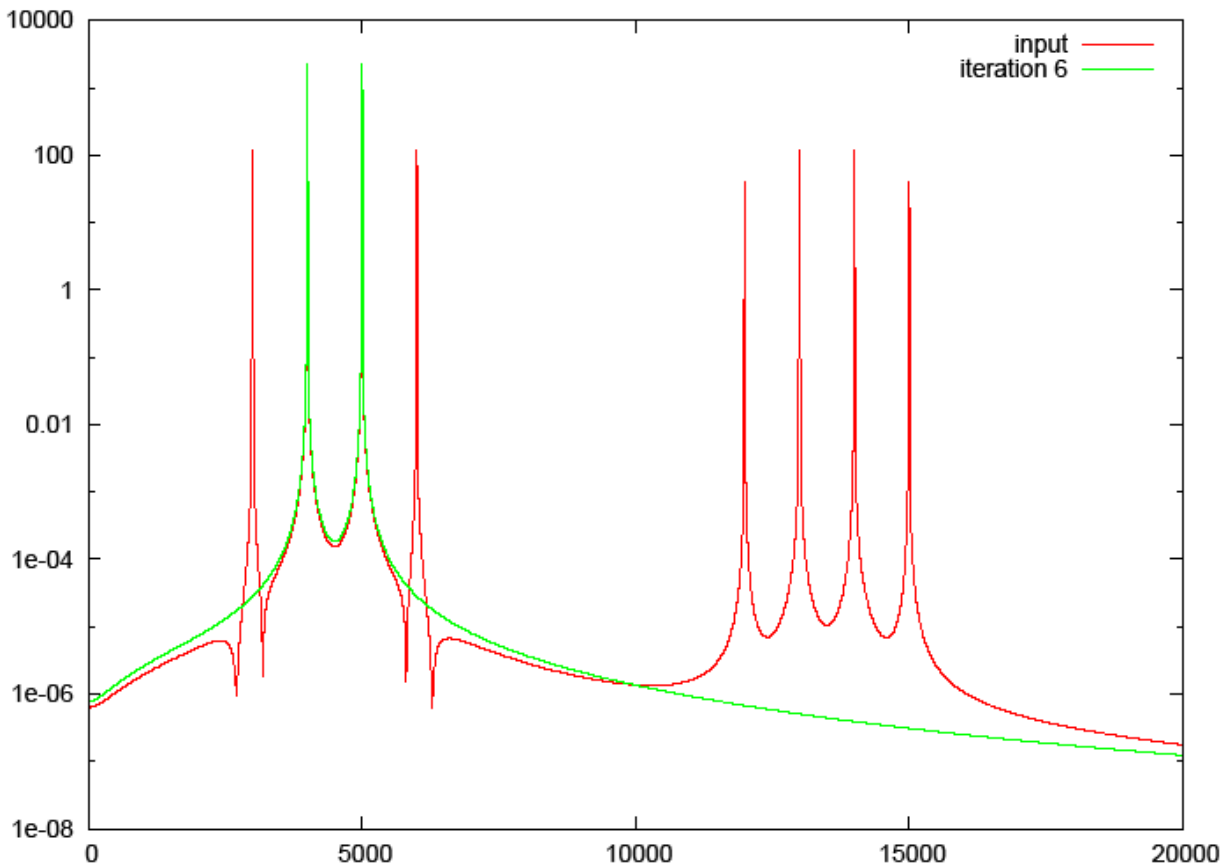
Simulation


input
iteration 4
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

Simulation


input
iteration 5
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

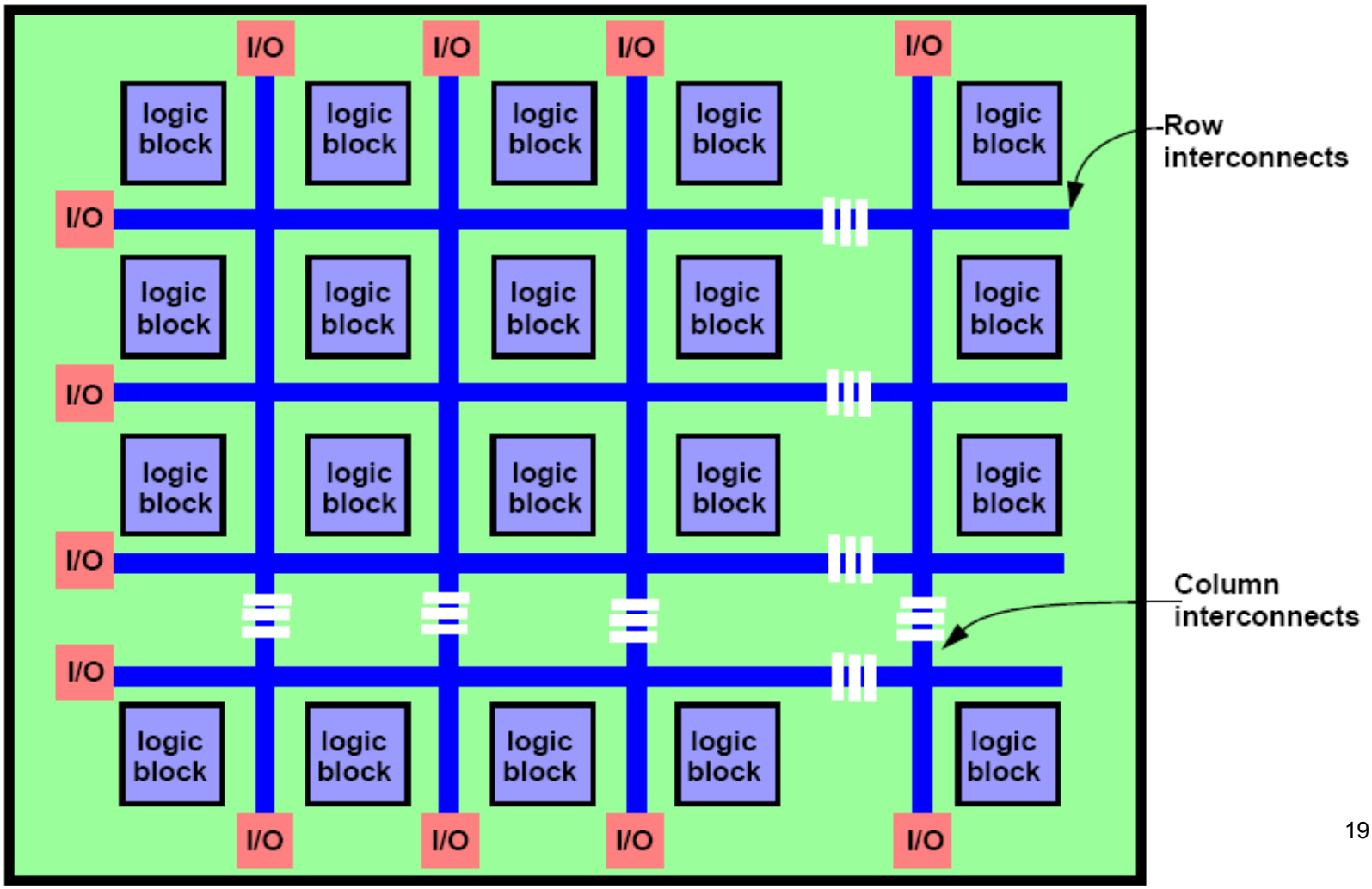
Simulation


input
iteration 6
10000
100
1
0.01
1e-04
1e-06
1e-08
0
5000
10000
15000
20000

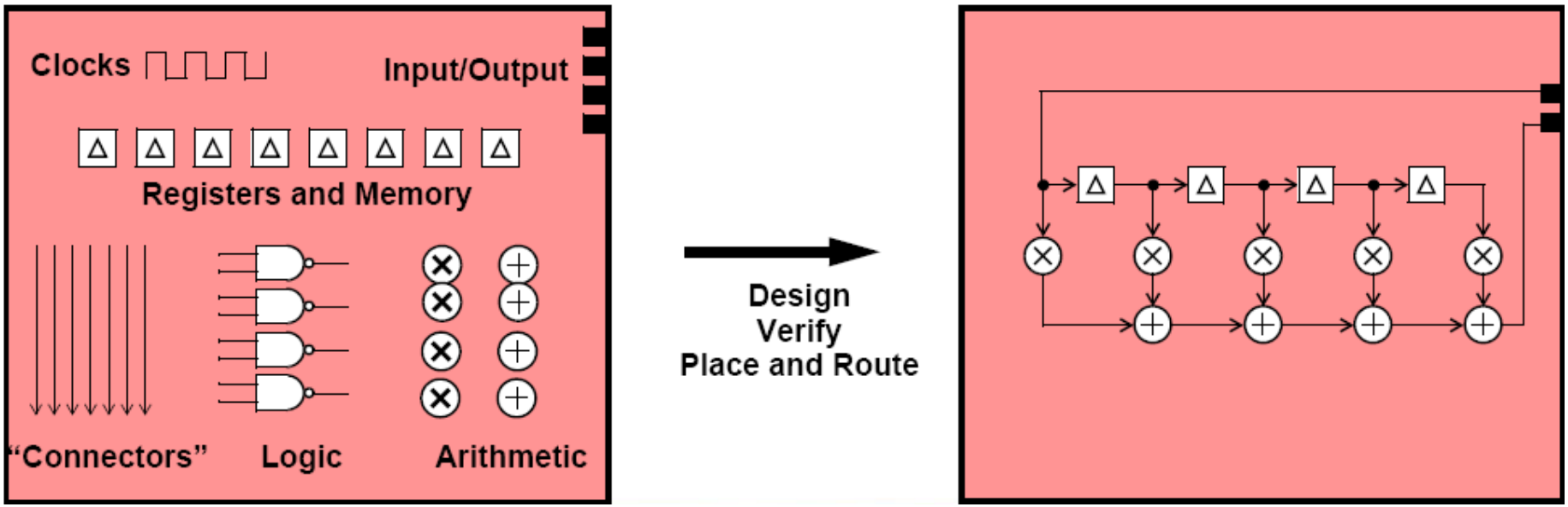
Outline

- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

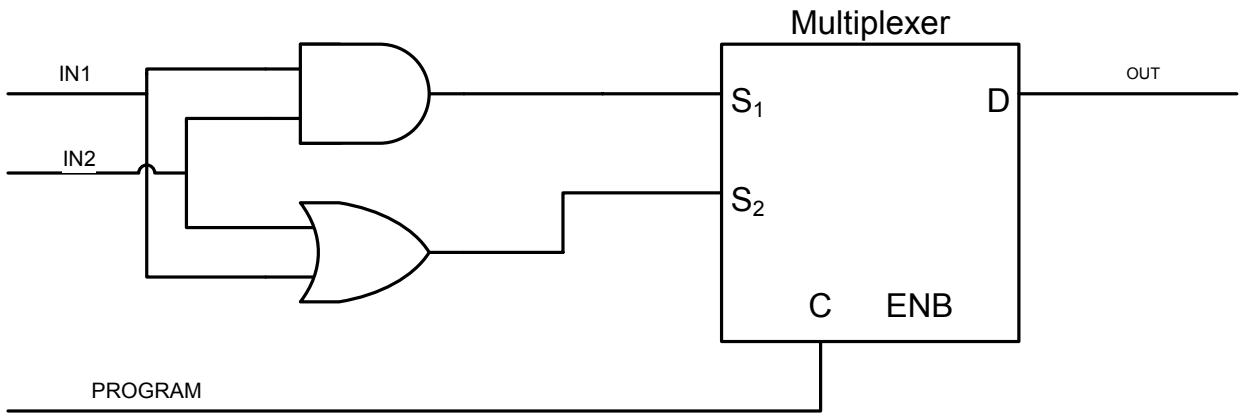
Basic FPGA architecture



FPGA: a “box” of DSP blocks



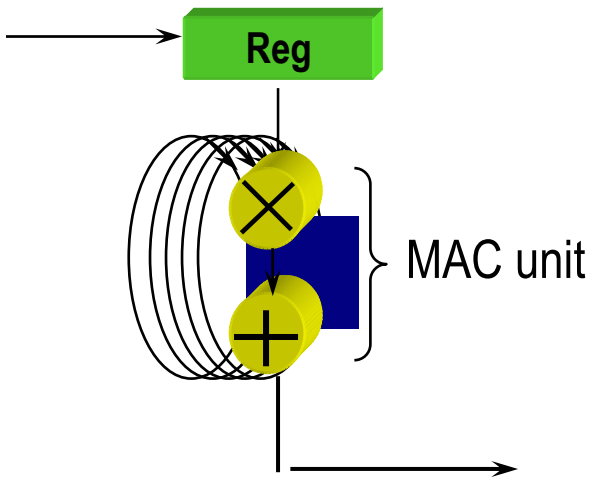
Programmable logic: one (simplistic) way of thinking of it



Why FPGAs for DSP? (1)

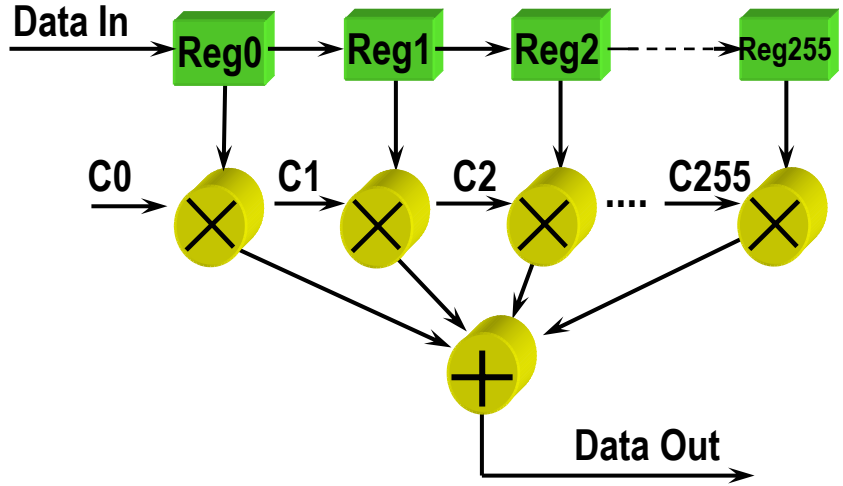
Reason 1: FPGAs handle high computational workloads

Conventional DSP Device
(Von Neumann architecture)



256 Loops needed to process samples

FPGA



All 256 MAC operations in 1 clock cycle

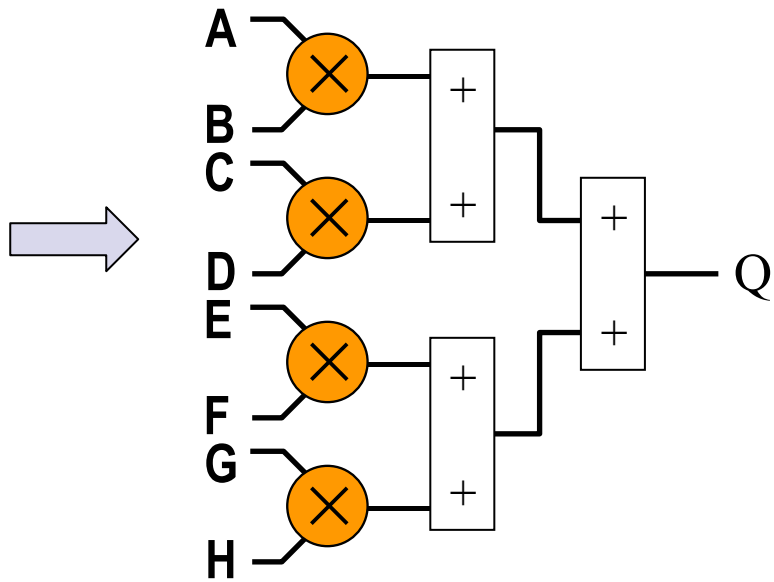
FPGAs benefit twice from Moore's law: speed *and* space.

Why FPGAs for DSP? (2)

Reason 2: Tremendous Flexibility

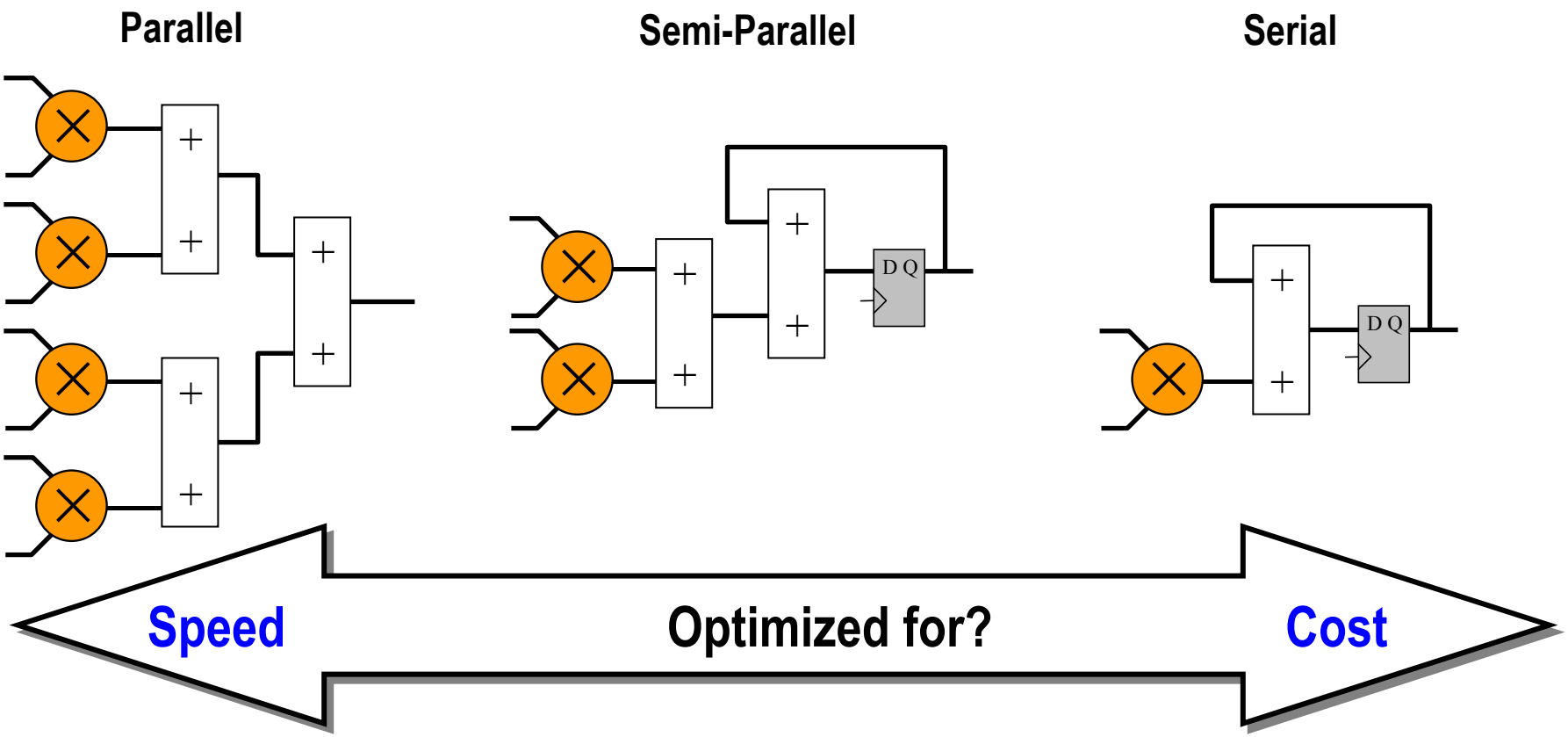
Q = (A x B) + (C x D) + (E x F) + (G x H)

can be implemented in parallel



But is this the only way in the FPGA?

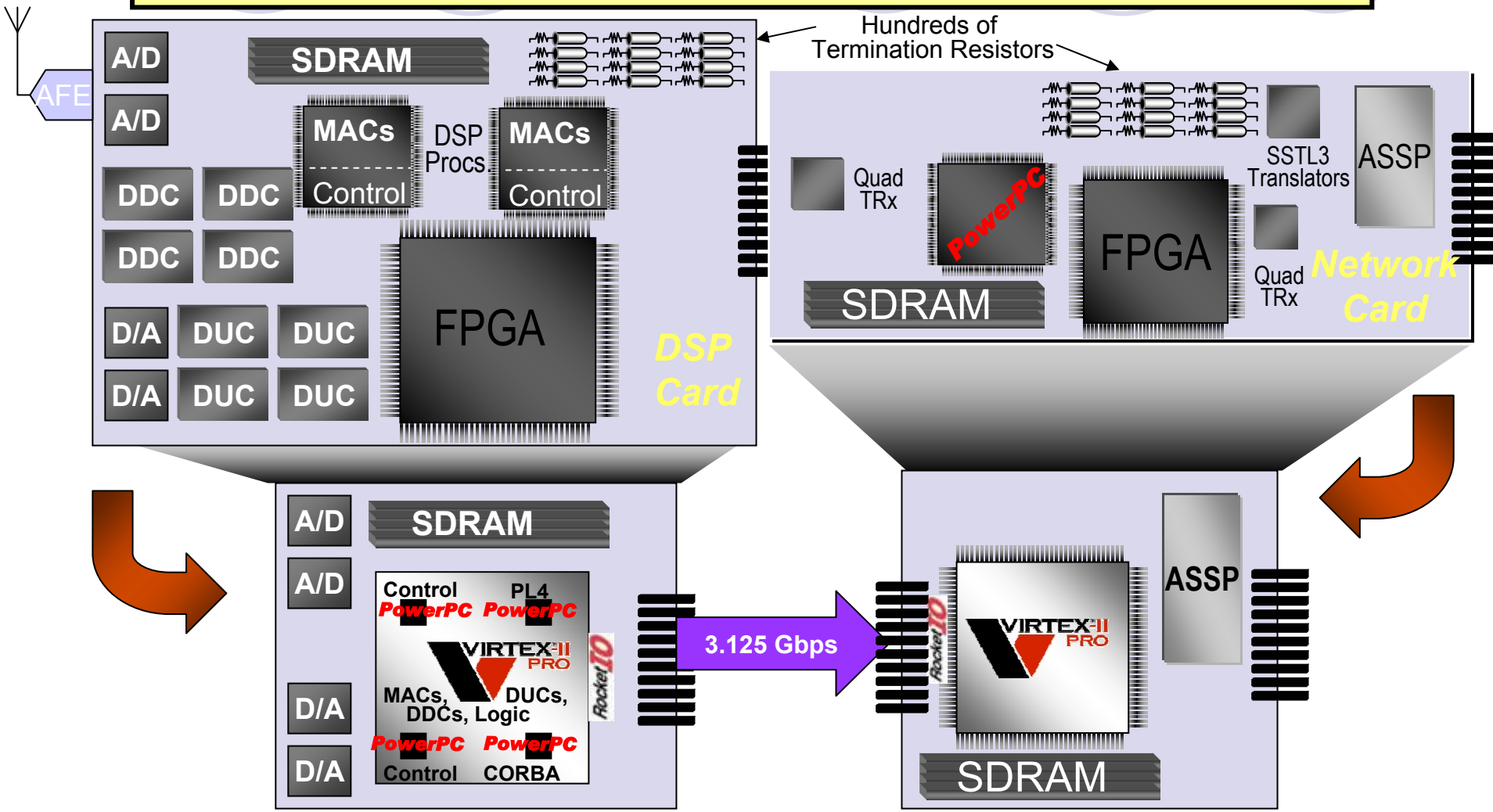
Customize Architectures to Suit Your Ideal Algorithms



FPGAs allow Area (cost) / Performance tradeoffs

Why FPGAs for DSP? (3)

