From Algorithms to Architectures

Prof. Hubert Kaeslin
Microelectronics Design Center
ETH Zürich

VLSI I: Architectures of VLSI Circuits

last update: June 25, 2012

Book chapter 2, pp44...130

Content

You will learn

about the options for tailoring hardware to data/signal processing algorithms.

- General-purpose vs. special-purpose architectures and all sorts of compromises between the two
- Transforms for optimizing VLSI architectures
  - Iterative decomposition, pipelining, replication, time sharing
  - Algebraic transforms
  - Retiming
  - Loop unfolding, pipeline interleaving
- Options for temporary storage of data
- Not so common architectural concepts
  - Bit-serial architectures, distributed arithmetic
  - Computing in semirings

The goals of architecture design

- Decide on the necessary hardware resources for carrying out computations from data and/or signal processing.
- Organize their interplay such as to meet target specifications.

The goals of architecture design

- Decide on the necessary hardware resources for carrying out computations from data and/or signal processing.
- Organize their interplay such as to meet target specifications.
- Concerns:
  1. Functional correctness
  2. Performance targets