Reason 3: Integration simplifies PCBs



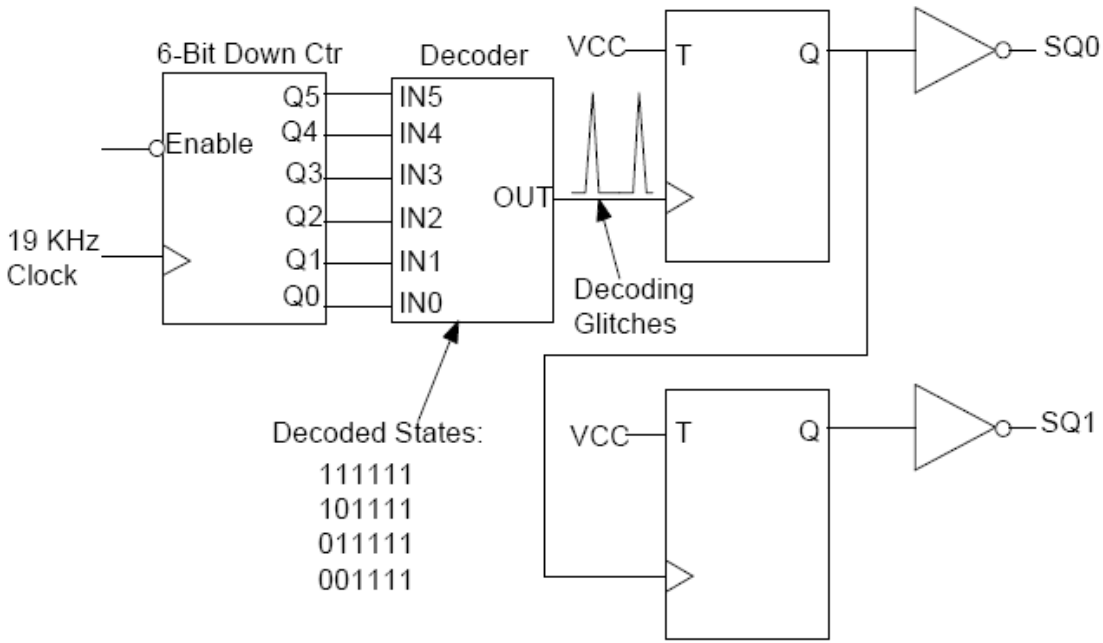
Outline

- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

Basics of digital design

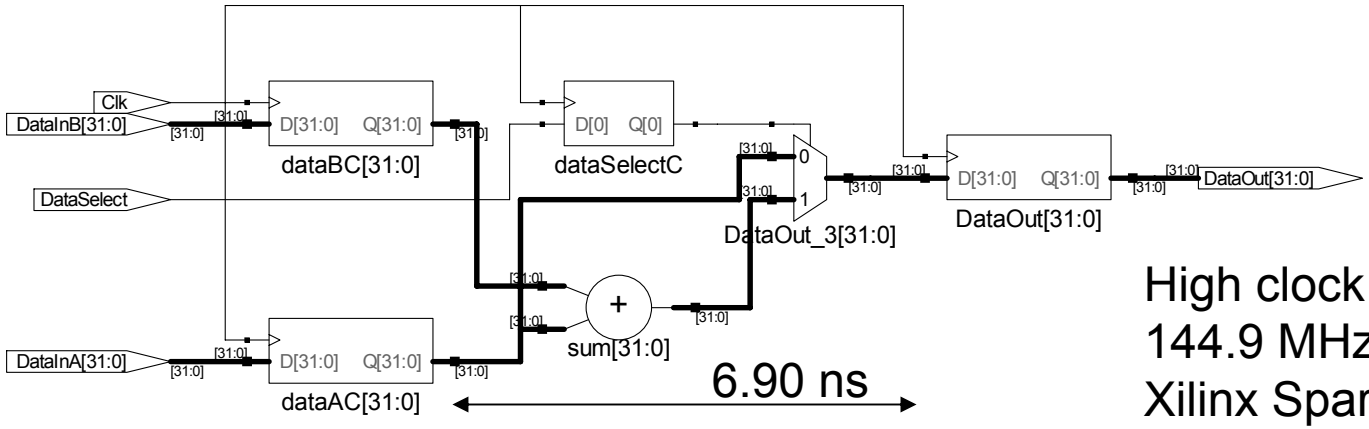
- Combinational logic: state of outputs depend on current state of inputs alone (forgetting about propagation delays for the time being). E.g. AND, OR, mux, decoder, adder...
- D-type Flip flops propagate D to Q upon a rising edge in the clk input.
- Unless you really know what you are doing, stick to synchronous design: sandwiching bunches of combinational logic in between flip flops.
- Synchronous design simplifies design analysis, which is good given today's logic densities.

Don’t do this!

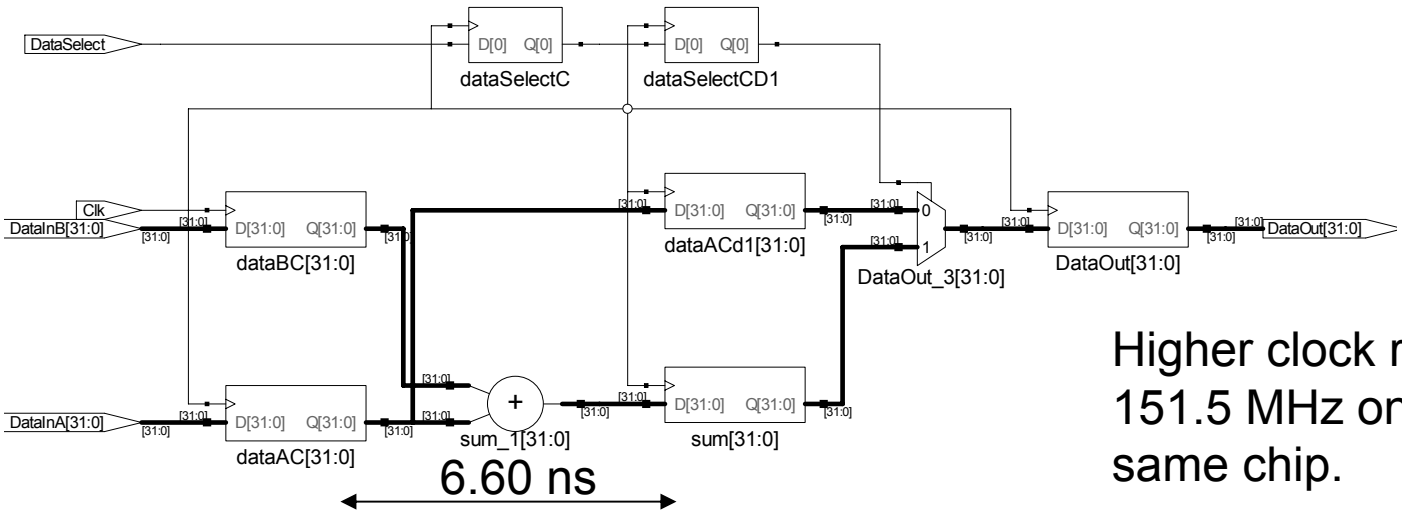


Toggle flip-flops get triggered by glitches produced by different path lengths of counter bits.

Basics of (synchronous) Digital Design



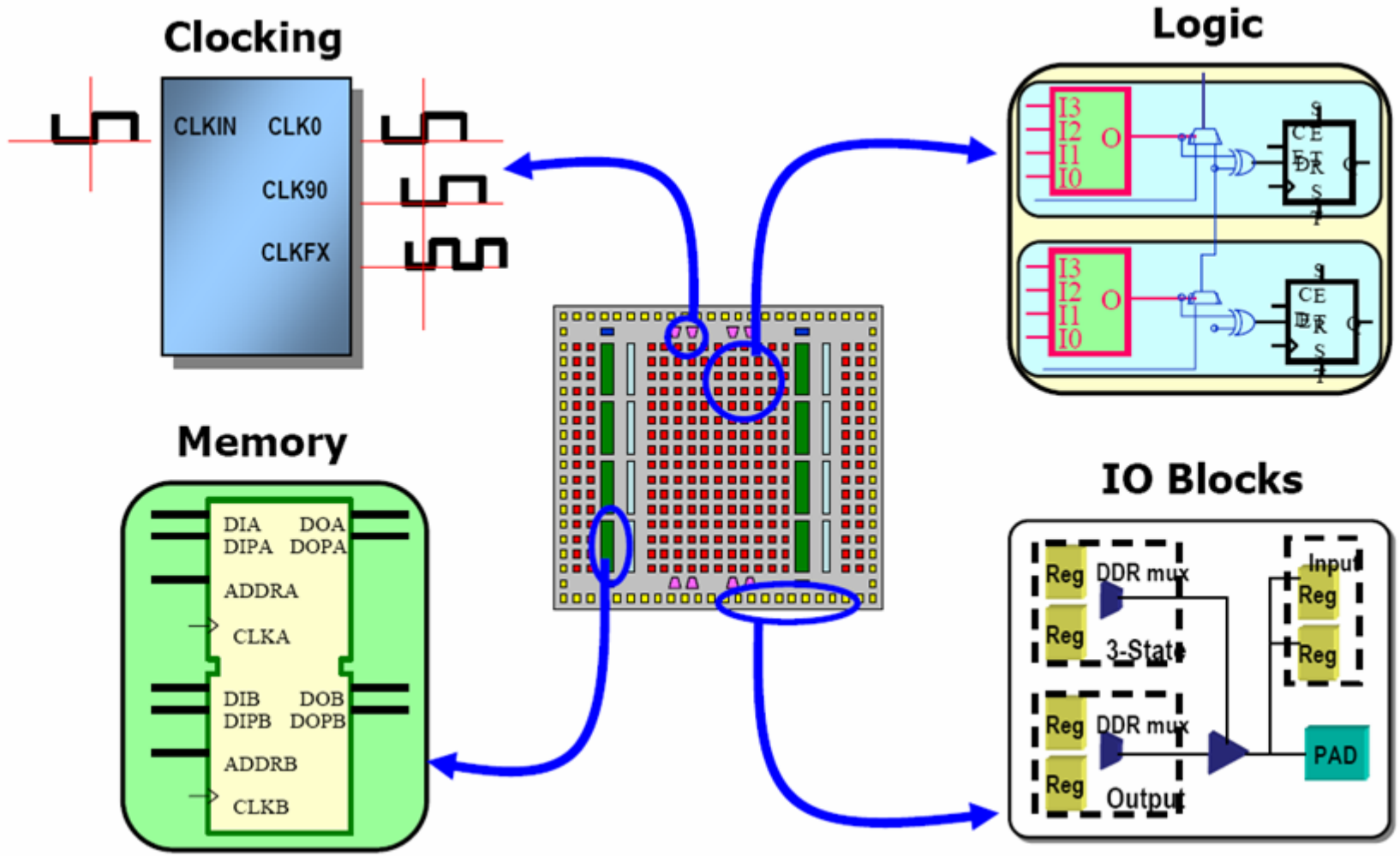
High clock rate: 144.9 MHz on a Xilinx Spartan IIE.



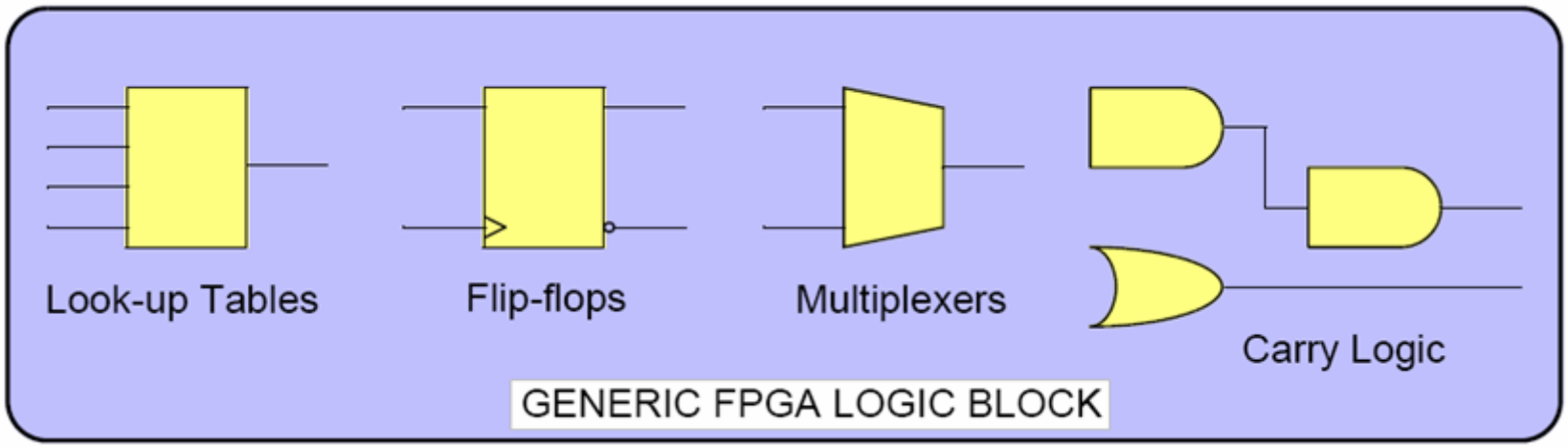
Higher clock rate: 151.5 MHz on the same chip.

Illustrating the latency/throughput tradeoff

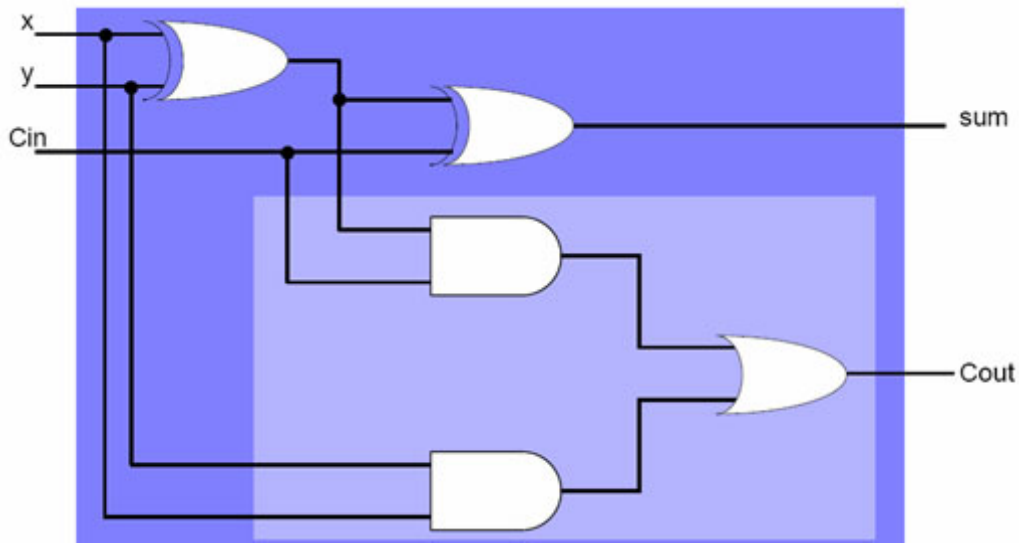
Basic FPGA architecture



The logic block: a summary view



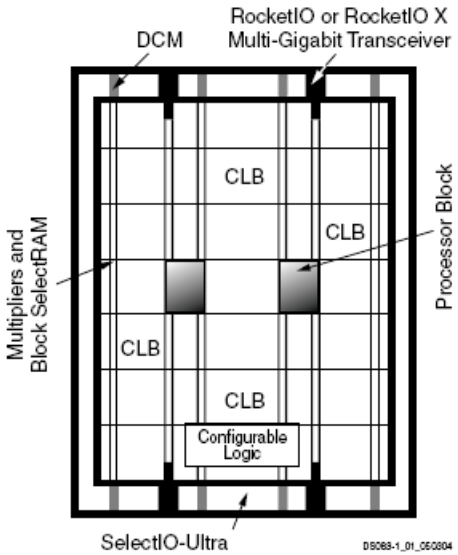
Example: using a LUT as a full adder.



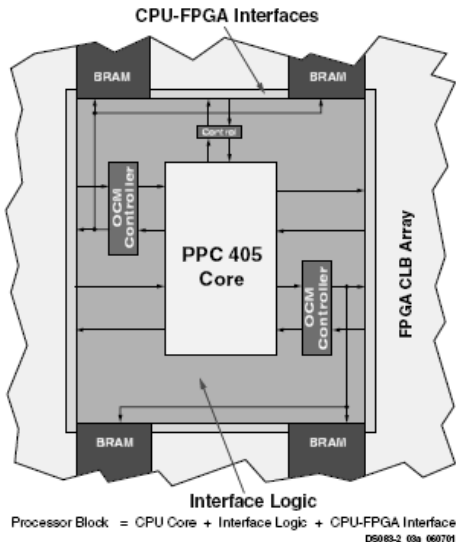
FPGA state of the art

- In addition to logic gates and routing, in a modern FPGA you can find:
  - Embedded processors (soft or hard).
  - Multi-Gb/s transceivers with equalization and hard IP for serial standards as PCI Express and Gbit Ethernet.
  - Lots of embedded MAC units, with enough bits to implement single precision floating point arithmetic efficiently.
  - Lots of dual-port RAM.
  - Sophisticated clock management through DLLs and PLLs.
  - System monitoring infrastructure including ADCs.
  - On-substrate decoupling capacitors to ease PCB design.
  - Digitally Controlled Impedance to eliminate on-board termination resistors.

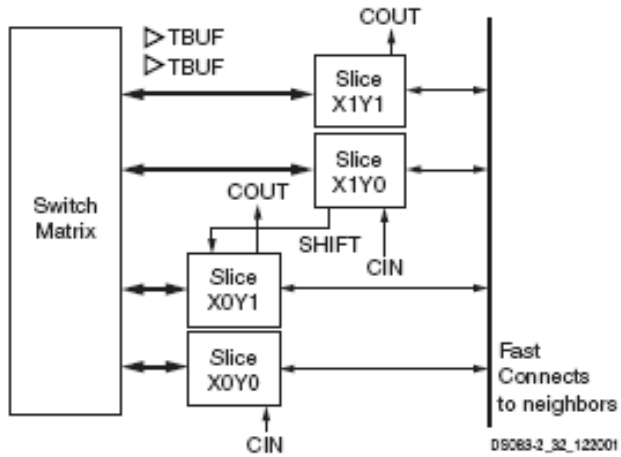
A practical example: Xilinx Virtex II Pro family



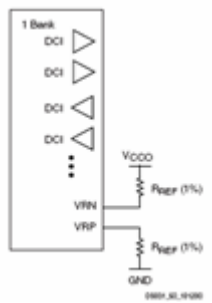
Overview



Embedded PowerPC

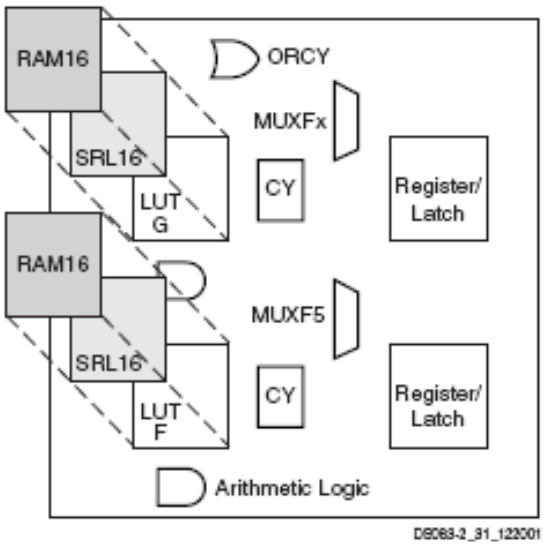


Configurable Logic Block (CLB)

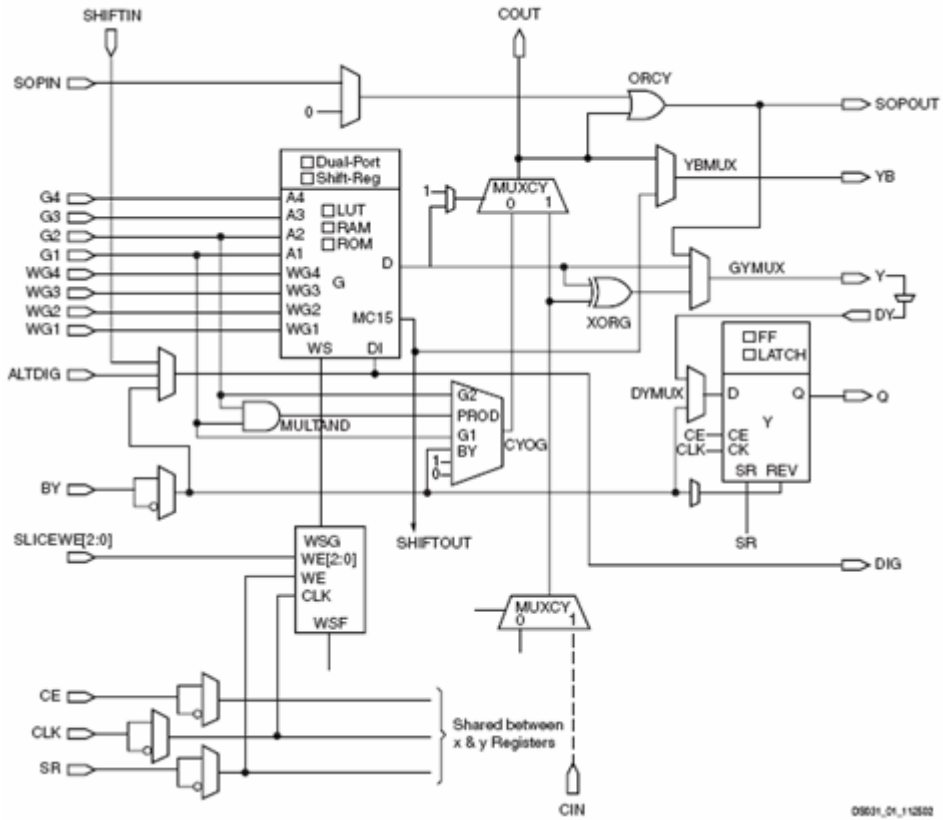


Digitally Controlled Impedance (DCI)

A practical example: Xilinx Virtex II Pro family

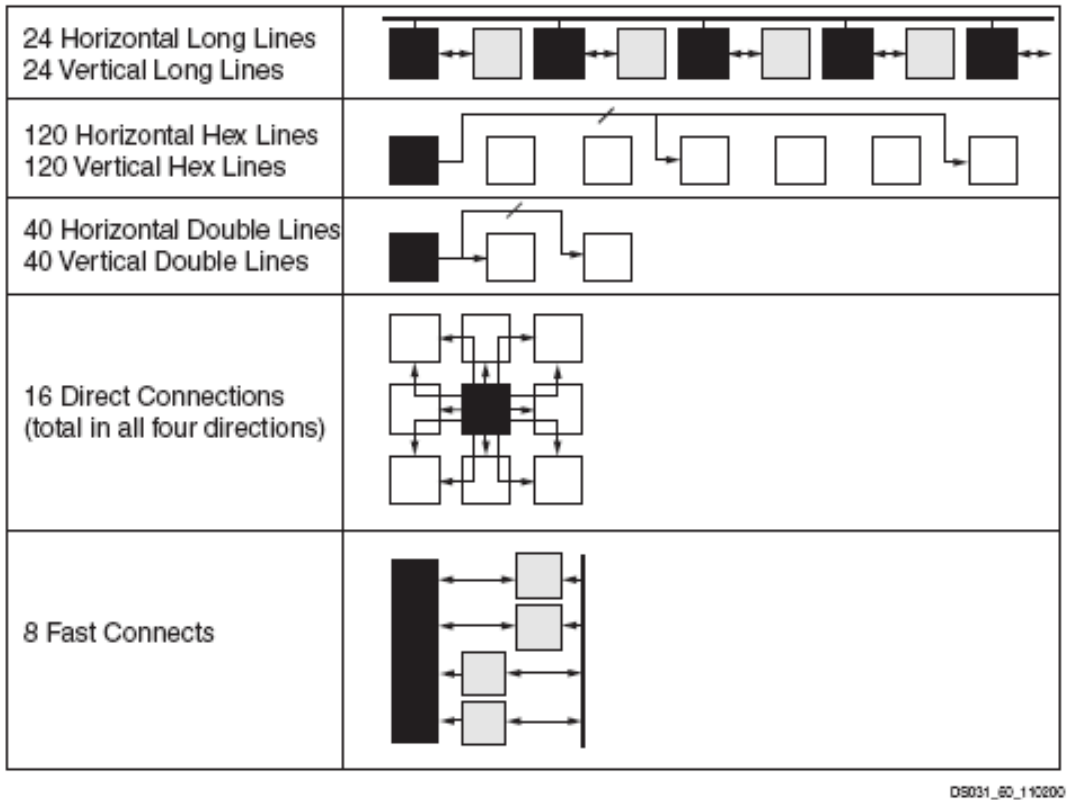


Slice



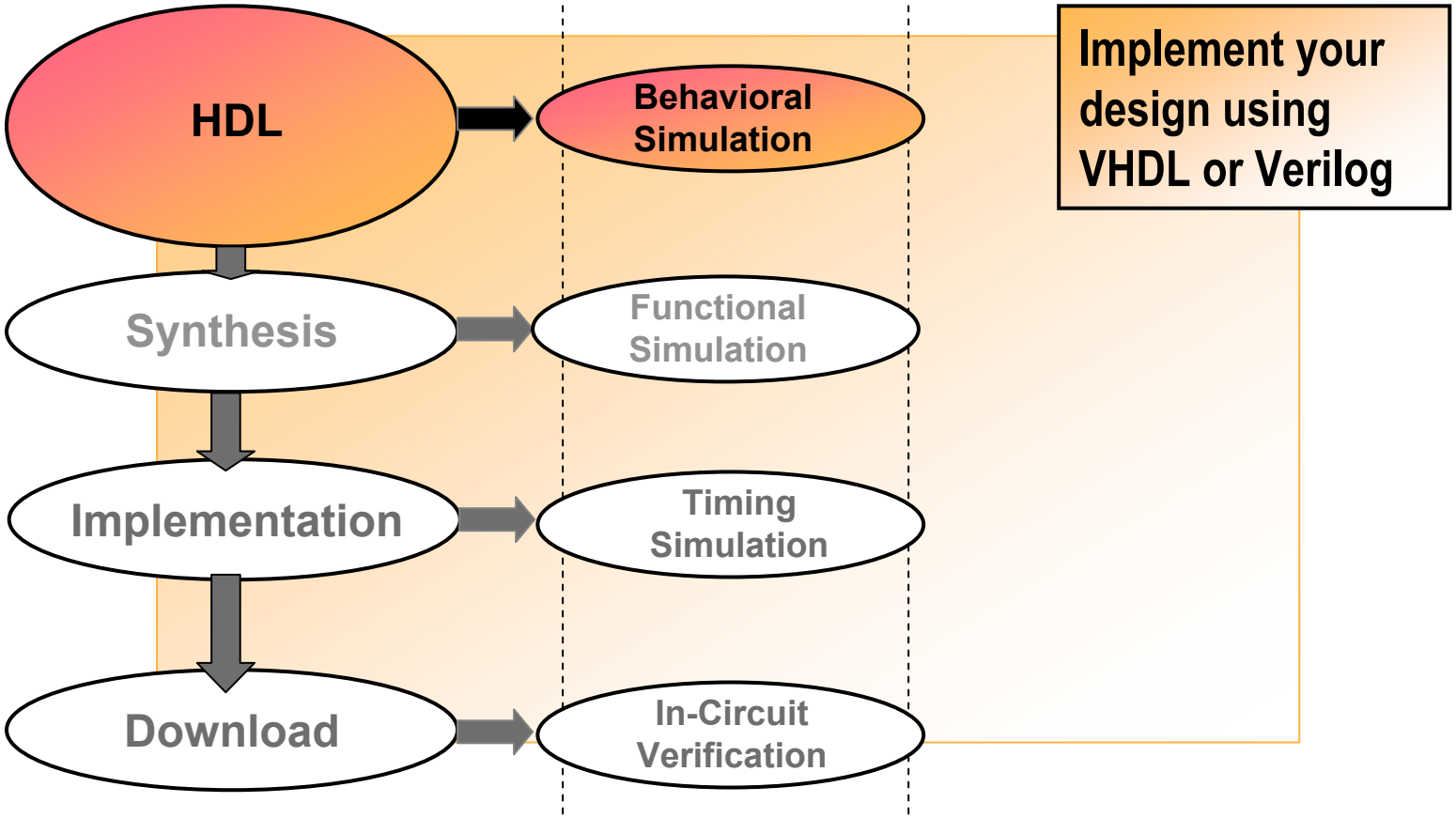
Detail of half-slice

A practical example: Xilinx Virtex II Pro family

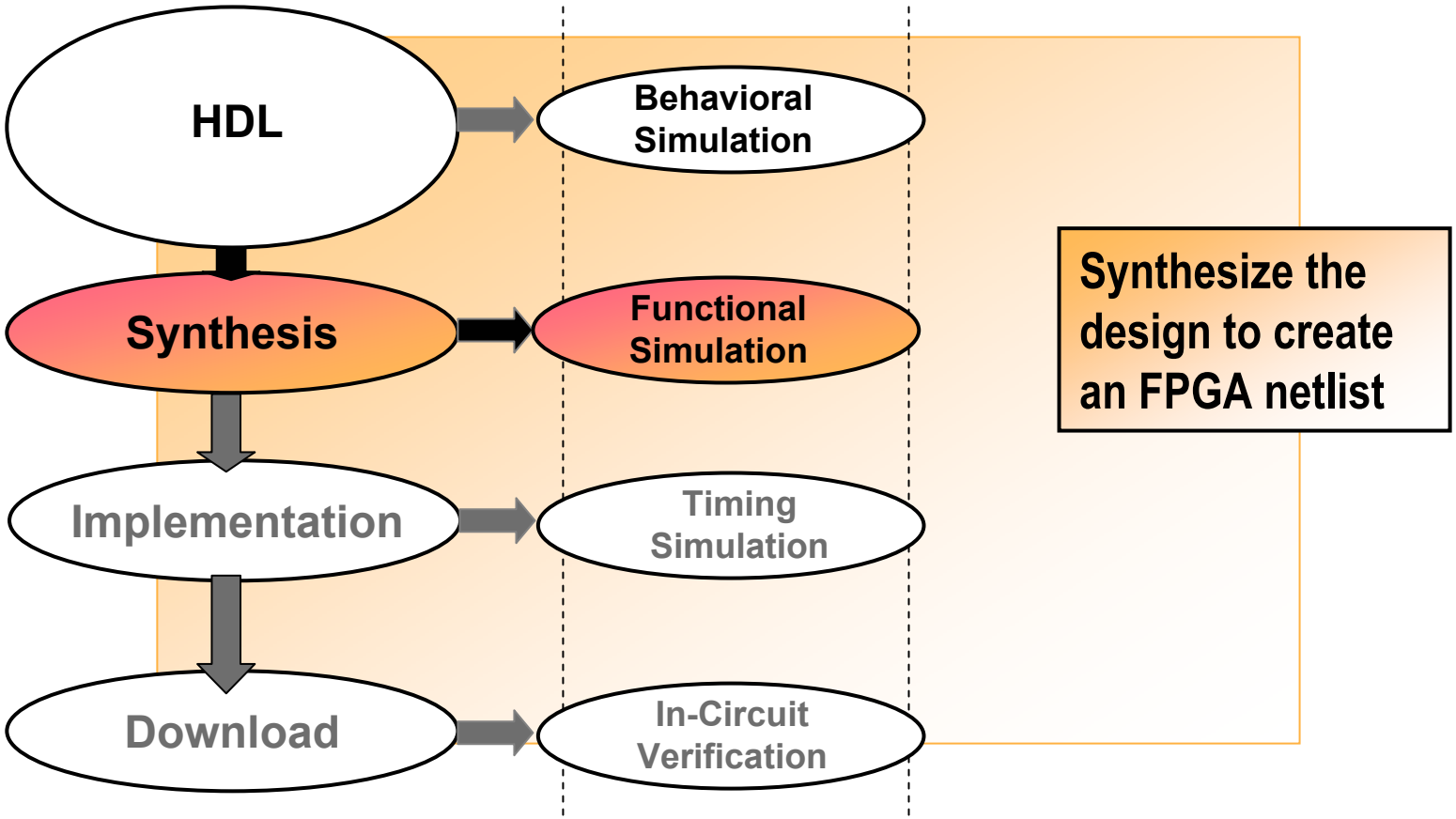


Routing resources

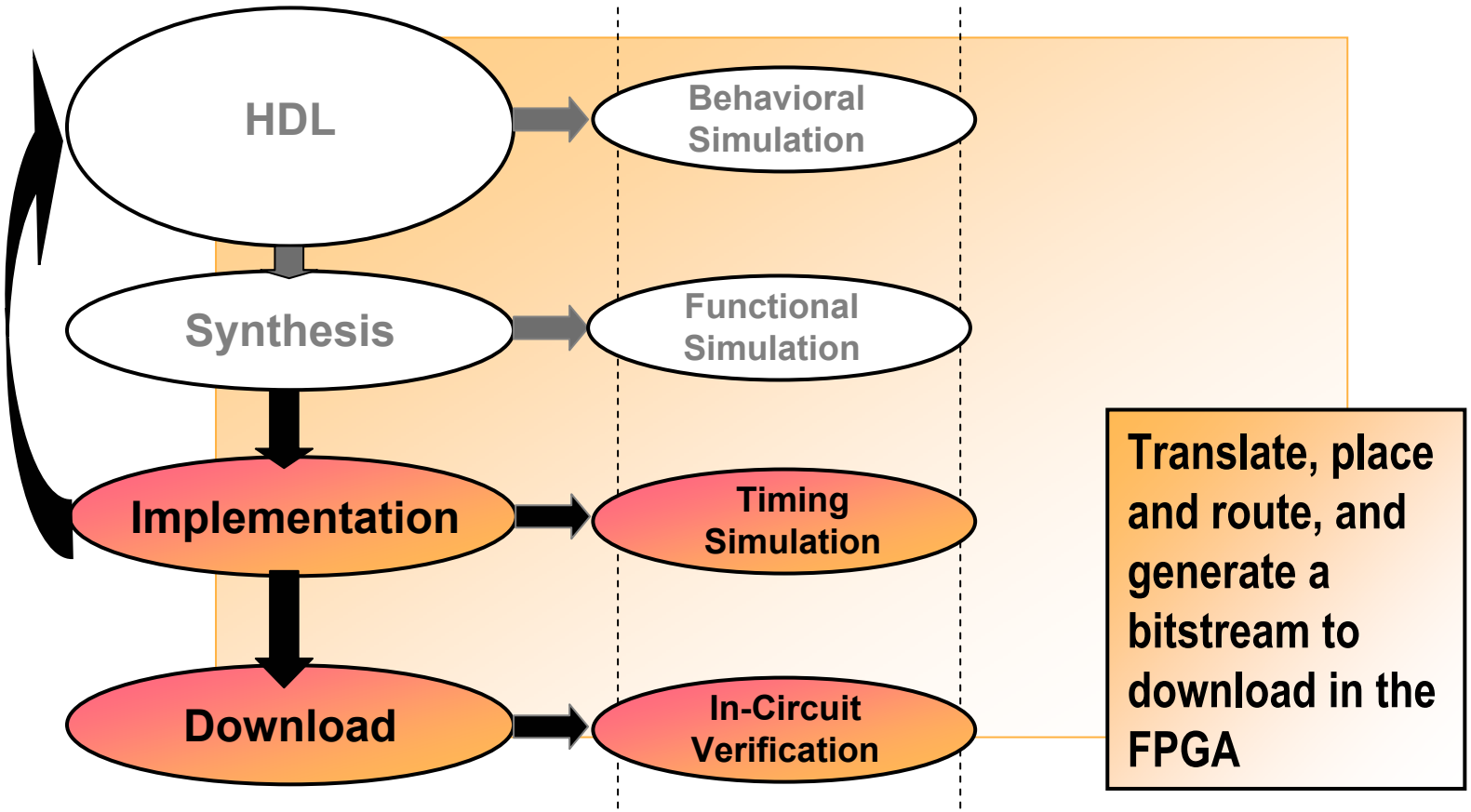
Traditional design flow 1/3



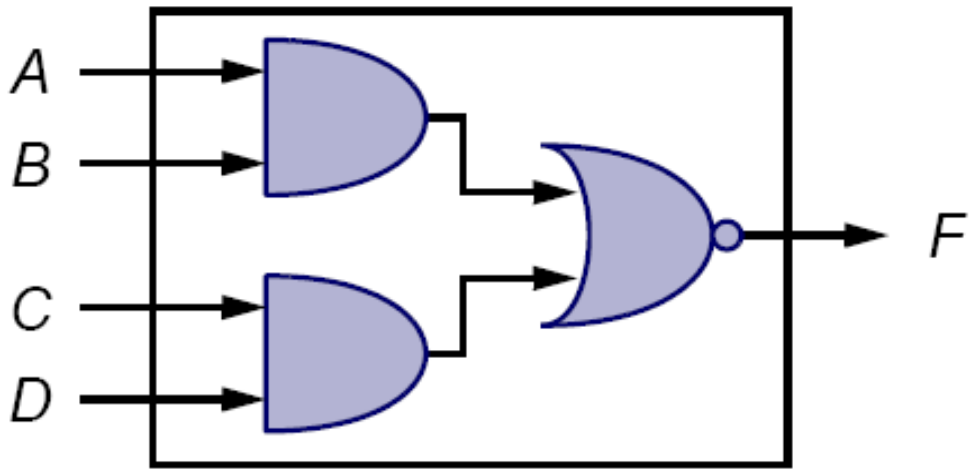
Traditional design flow 2/3



Traditional design flow 3/3



VHDL 101



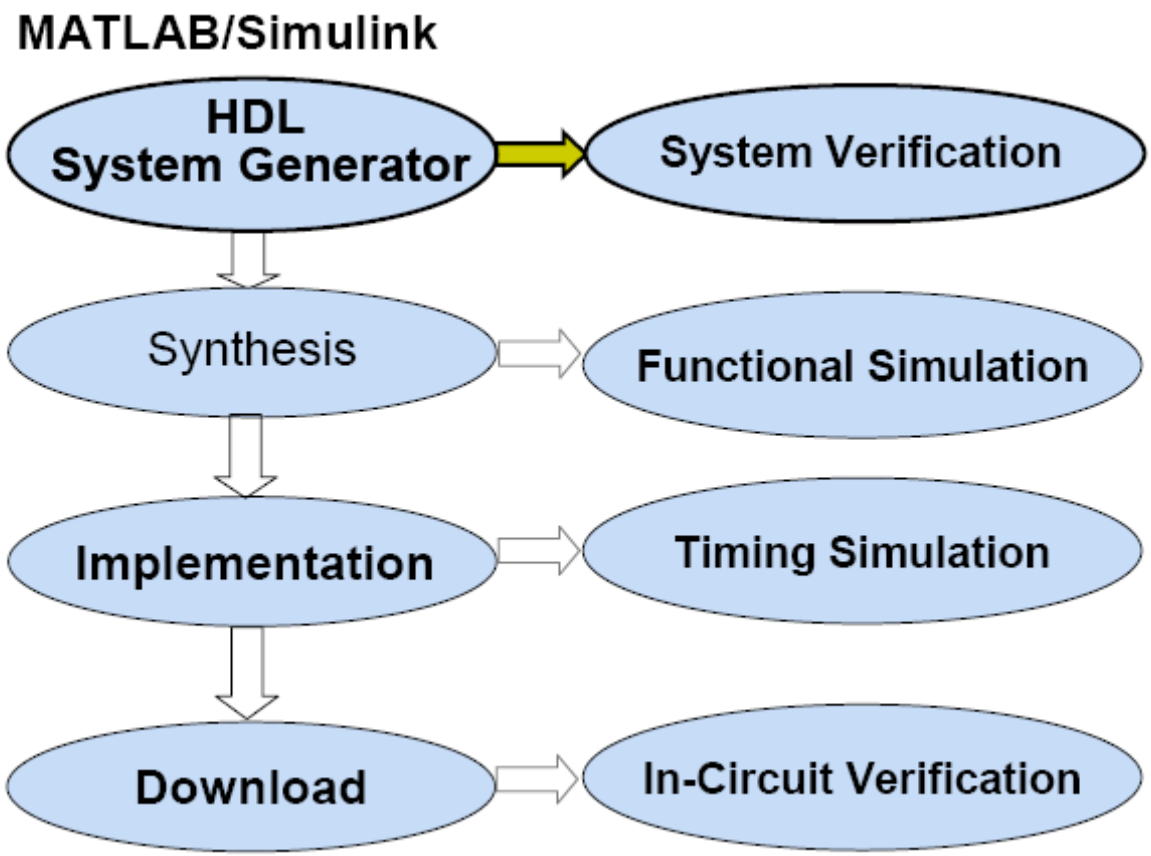
Boolean Expression: $F = \overline{A \cdot B + C \cdot D}$

VHDL Code 1:
```
F <= not((A and B) or (C and D))
```

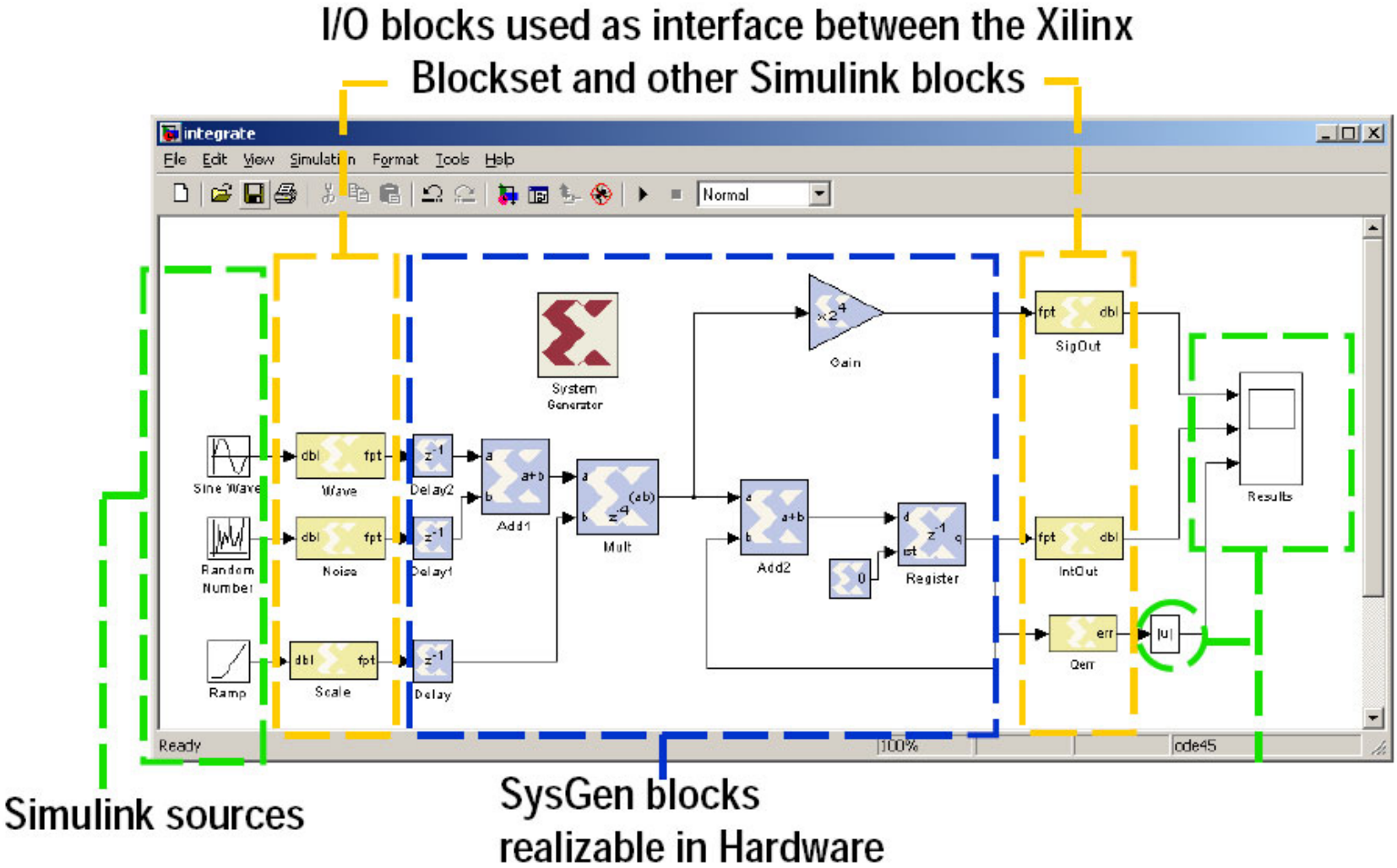
VHDL Code 2:
```
F <= (A and B) nor (C and D)
```

Both VHDL code segments produce exactly the same hardware.

Simulink design flow



Simulink design flow

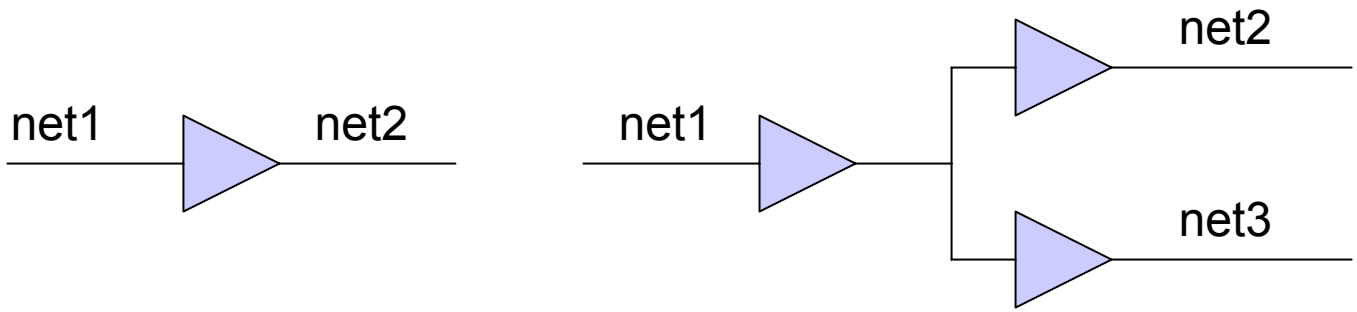


Outline

- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

Buffering

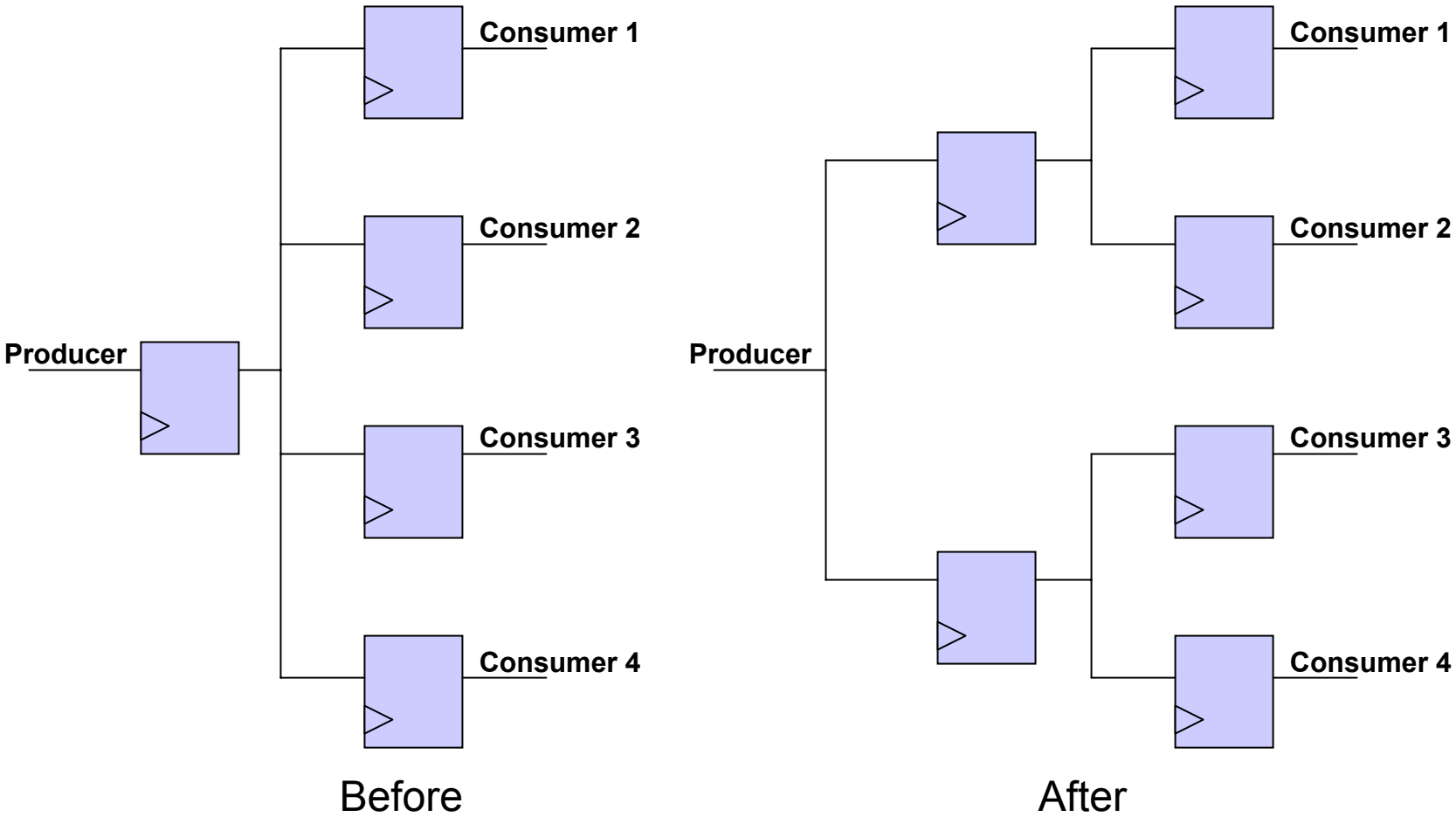
- Delay in modern designs can be as much as 90% routing, 10% logic. Routing delay is due to long nets + capacitive input loading.
- Buffering is done automatically by most synthesis tools and reduces the fan out on affected nets:



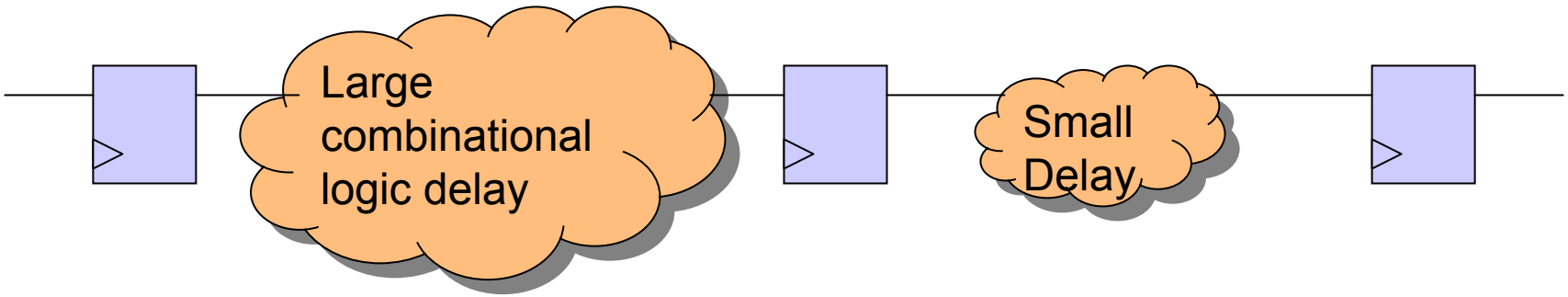
Before buffering

After buffering

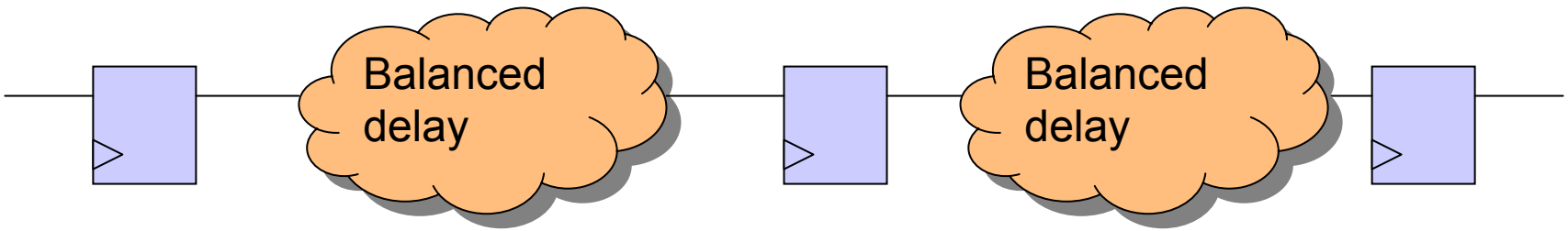
Replicating registers (and associated logic if necessary)



Retiming (a.k.a. register balancing)

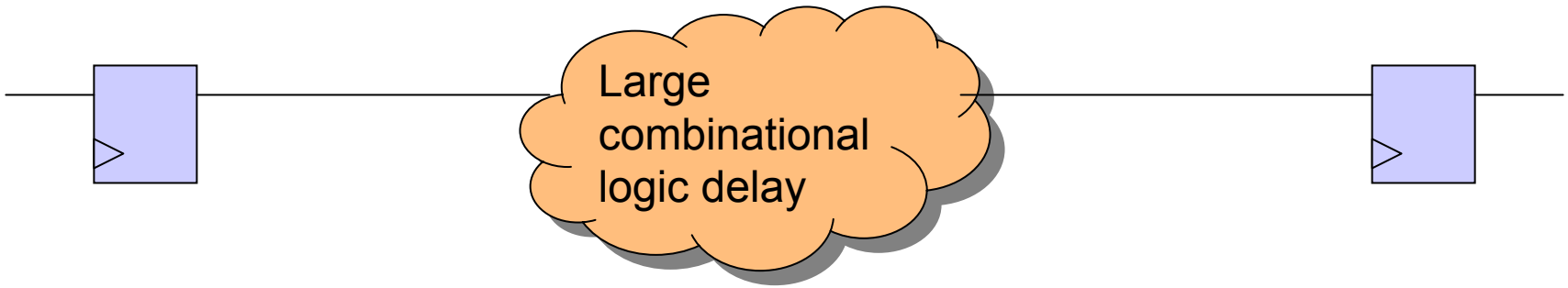


Before



After

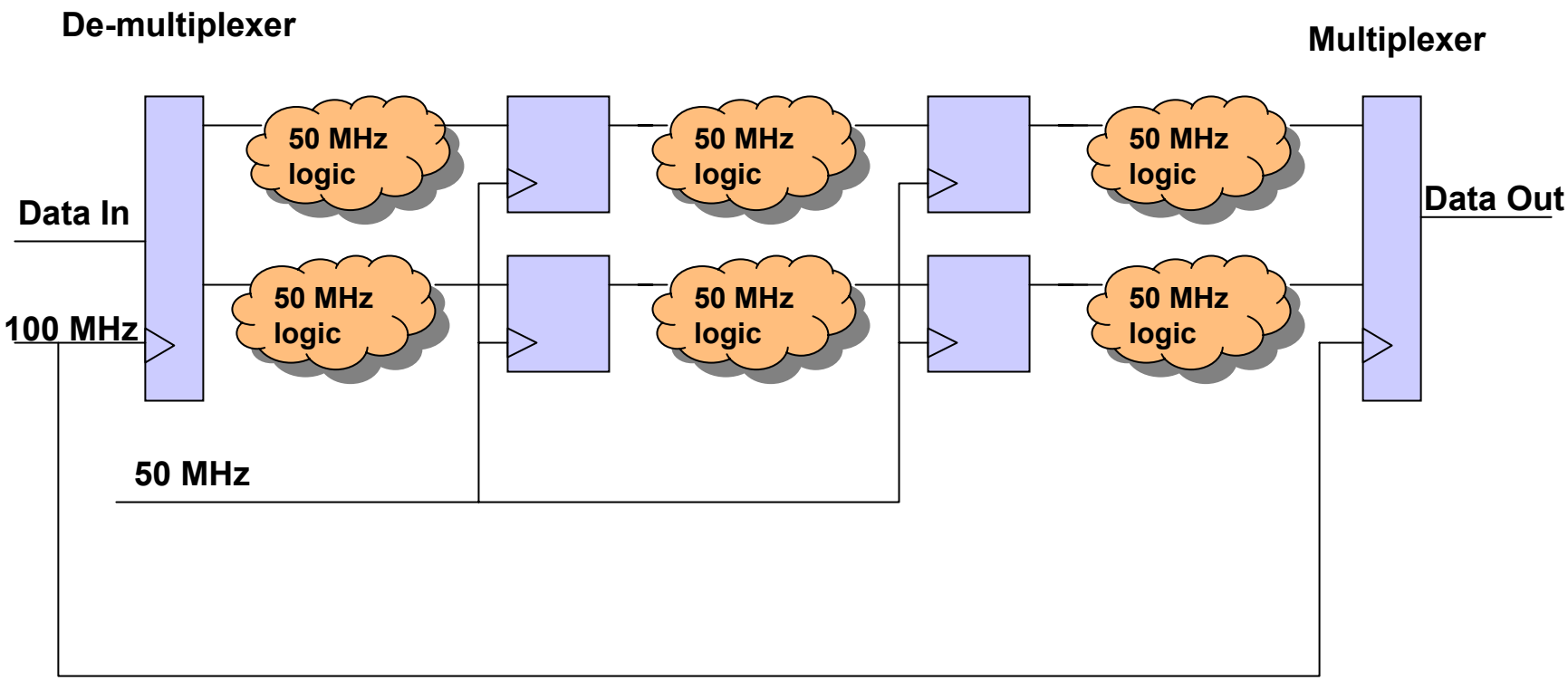
Pipelining



Before

After

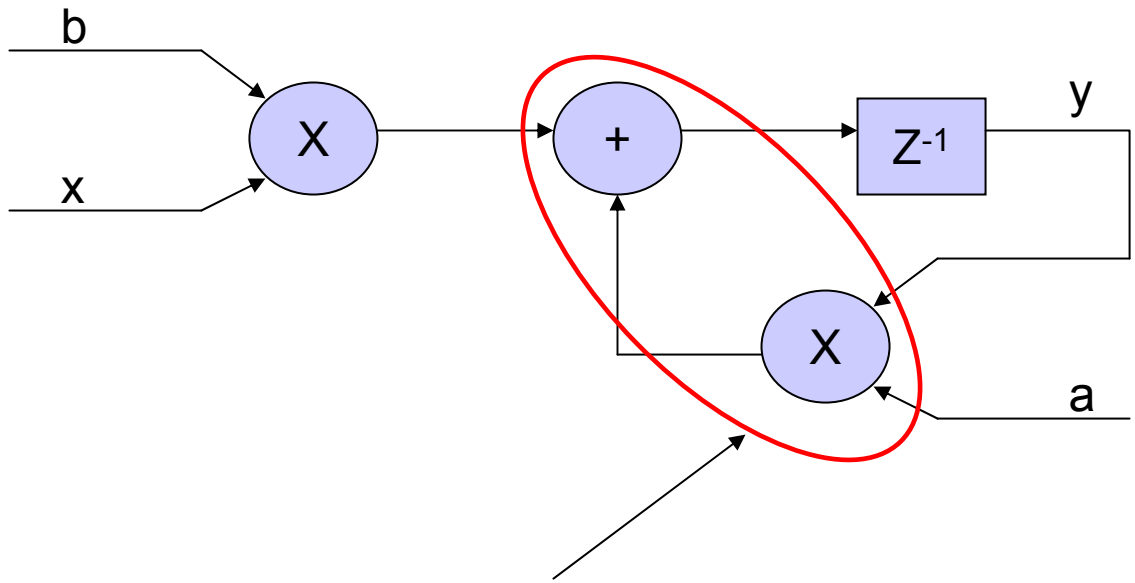
Time multiplexing



An example: boosting the performance of an IIR filter (1/2)

Simple first order IIR: y[n+1] = ay[n] + b x[n]

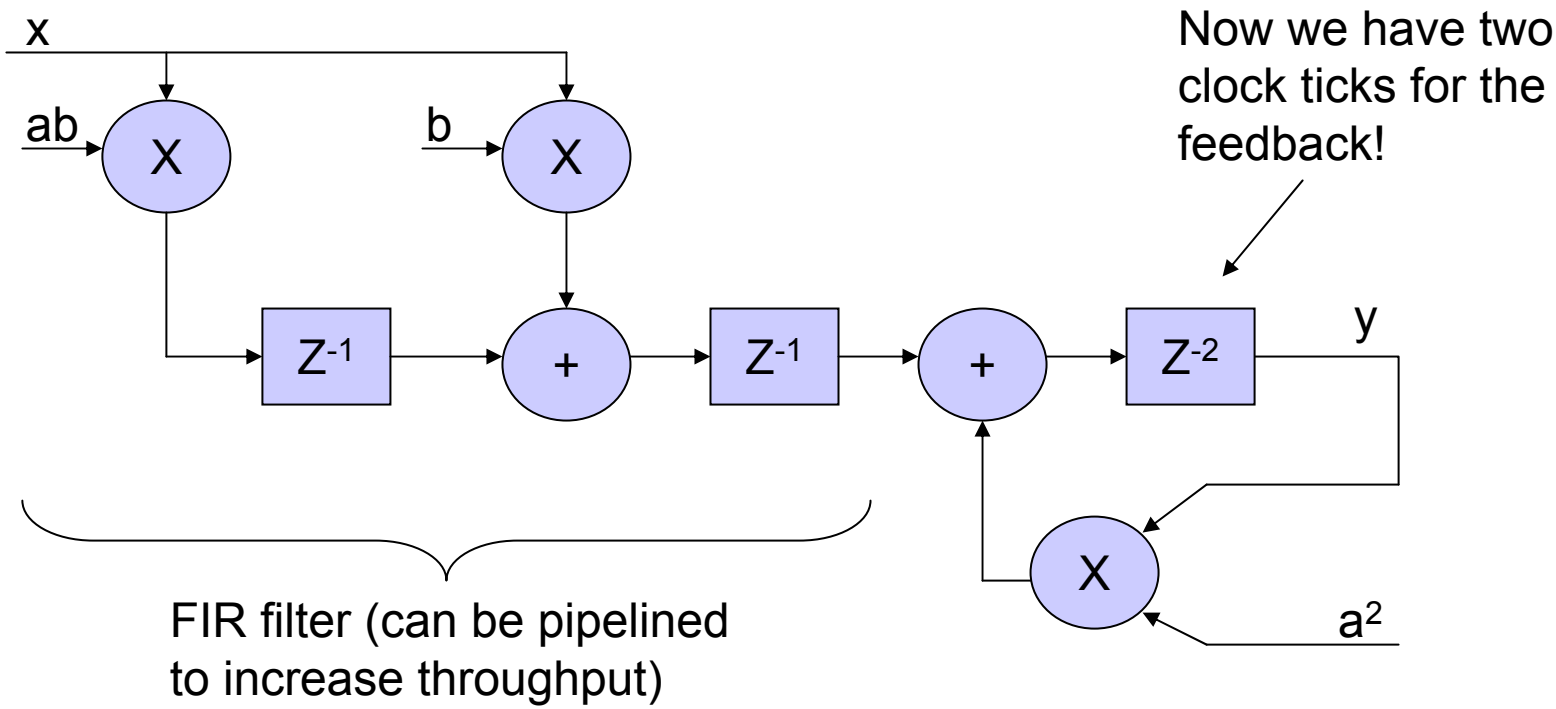
Problem found in the phase filter of a PLL used to track bunch frequency in CERN's PS



Performance bottleneck in the feedback path

An example: boosting the performance of an IIR filter (2/2)

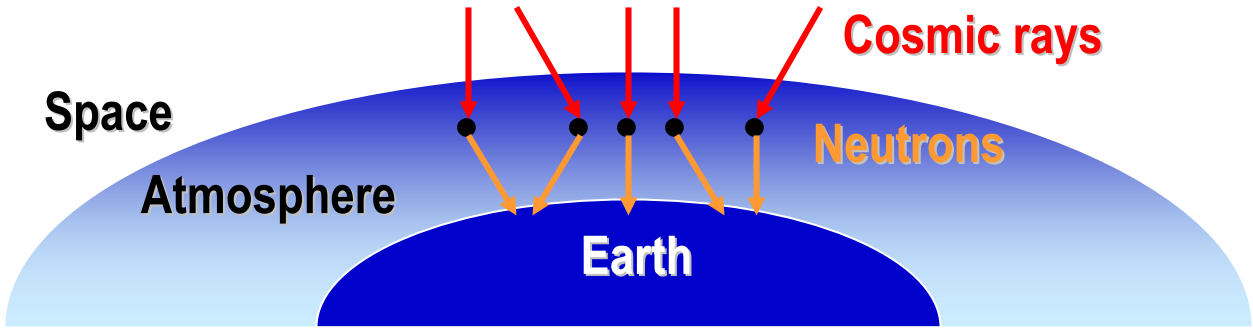
Look ahead scheme:
From $y[n+1] = ay[n] + b\,x[n]$ we get
$y[n+2] = ay[n+1] + bx[n+1] = a^2y[n] + abx[n] + bx[n+1]$

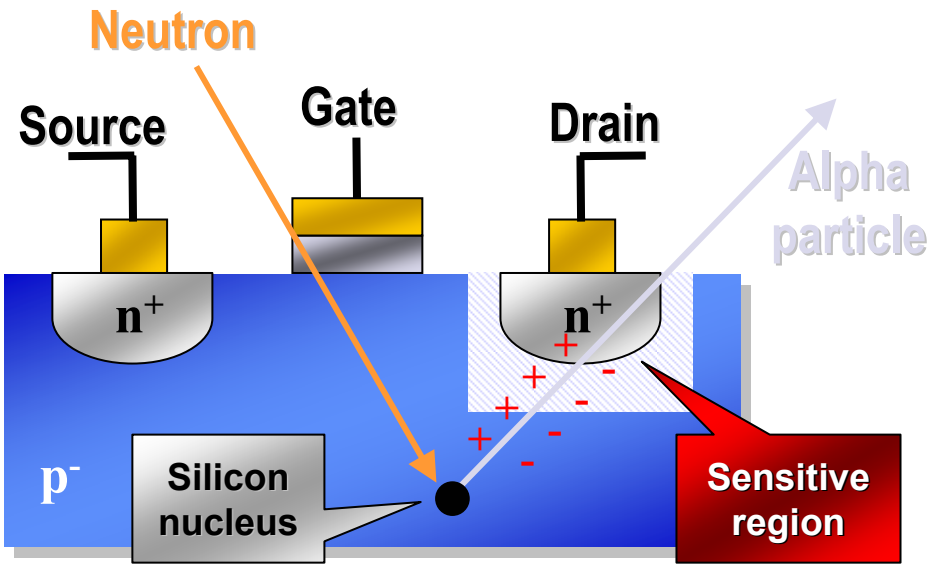


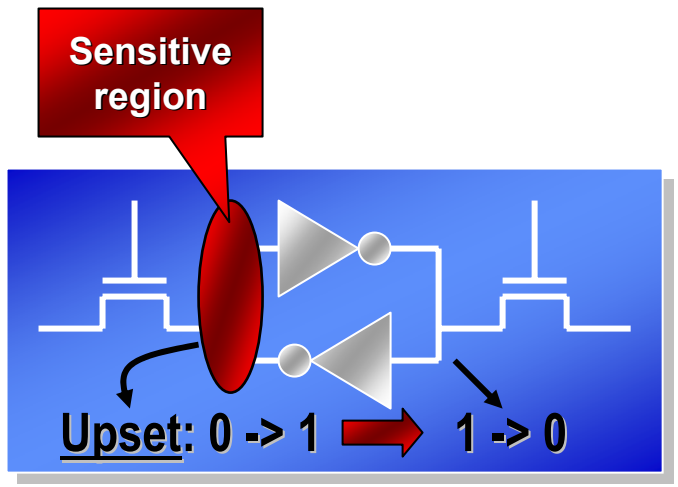
Outline

- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

Single Event Effects (SEE) created by neutrons







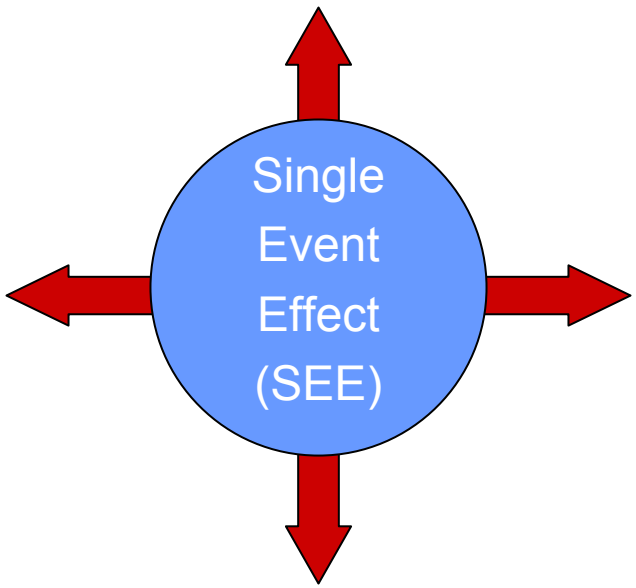
Memory Cell: CMOS Configuration Latch (CCL)

Classification of SEEs

Single Event Transient

(**SET**)

A signal briefly fluctuates somewhere in design



Single Event Latch-Up

(**SEL**)

Parasitic transistors activated in a device, causing internal short

Single Event Upset

(**SEU**)

Bit-Flip Somewhere

Single Event Functional Interrupt

(**SEFI**)

Bit-Flip specifically in a control register – POWER ON RESET/JTAG etc.

SEU Failures in Time (FIT)

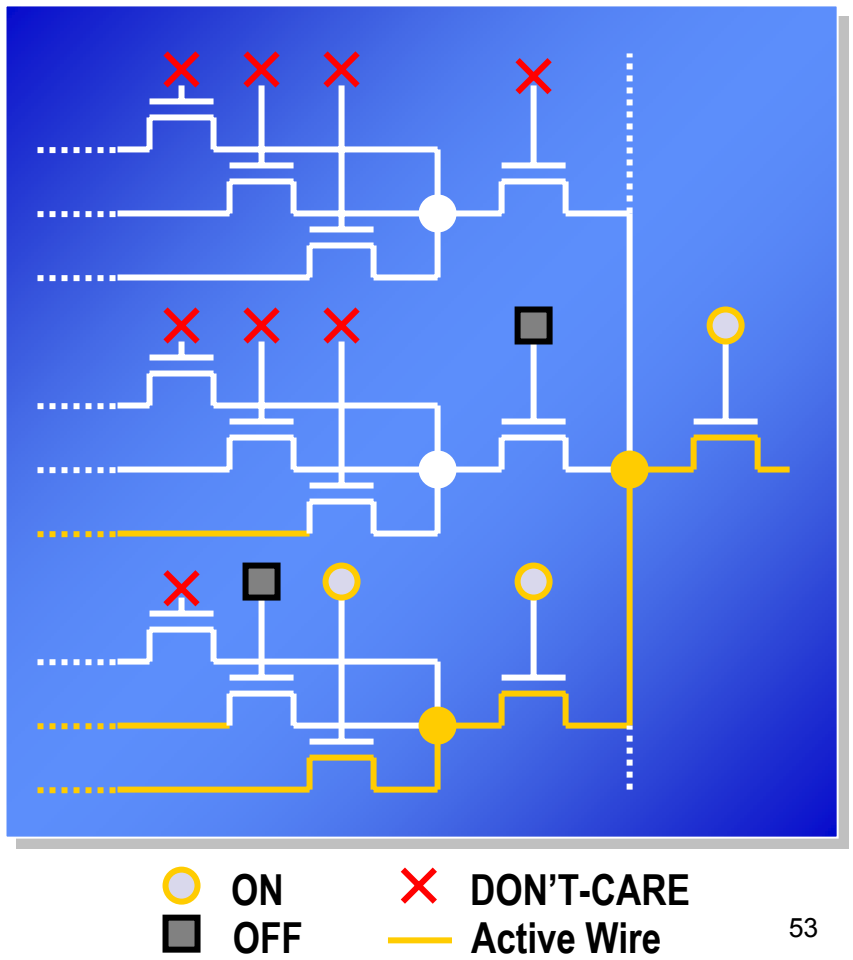
- Defined as the number of failures expected in 10^9 hours.
- In practice, configuration RAM dominates. Example:

Virtex XCV1000 memory Utilization

Memory Type	# of bits	%
Configuration	5,810,048	97.4
Block RAM	131,072	2.2
CLB flip-flops	26,112	0.4

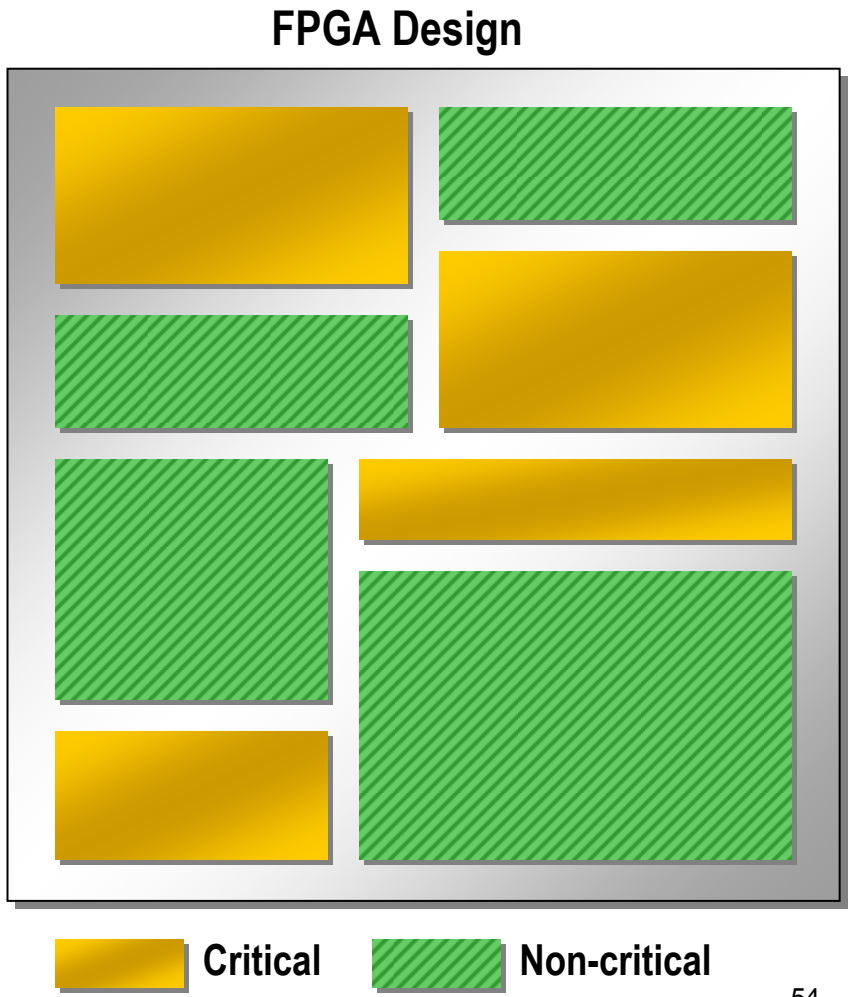
- Average of only 10% of FPGA configuration bits are used in typical designs
  - Even in a 99% full design, only up to 30% are used
  - Most bits control interconnect muxes
  - Most mux control values are “don’t-care”
- Must include this ratio for accurate SEU FIT rate calculations.

FPGA Interconnect

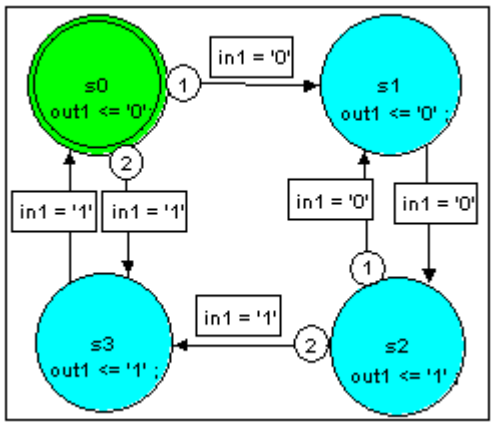


Not all parts of the design are critical

- Average of only 40% of circuits in FPGA designs are critical
  - Substantial circuit overhead for startup logic, diagnostics, debug, monitoring, fault-handling, control path, etc.
- Must also include this ratio in SEU FIT rate calculations



Safe state machines



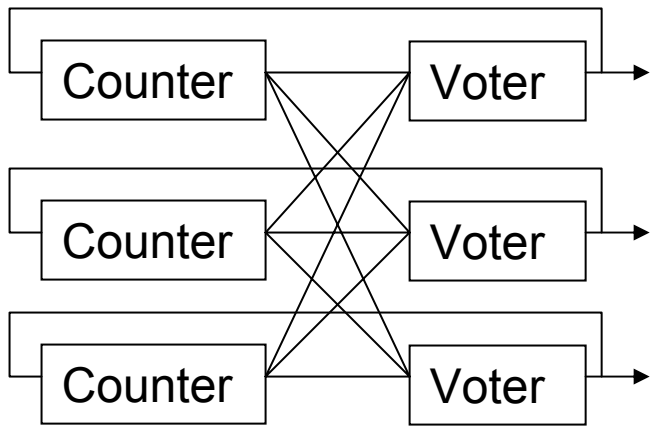
One-hot encoding:
s0 => 0001
s1 => 0010
s2 => 0100
s3 => 1000
12 “illegal states” not covered, or covered with a “when others” in VHDL or equivalent.
→ Use option in synthesis tool to prevent optimization of illegal states.

Mitigation techniques: scrubbing

- Readback and verification of configuration.
  - Most internal logic can be verified during normal operation.
  - Sets limits on duration of upsets.
- Partial configuration
  - Not supported by all FPGA vendors/families.
  - Allows fine grained reconfiguration.
  - Does not reset entire device.
    - Allows user logic to continue to function.
- Complete reconfiguration
  - Required after SEFI.
  - No user functionality for the duration of reconfiguration.
- Verification by dedicated device
  - Usually radiation tolerant antifuse FPGA
  - Secure storage of checksums and configuration an issue
    - FLASH is radiation sensitive
- Self verification
  - Often the only option for existing designs
  - Not possible in all device families
    - Utilizes logic intended for dynamic reconfiguration
    - Verification logic has small footprint
      - Usually a few dozen CLBs and 1 block RAM (for checksums).

Triple Module Redundancy (TMR)

- Feedback TMR
  - Three copies of user logic
  - State feedback from voter
    - Counter example
  - Handles faults
  - Resynchronizes
    - Operational through repair
  - Speed penalty due to feedback
  - Desirable for state based logic



Alternatives

- Antifuse
  - Configuration based on physical shorts
    - Invulnerable to upset
    - Cannot be altered
  - Over 90% smaller upset cross section for comparable geometry
  - Signal routing more efficient
    - Much lower power dissipation for similar device geometry
  - Lags SRAM in fabrication technology
    - Usually one generation behind
    - Latch up more of a problem than in SRAM devices
- Rad-hard Antifuse
  - All flip-flops TMRed in silicon
    - Unmatched reliability
    - High (extreme) cost
    - Unimpressive performance
      - Feedback TMR built in
      - Usually larger geometry
      - Not available in highest densities offered by antifuse
- FLASH FPGAs
  - Middle ground in base susceptibility
  - Readback/Verification problematic
    - Usually only JTAG (slow) supported
  - Maximum number of write cycles an issue

Outline

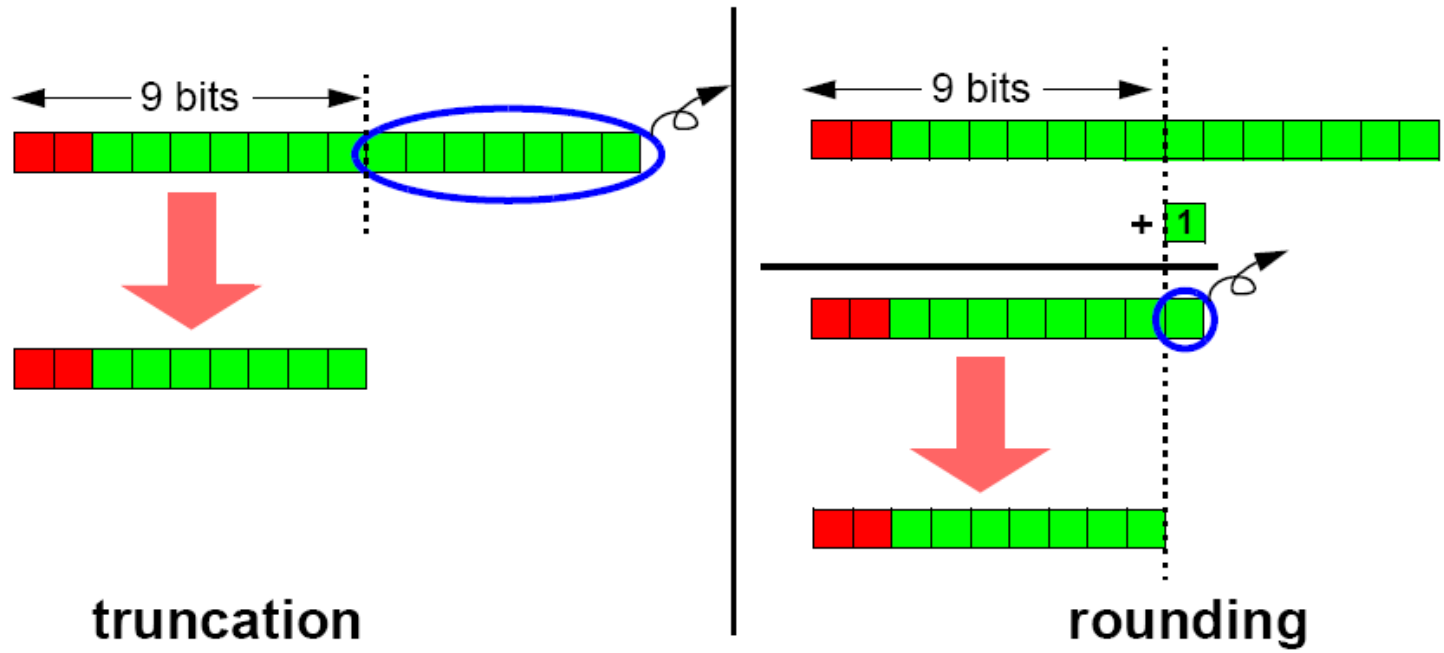
- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

Fixed point (2's complement) binary numbers

digit worth								decimal value
$-(2^2)$	2^1	2^0 •	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	
-4	**2**	**1** •	**0.5**	**0.25**	**0.125**	**0.0625**	**0.03125**	
0	0	0 •	0	0	0	0	1	0.03125
0	0	0 •	0	0	0	1	0	0.0625
1	0	1 •	0	0	0	0	0	-3.0
1	1	0 •	0	0	1	1	1	-1.78125
1	1	1 •	1	1	1	1	1	-0.03125

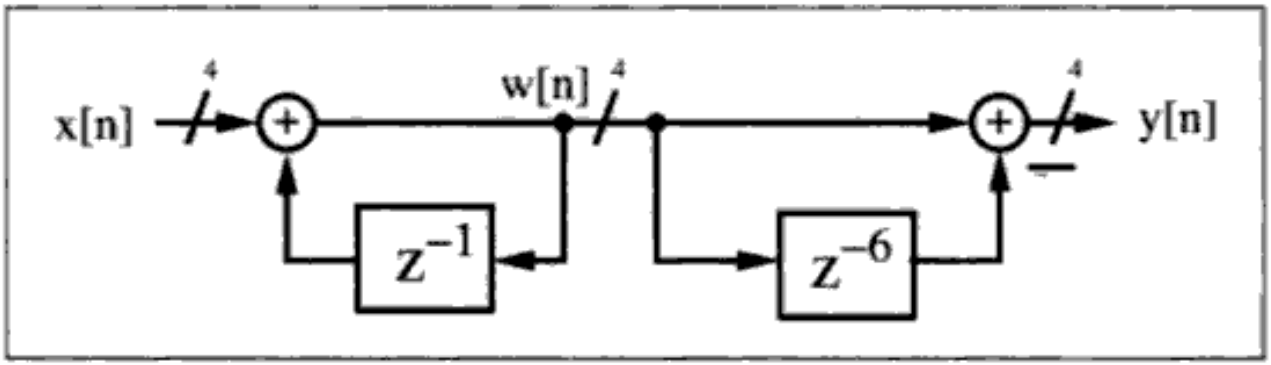
Example: 3 integer bits and 5 fractional bits

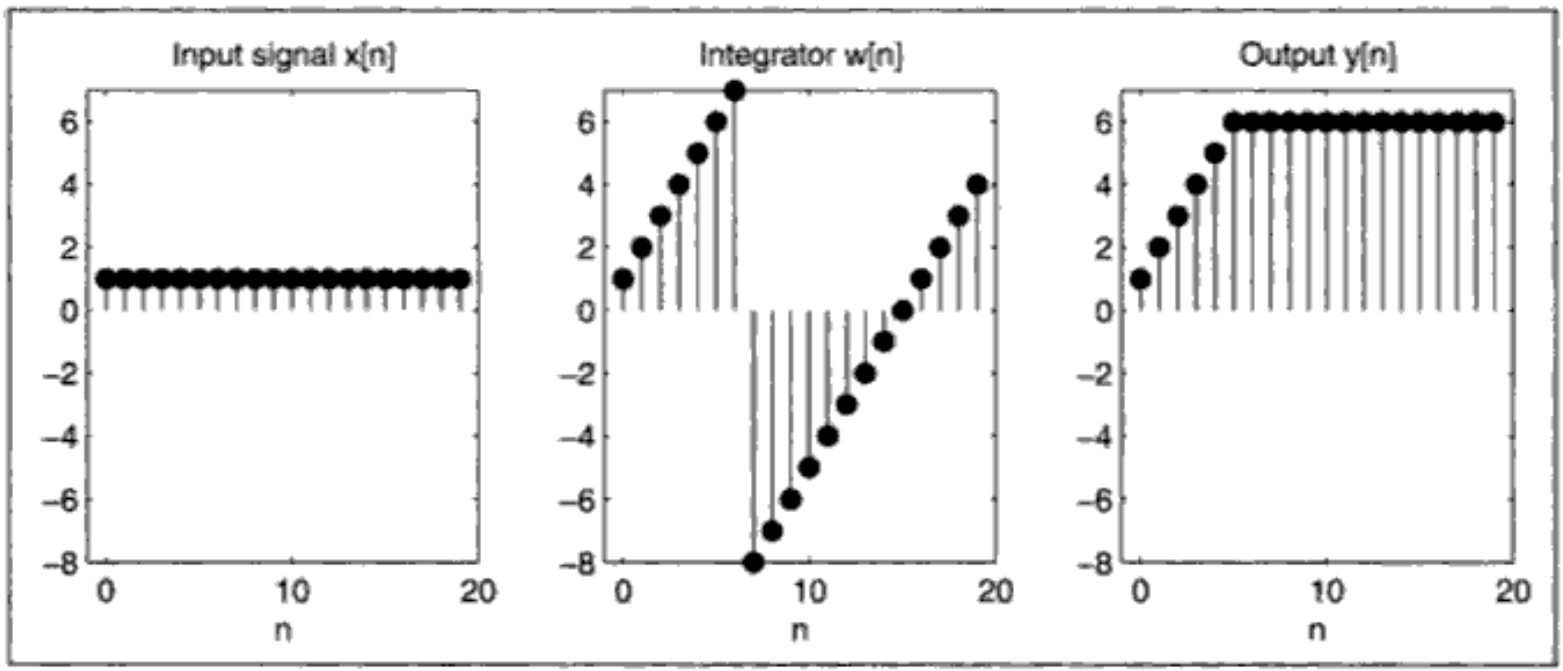
Fixed point truncation vs. rounding



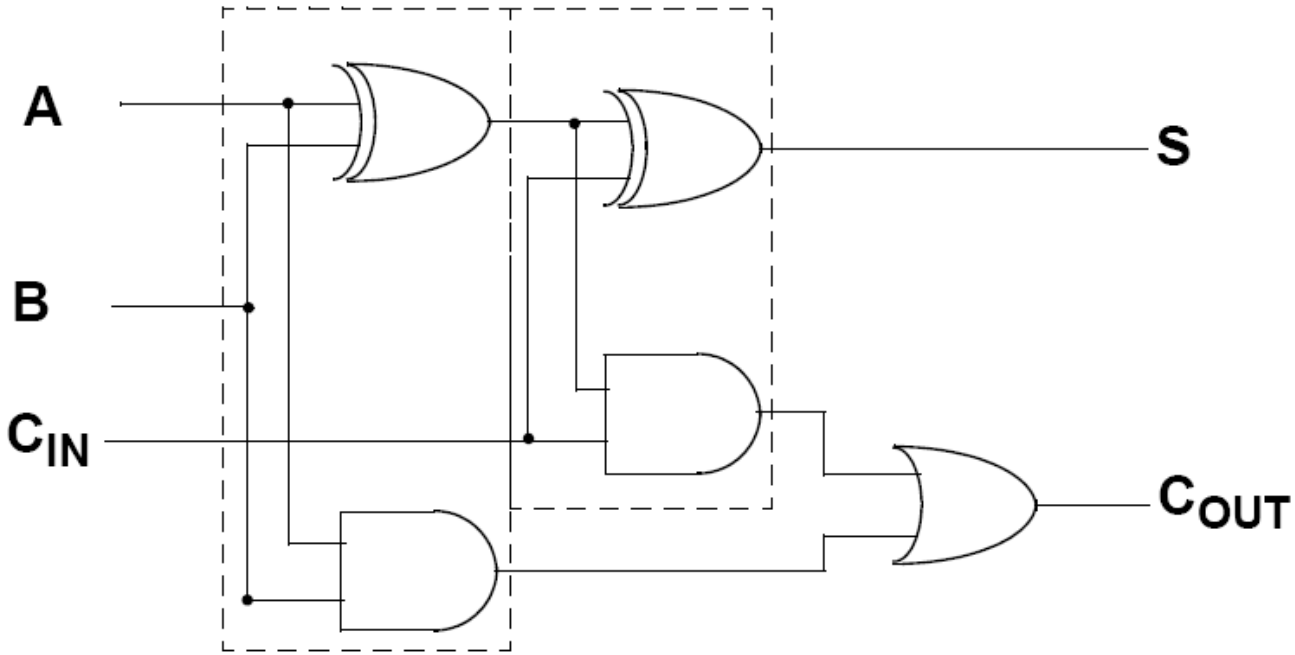
Note that in 2's complement, truncation is biased while rounding isn't.

Interesting property of 2's complement numbers

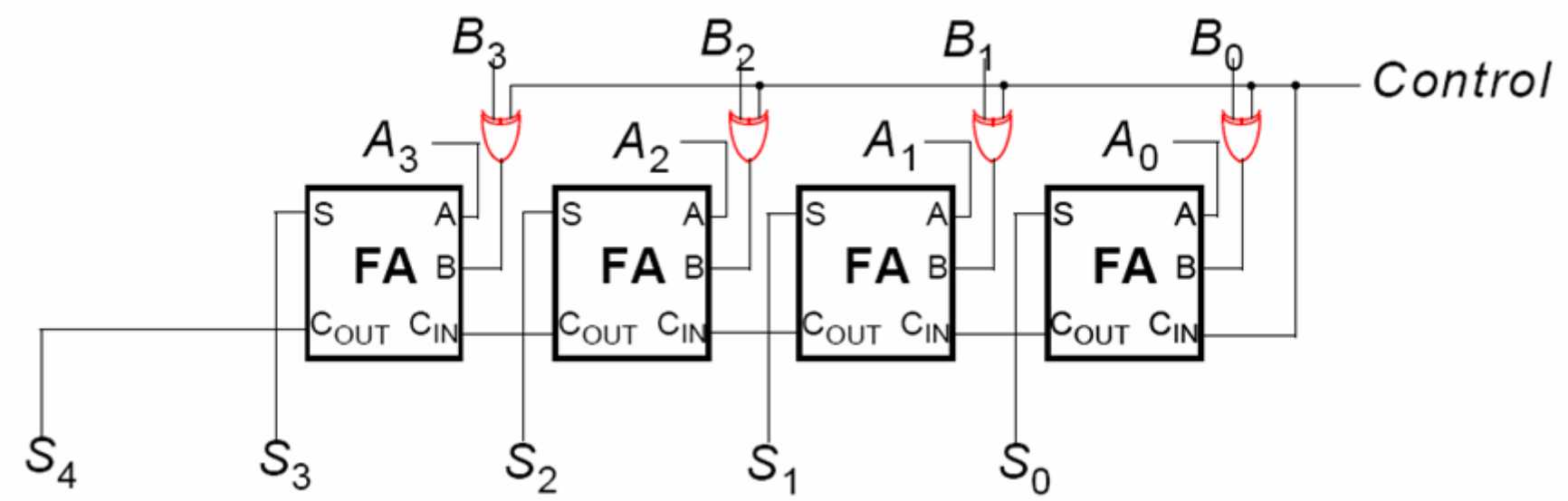




The Full Adder (FA)

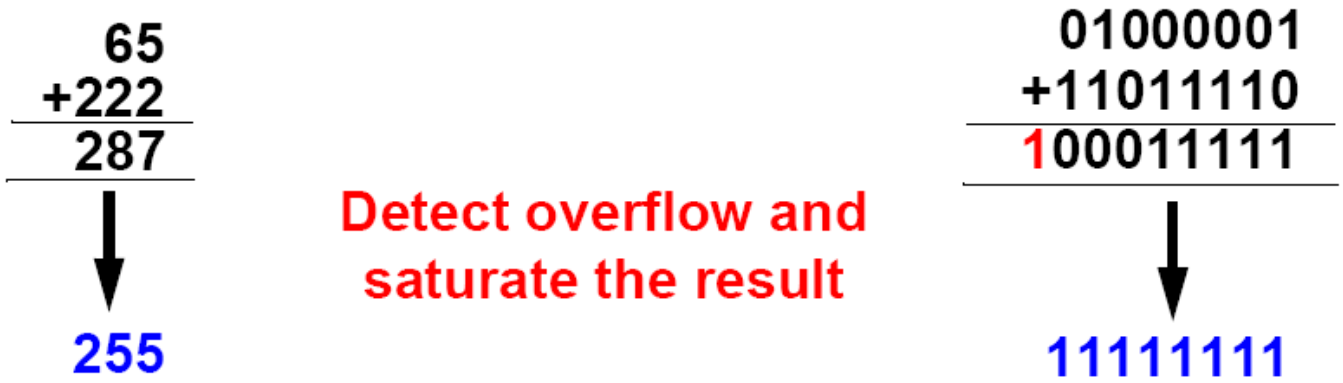


Add/subtract circuit



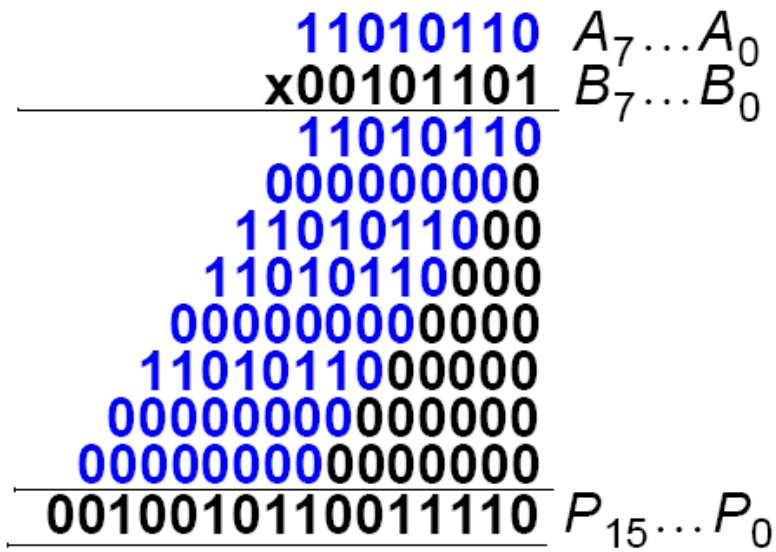
S = A+B when Control='0'
S = A-B when Control='1'

Saturation

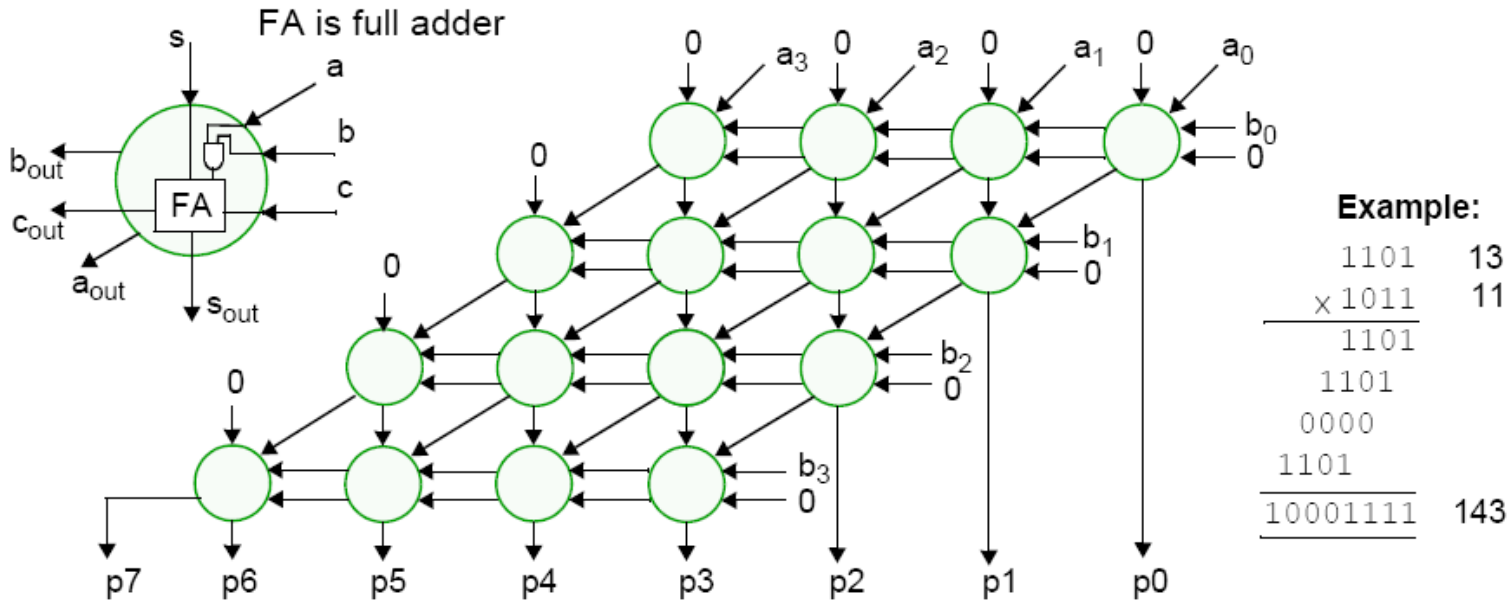


You can’t let the data path become arbitrarily wide. Saturation involves overflow detection and a multiplexer. Useful in accumulators (like the one in a PI controller).

Multiplication: pencil & paper approach

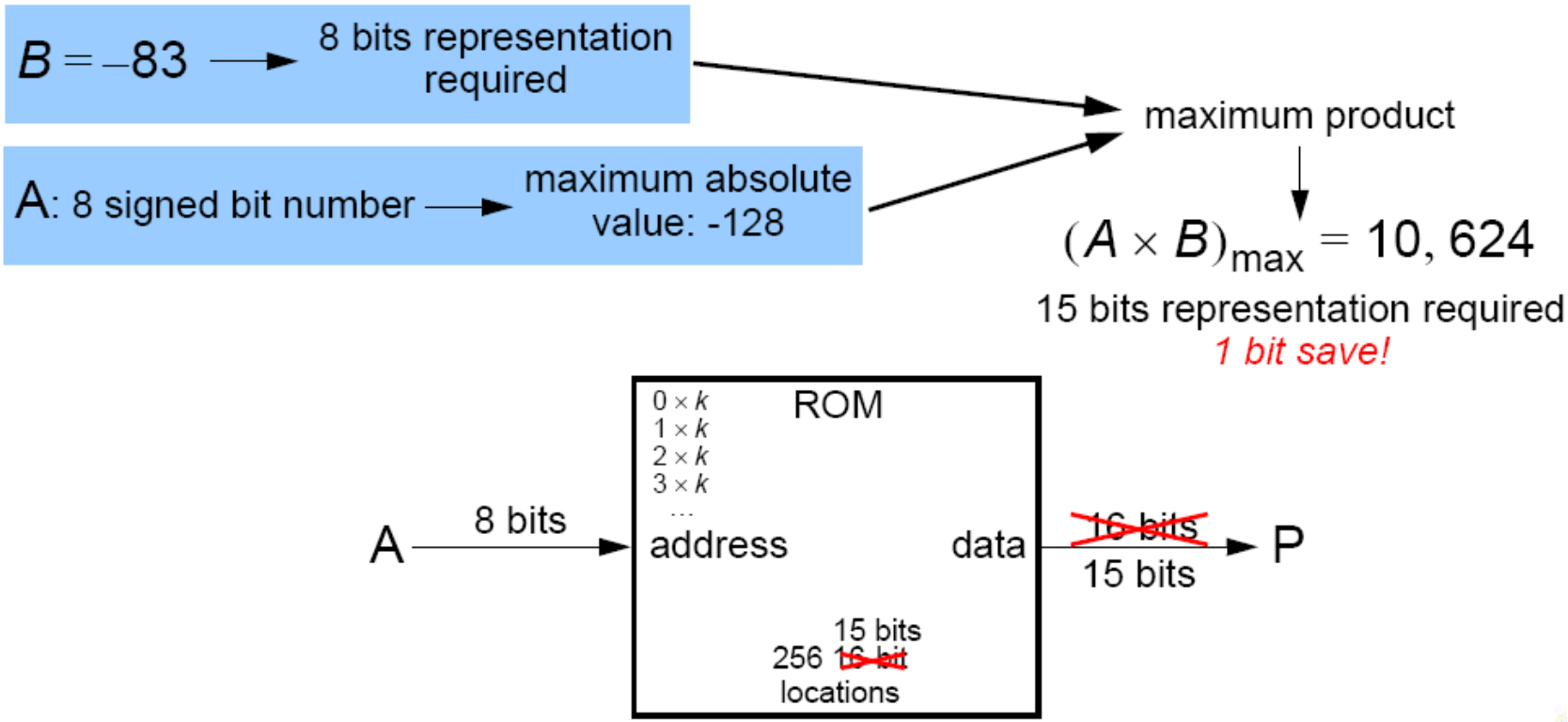


A 4-bit unsigned multiplier using Full Adders and AND gates



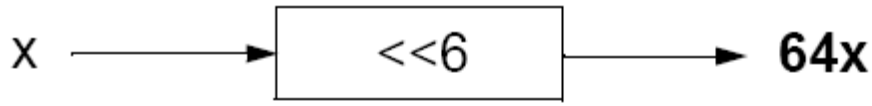
Of course, you can use embedded multipliers if your chip has them!

Constant coefficient multipliers using ROM

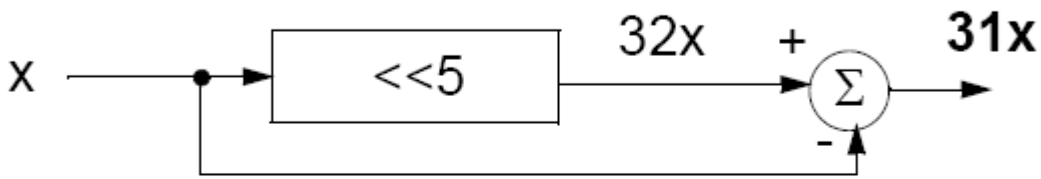


For “easy” coefficients, there are smarter ways. E.g. to multiply a number A by 31, left-shift A by 5 places then subtract A.

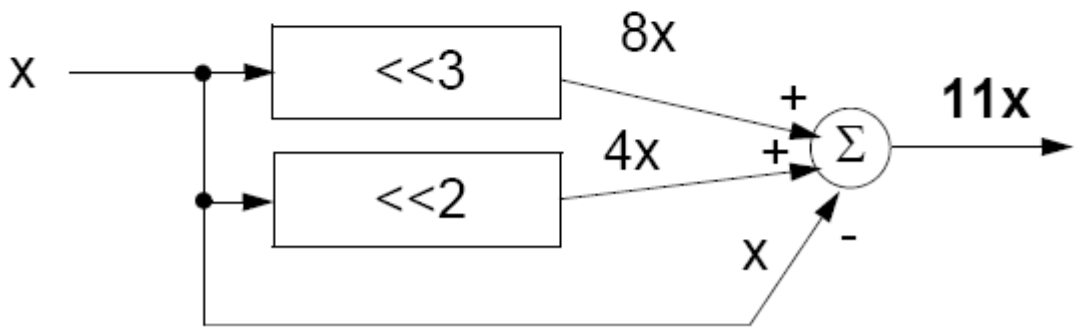
Smart multiplier examples



Multiply by 64

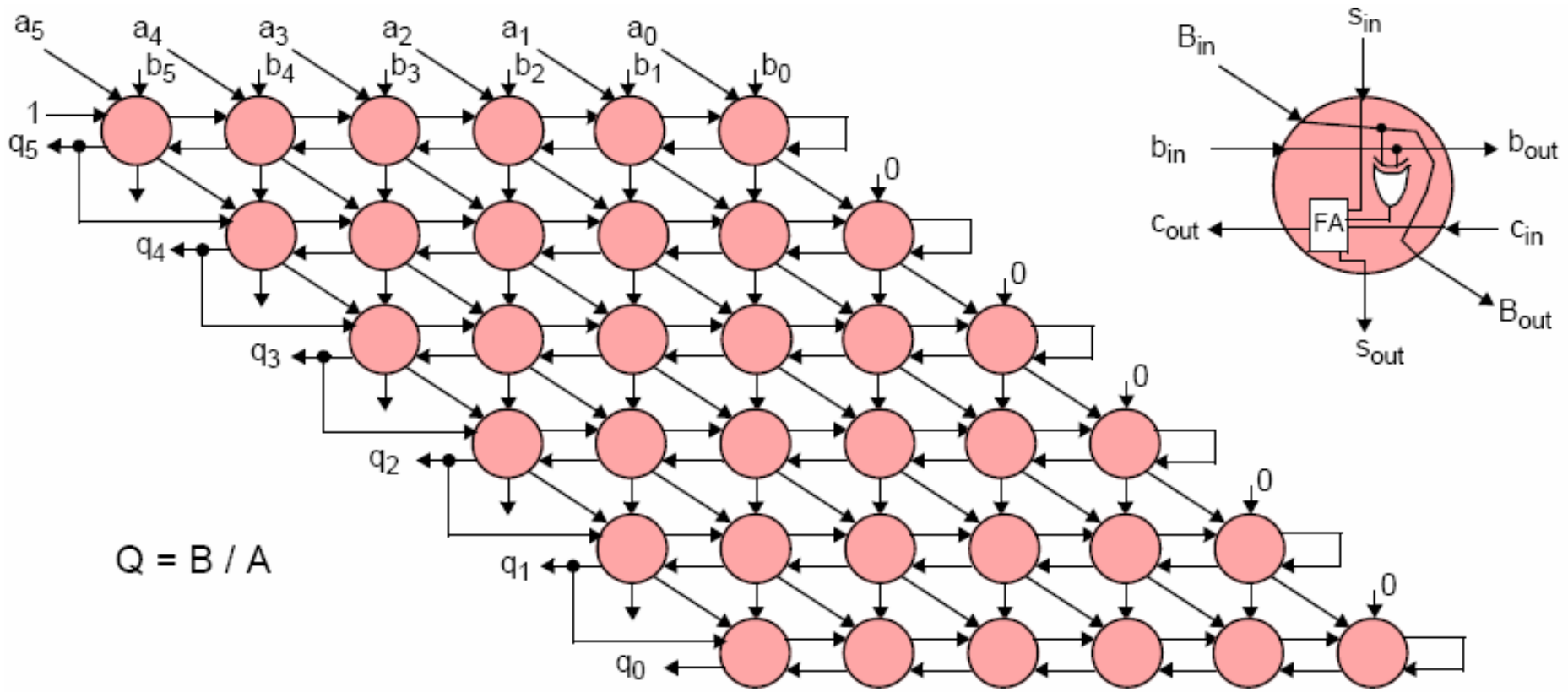


Multiply by 31



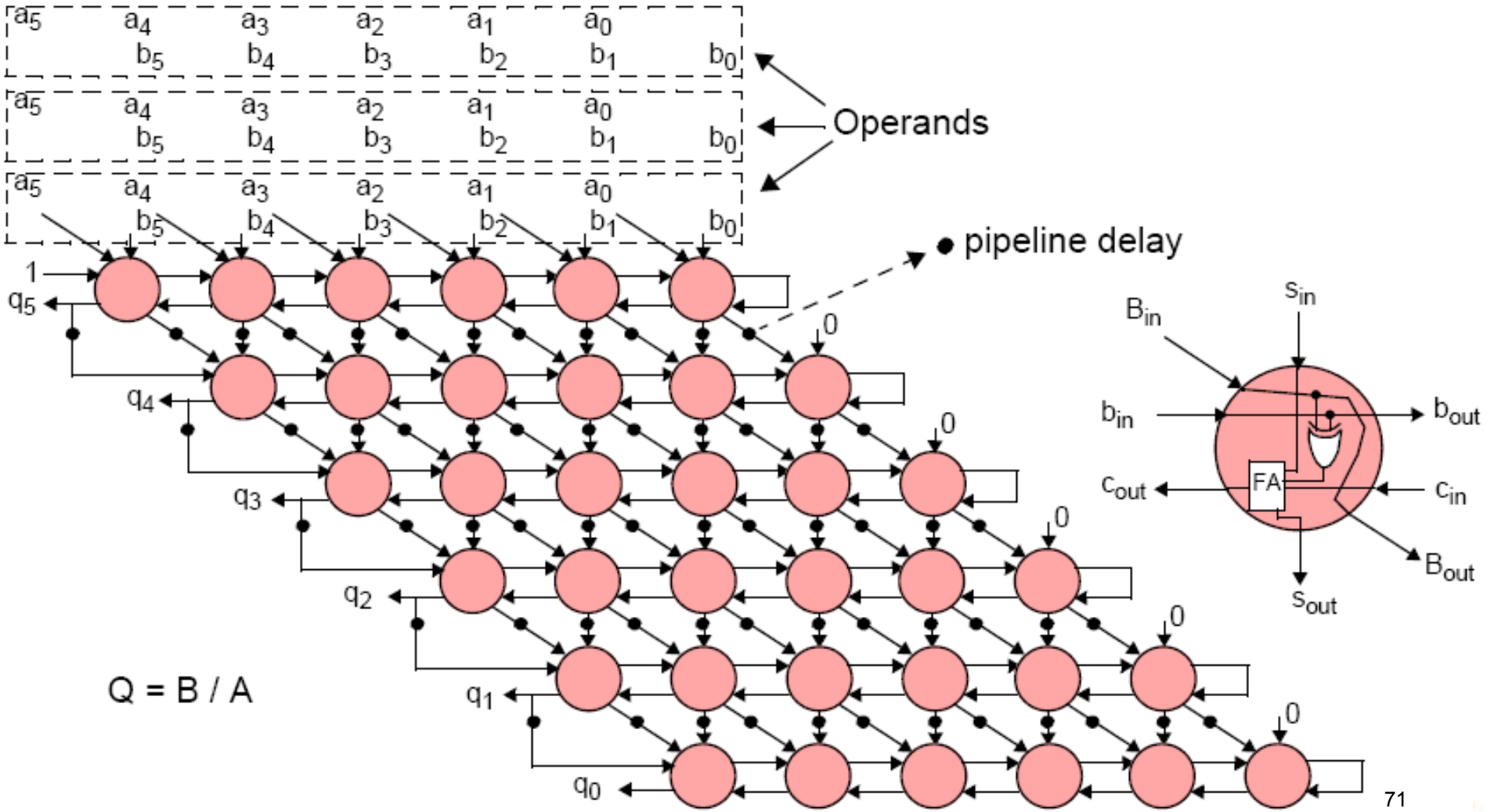
Multiply by 11

Division: pencil & paper

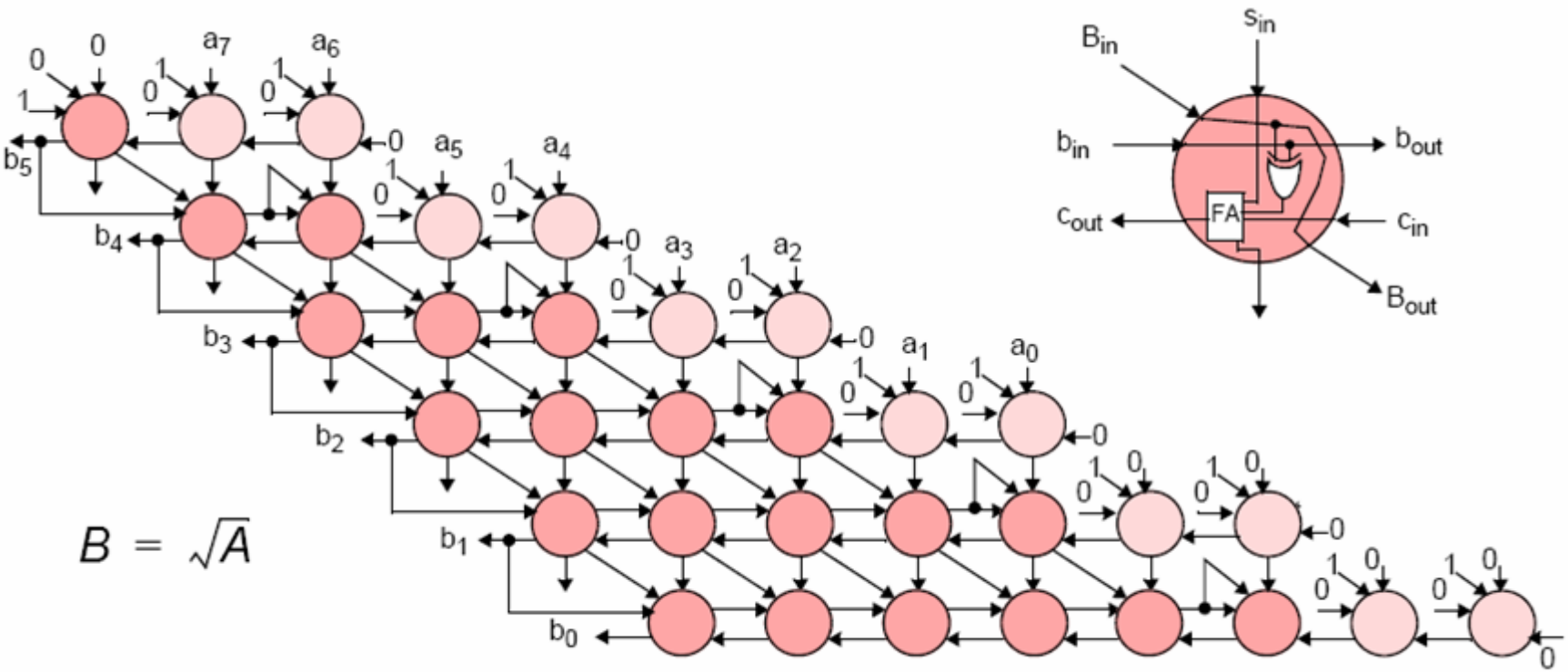


- Uses add/subtract blocks presented earlier.
- MSB produced first: this will usually imply we have to wait for whole operation to finish before feeding result to another block.
- Longer combinational delays than in multiplication: an N by N division will always take longer than an N by N multiplication.

Pipelining the division array



Square root



- Take a division array, cut it in half (diagonally) and you have square root. Square root is therefore faster than division!
- Although with less ripple through, this block suffers from the same problems as the division array.
- Alternative approach: first guess with a ROM, then use an iterative algorithm such as Newton-Raphson.

Distributed Arithmetic (DA) 1/2

Digital filtering is about sums of products:

$$y = \sum_{n=0}^{N-1} c[n] \cdot x[n]$$

Let's assume:
- c[n] constant (prerequisite to use DA)
- x[n] input signal B bits wide

Then:
$$y = \sum_{n=0}^{N-1} \left(c[n] \cdot \sum_{b=0}^{B-1} x_b[n] \cdot 2^b \right)$$

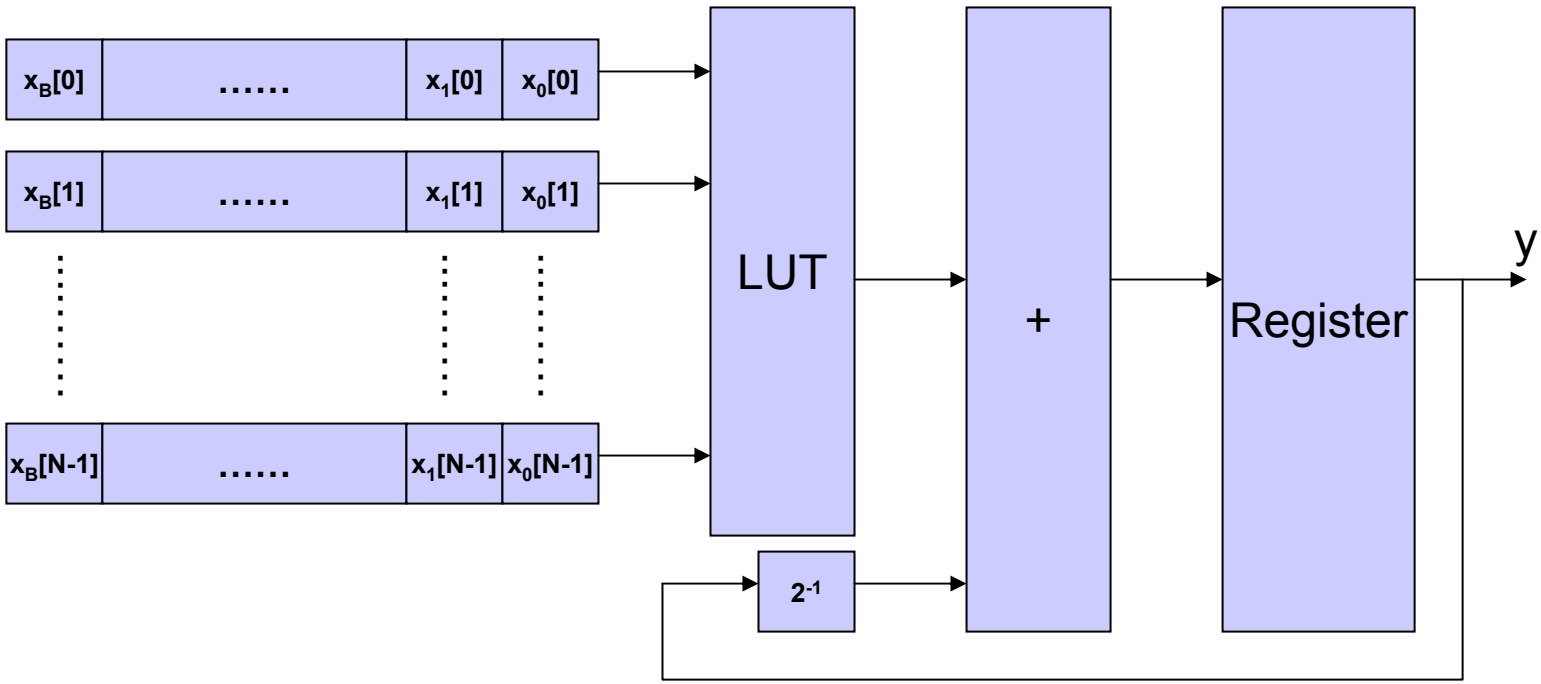
$x_b[n]$ is bit number b of x[n] (either 0 or 1)

And after some rearrangement of terms:
$$y = \sum_{b=0}^{B-1} 2^b \cdot \underbrace{\left(\sum_{n=0}^{N-1} c[n] \cdot x_b[n] \right)}$$

This can be implemented with an N-input LUT

Distributed Arithmetic (DA) 2/2

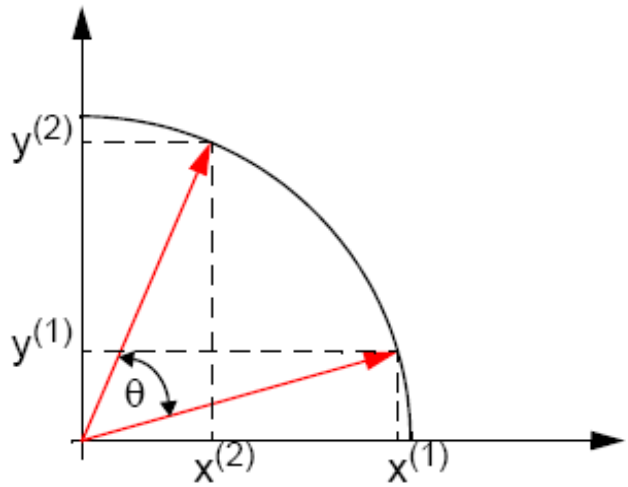
$$y = \sum_{b=0}^{B-1} 2^b \cdot \left(\sum_{n=0}^{N-1} c[n] \cdot x_b[n] \right)$$



Generates a result every B clock ticks. Replicating logic one can trade off speed vs. area, to the limit of getting one result per clock tick.

COordinate Rotation DIgital Computer

- The CORDIC method is based on the rotation of a vector from position $(x^{(1)}, y^{(1)})$ to $(x^{(2)}, y^{(2)})$:



- The new position can be calculated using the Givens rotation:

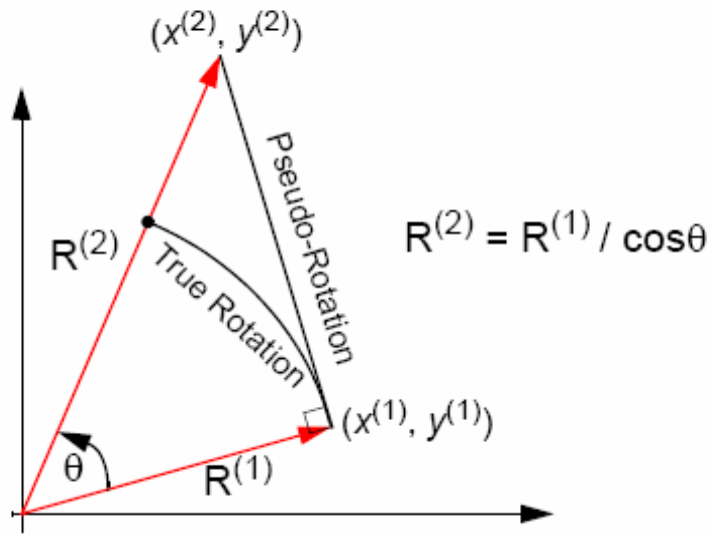
$$x^{(2)} = x^{(1)}\cos\theta - y^{(1)}\sin\theta = \cos\theta(x^{(1)} - y^{(1)}\tan\theta)$$

$$y^{(2)} = x^{(1)}\sin\theta + y^{(1)}\cos\theta = \cos\theta(y^{(1)} + x^{(1)}\tan\theta)$$

Pseudo-rotations

- By removing the $\cos\theta$ term, the equations give the result of a Pseudo-Rotation:

$$x^{(2)} = x^{(1)} - y^{(1)}\tan\theta$$
$$y^{(2)} = y^{(1)} + x^{(1)}\tan\theta$$



$R^{(2)} = R^{(1)} / \cos\theta$

Basic CORDIC iterations

- The key to the CORDIC method is to only rotate by angles of θ where $\tan\theta^{i} = 2^{-i} \Rightarrow$ multiplication by tangent term becomes a shift!

- The table below shows the first few rotation angles that must be used for each iteration (i) of the CORDIC algorithm:

i	θ^{i}	$\tan\theta^{i} = 2^{-i}$
0	45	1
1	26.6	0.5
2	14	0.25
3	7.1	0.125
4	3.6	0.0625

- Thus rotating by an arbitrary angle θ now becomes an iterative process made up of successively smaller pseudo-rotations.

Angle accumulator

- The simplified Givens transform shown earlier can now be expressed for each iteration as:

$$x^{(i+1)} = x^{(i)} - d_i(2^{-i}y^{(i)})$$

$$y^{(i+1)} = y^{(i)} + d_i(2^{-i}x^{(i)})$$

- At this stage we introduce a 3rd equation called the Angle Accumulator which is used to keep track of the accumulative angle rotated at each iteration:

$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)} \qquad \text{(Angle Accumulator)}$$

where d_i = +/- 1

- The symbol d_i is a decision operator and is used to decide which direction to rotate.

The scaling factor

- The Scaling Factor is a by-product of the pseudo-rotations.
- When simplifying the algorithm to allow pseudo-rotations the $\cos\theta$ term was omitted.
- Thus outputs $x^{(n)}$, $y^{(n)}$ are scaled by a factor K_n where:

$$K_n = \prod_n 1/(\cos\theta^{(i)}) = \prod_n (\sqrt{1+2^{(-2i)}})$$

- However if the number of iterations are known then the Scaling Factor K_n can be precomputed.
- Also, $1/K_n$ can be precomputed and used to calculate the true values of $x^{(n)}$ and $y^{(n)}$.

Rotation Mode

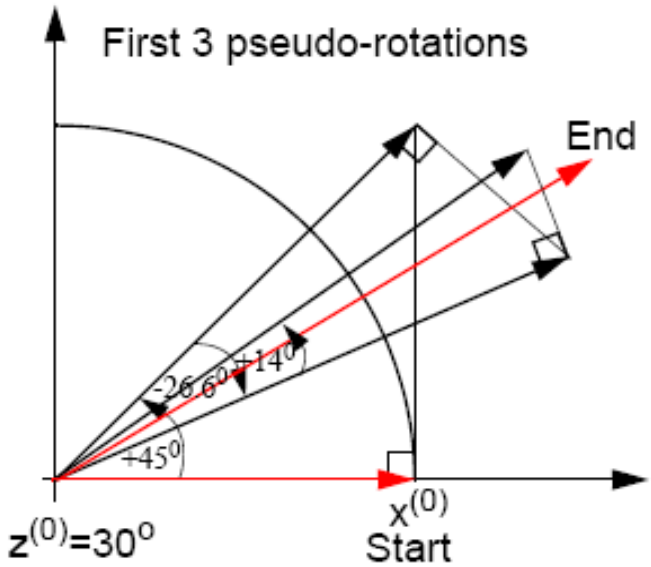
- The CORDIC method is operated in one of two modes;
- The mode of operation dictates the condition for the control operator d_i;
- In Rotation Mode choose: $d_i = \text{sign}(z^{(i)}) \Rightarrow z^{(i)} \rightarrow 0$;
- After n iterations we have:

$$x^{(n)} = K_n(x^{(0)}\cos z^{(0)} - y^{(0)}\sin z^{(0)})$$
$$y^{(n)} = K_n(y^{(0)}\cos z^{(0)} + x^{(0)}\sin z^{(0)})$$
$$z^{(n)} = 0$$

- Can compute $\cos z^{(0)}$ and $\sin z^{(0)}$ by starting with $x^{(0)} = 1/K_n$ and $y^{(0)} = 0$

Example: calculate sin and cos of 30º

i	d_i	$\theta^{(i)}$	$z^{(i)}$	$y^{(i)}$	$x^{(i)}$
0	+1	45	+30	0	0.6073
1	-1	26.6	-15	0.6073	0.6073
2	+1	14	+11.6	0.3036	0.9109
3	-1	7.1	-2.4	0.5313	0.8350
4	+1	3.6	+4.7	0.4270	0.9014
5	+1	1.8	+1.1	0.4833	0.8747
6	-1	0.9	-0.7	0.5106	0.8596
7	+1	0.4	+0.2	0.4972	0.8676
8	-1	0.2	-0.2	0.5040	0.8637
9	+1	0.1	+0	0.5006	0.8657



Vectoring Mode

- In Vectoring Mode choose: $d_i = -\text{sign}(x^{(i)}y^{(i)}) \Rightarrow y^{(i)} \rightarrow 0$
- After *n* iterations we have:

$$x^{(n)} = K_n\left(\sqrt{(x^{(0)})^2 + (y^{(0)})^2}\right)$$

$$y^{(n)} = 0$$

$$z^{(n)} = z^{(0)} + \tan^{-1}\left(\frac{y^{(0)}}{x^{(0)}}\right)$$

Vector magnitude

- Can compute $\tan^{-1} y^{(0)}$ by setting $x^{(0)} = 1$ and $z^{(0)} = 0$

Outline

- Analog vs. digital.
- DSP vs. FPGA.
- Digital design techniques.
  - Performance enhancement techniques.
  - Safe design.
- Digital Signal Processing blocks.
- Examples.

LHC BLM Tunnel Card

Basic components used:

Off-the-shelves

- Current-to-Frequency Converter (x8)
  - Irradiation tests at cyclotrons of Lauvain and PSI
- Actel FPGA (54SX32A) (x1)
  - 208 pin
  - 32,000 LE
  - One-time-programmable
  - Redundant Inputs
    - Triple counter inputs
  - Redundant outputs
    - Double 16bit data bus
    - Double 4bit control bus

Custom Radiation Tolerant ASICs

- A/D Converter AD74240 CMOS (x2)
  - Quad 12bit
  - 40Ms/s
  - Parallel output
- Line Driver LVDS_RX CMOS (x6)
  - 8 LVDS to CMOS line receivers
- Temperature Sensor DCU2 (x1)
  - 12 bit output
- GOL (Gigabit Optical Link) (x2)
  - Analogue parts needed to drive the laser.
  - Algorithm running that corrects SEU.
  - 8b/10b encoding.
  - 16 or 32 bit input.
  - Error reporting (SEU, loss of synchronisation,..)

Figure: CFC card top view.

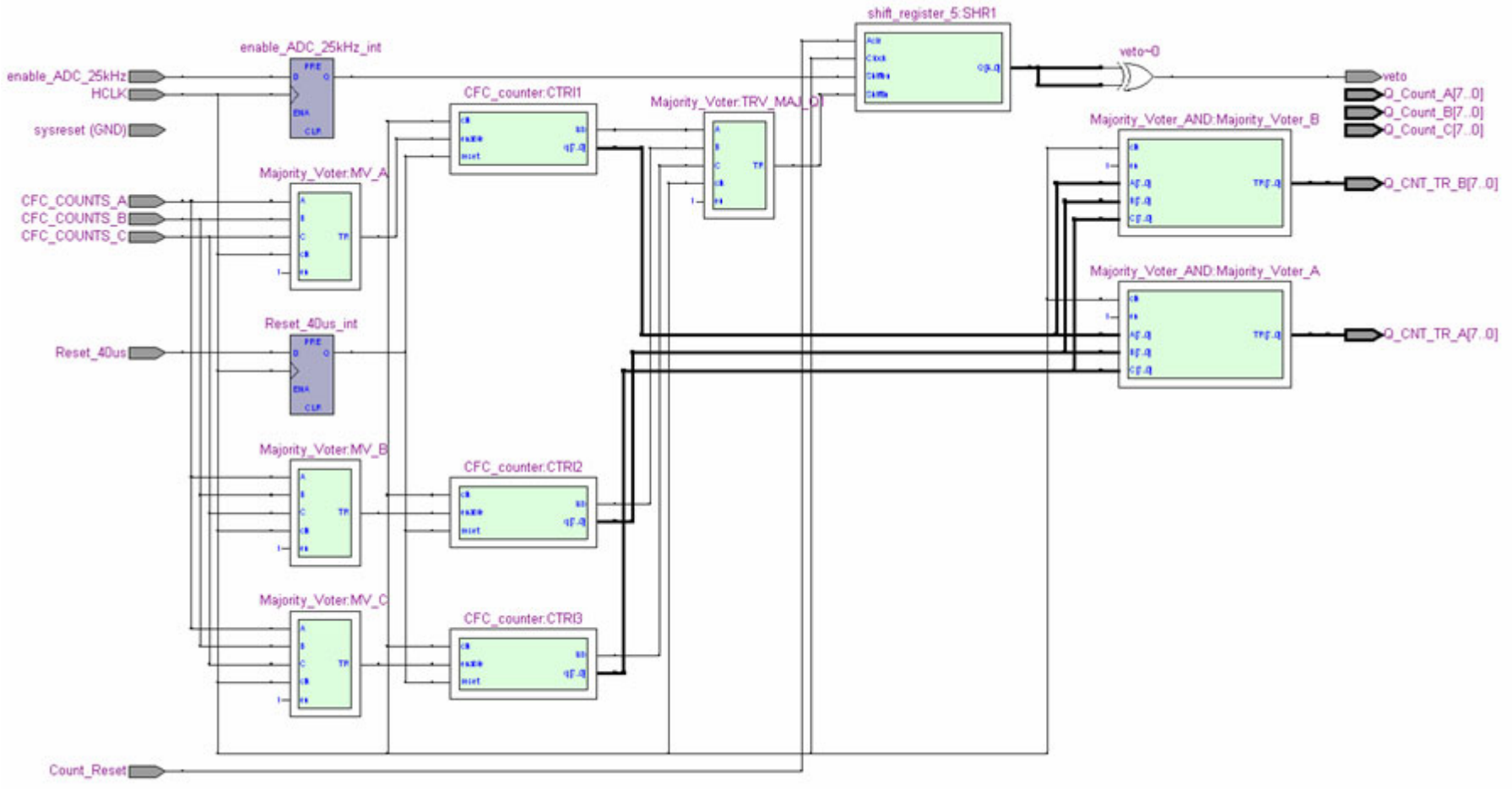
Part Name	Integral Dose (KGy)*
CFC	0.5
ACTEL	3.2
AD41240	10
LVDS_RX	10
DCU2	10
GOH	3.14

Table: Radiation dose withstood by component without error.

* For 20 years of nominal operation it is expected to receive around 200 Gy.

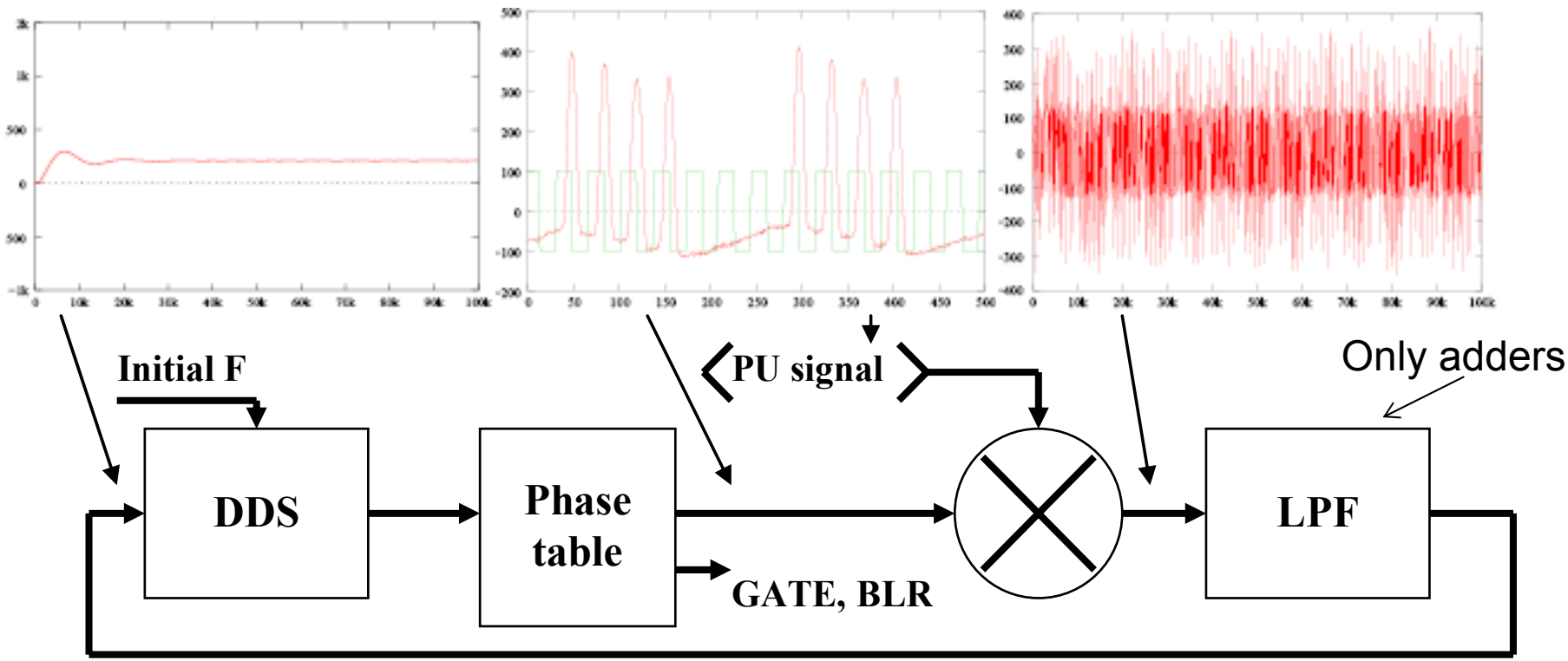
Courtesy Christos Zamantzas

LHC BLM Tunnel Card



Courtesy Christos Zamantzas

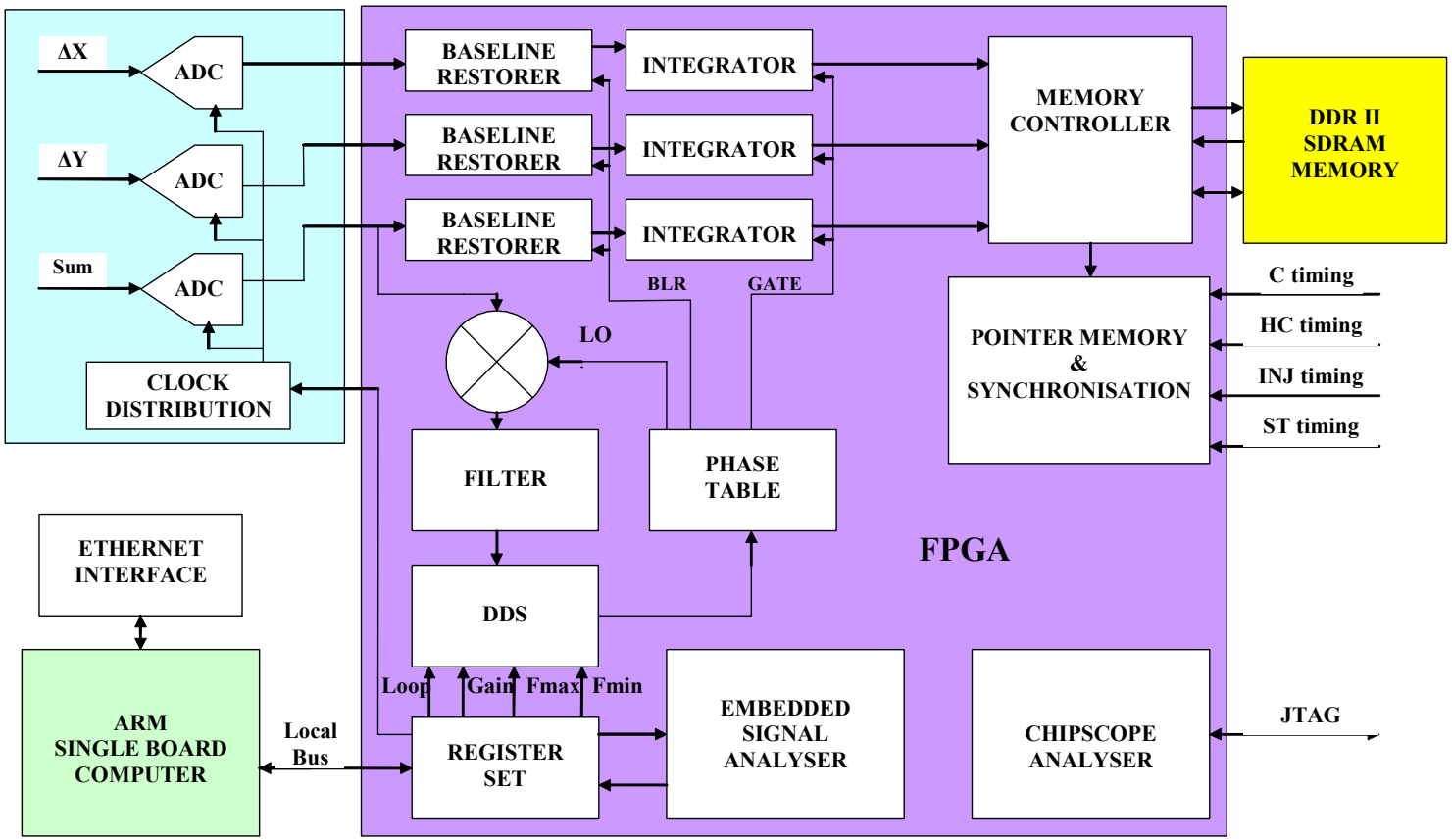
CERN PS Beam Trajectory and Orbit



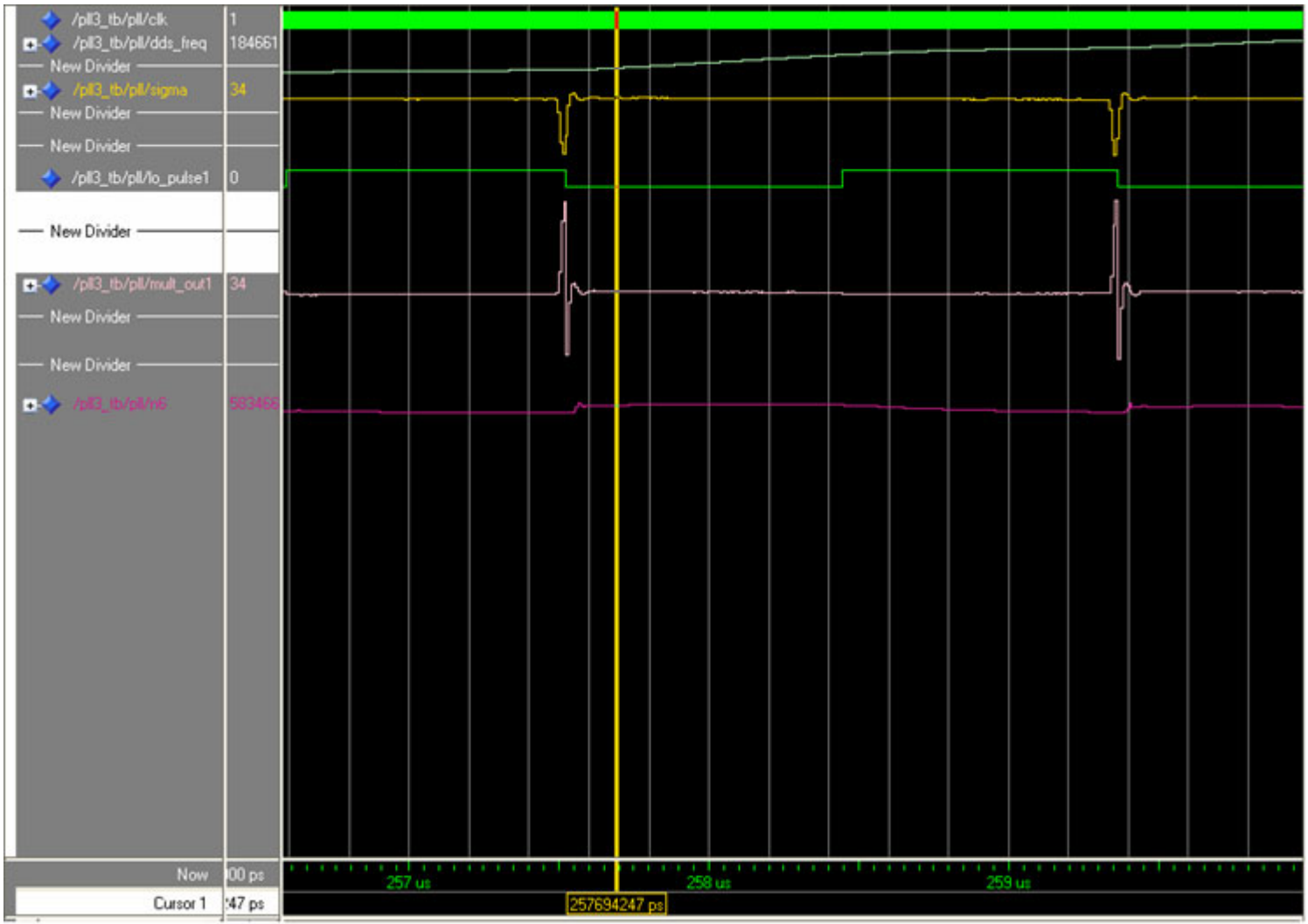
PLL to track non constant revolution frequency

Courtesy Greg Kasprowicz

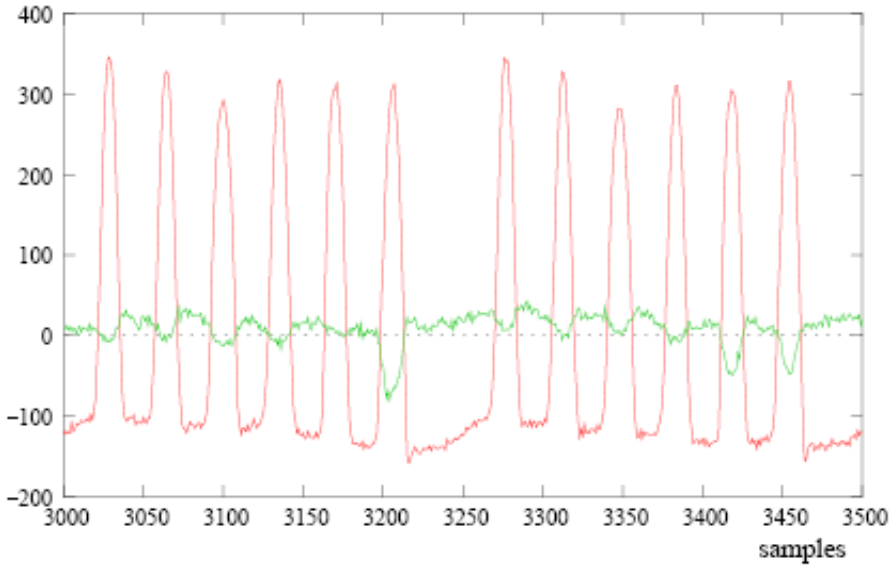
CERN PS Beam Trajectory and Orbit

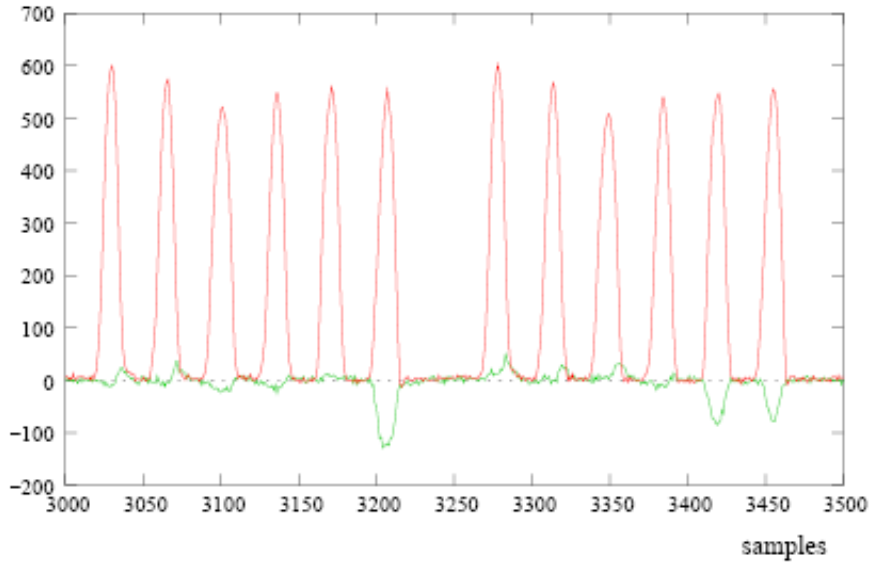


CERN PS Beam Trajectory and Orbit



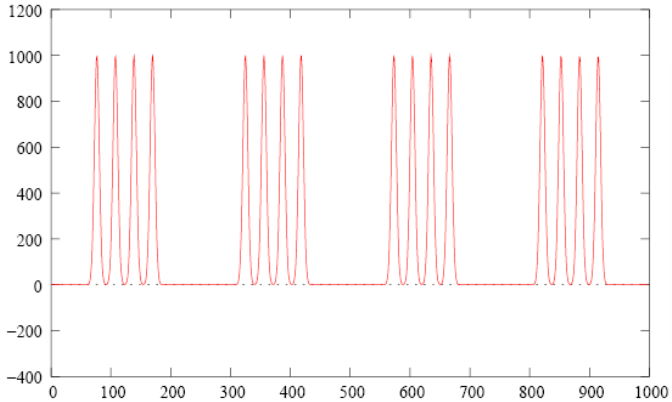
Baseline restoration



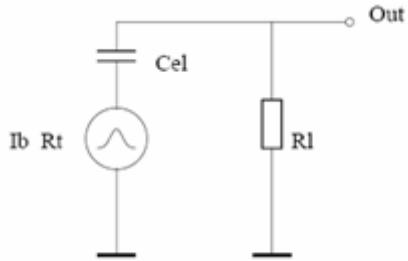


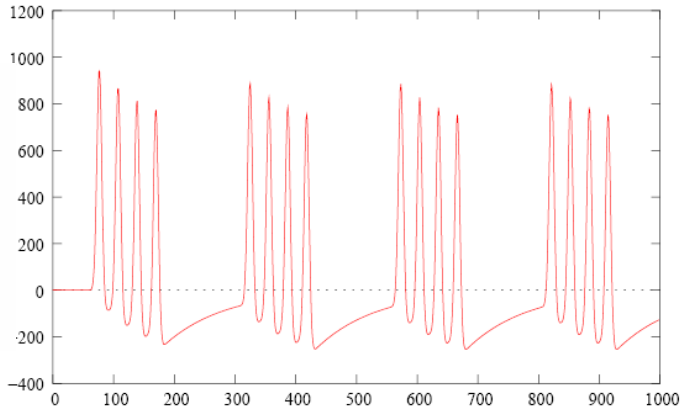
Low pass filter the signal to get an estimate of the base line
Add this to the original signal

Problems with the baseline restorer

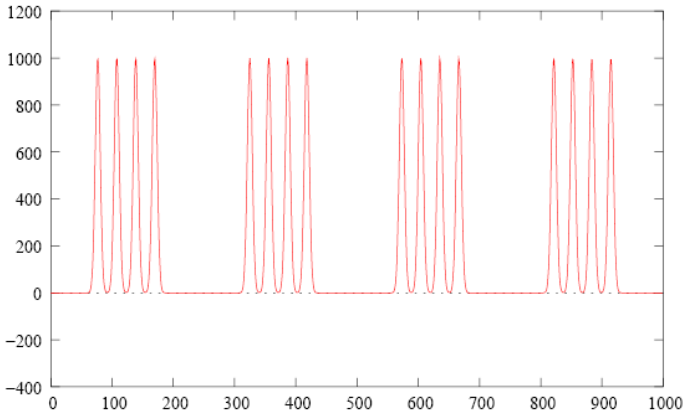


Simulated signal

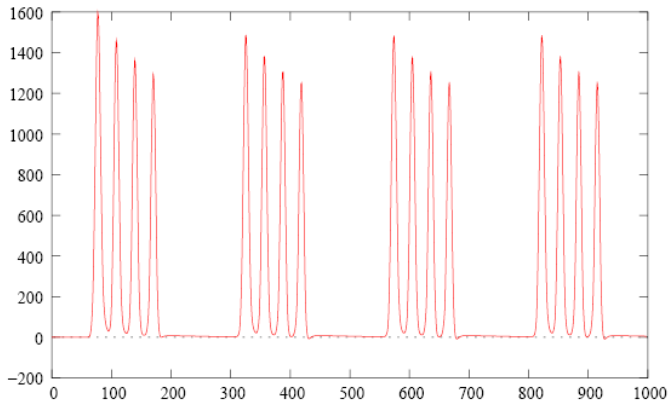




After high pass filter



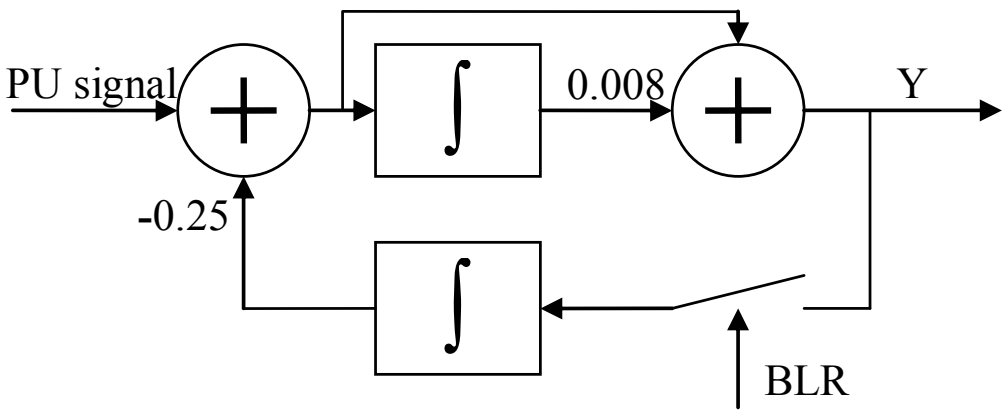
Expected baseline corrected signal



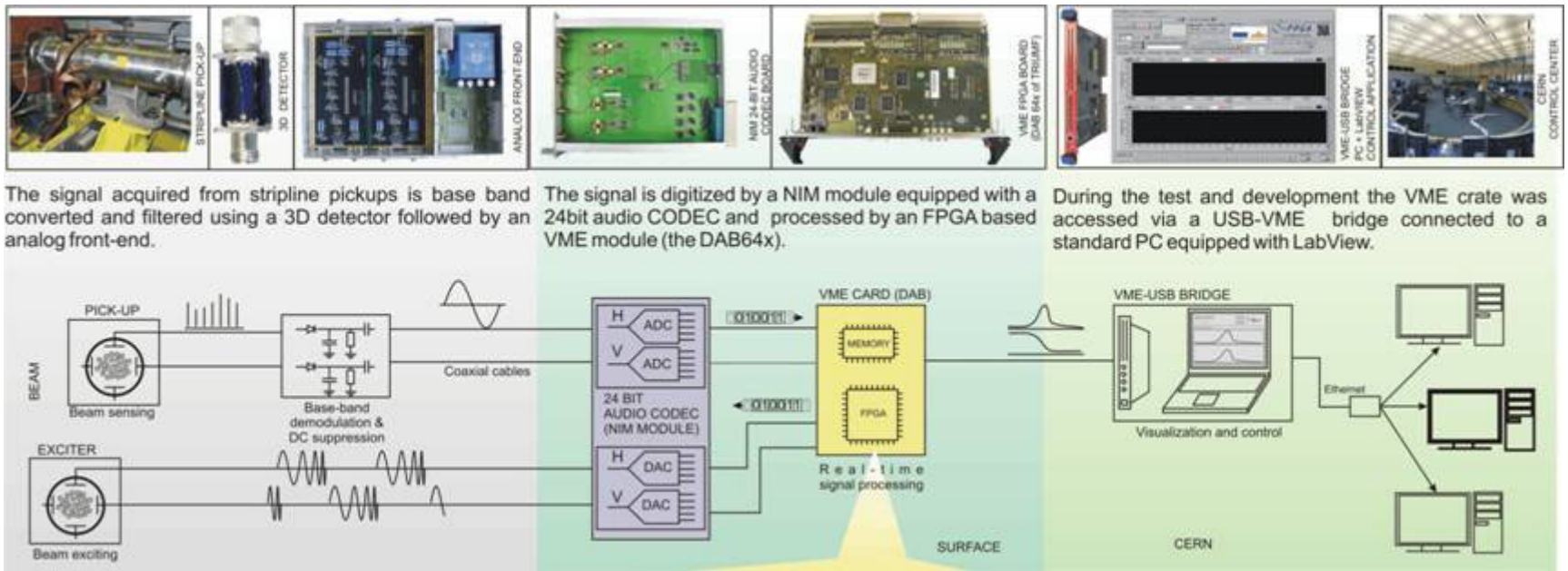
What we get!

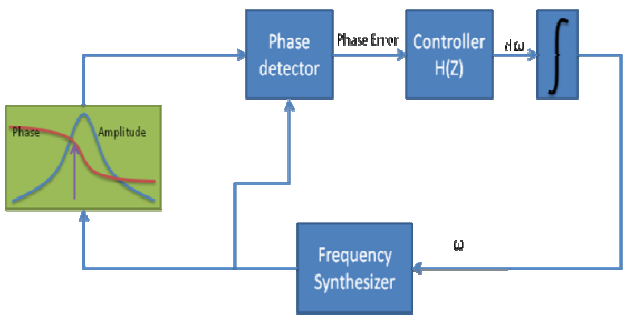
Courtesy Uli Raich

Baseline correction solution

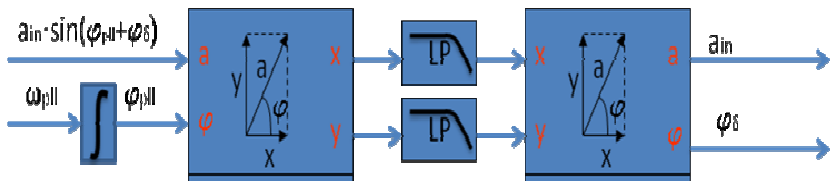


LHC tune PLL





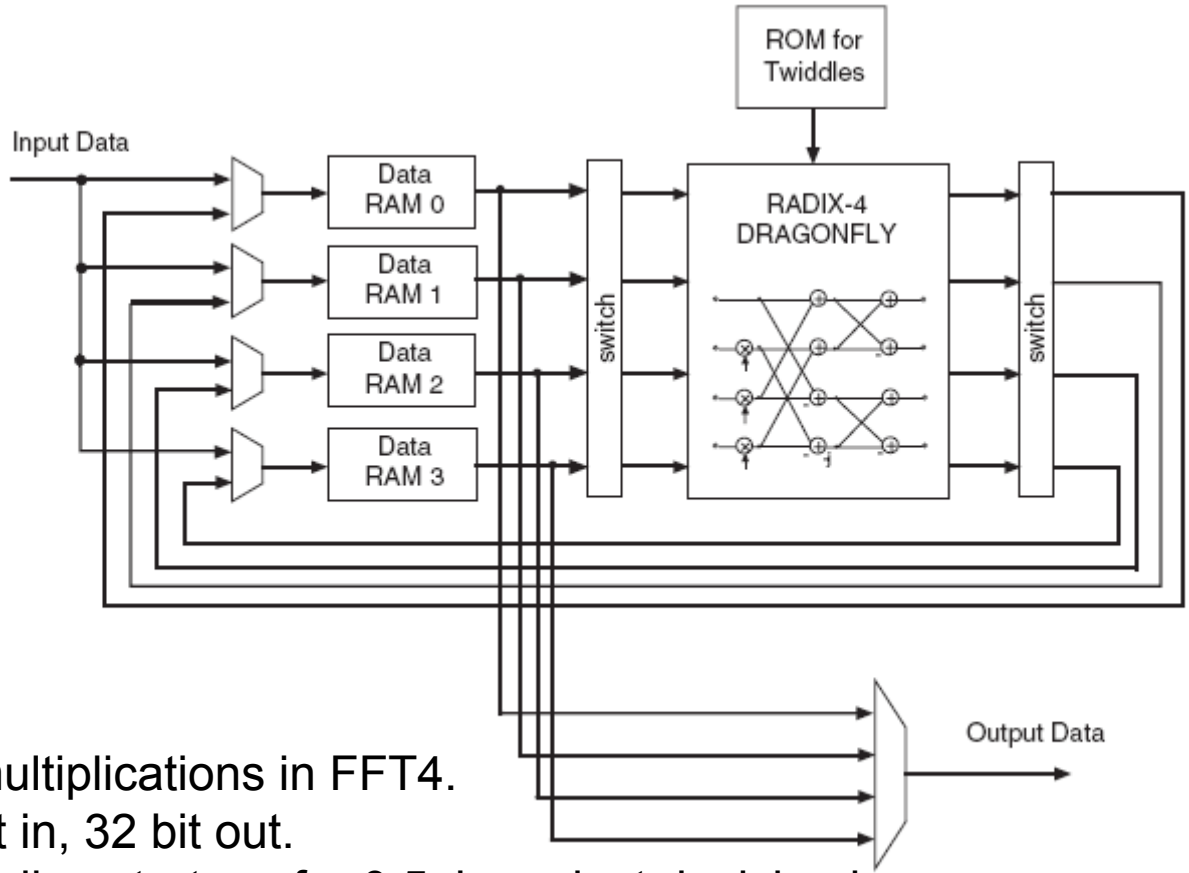
PLL functional diagram



Phase detector

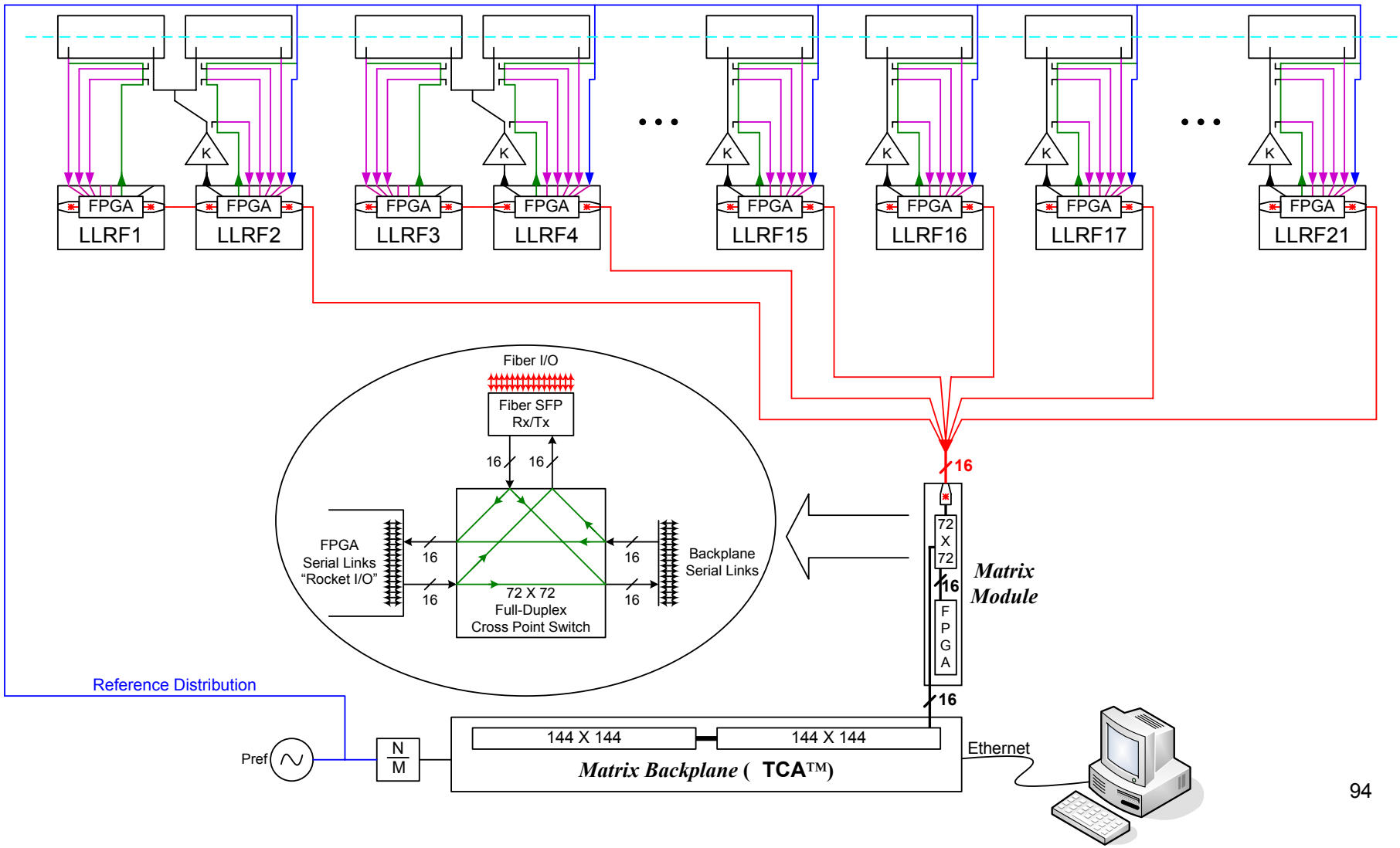
Courtesy Andrea Boccardi

LHC tune FFT



- No multiplications in FFT4.
- 32 bit in, 32 bit out.
- Rounding strategy for 0.5: keep last decision in memory.
- 1024 point FFT in 0.5 ms, dominated by memory access (acquisition time 100 ms).

The future


FPGA
LLRF1
LLRF2
LLRF3
LLRF4
LLRF15
LLRF16
LLRF17
LLRF21
K
Fiber I/O
Fiber SFP
Rx/Tx
16
FPGA
Serial Links
"Rocket I/O"
Backplane
Serial Links
72 X 72
Full-Duplex
Cross Point Switch
72
X
72
Matrix
Module
F
P
G
A
Reference Distribution
Pref
N
M
144 X 144
144 X 144
Matrix Backplane (TCA™)
Ethernet

Acknowledgements

- Many thanks to the following people for material used in this presentation:
  - Tony Rohlev, Elettra.
  - Jeff Weintraub, Xilinx University program.
  - Prof. Stewart, University of Strathclyde.
  - Andrea Boccardi, CERN.
  - Christos Zamantzas, CERN.
  - Greg Kasprowicz, CERN.
  - Uli Raich, CERN.
  - Matt Stettler, LANL.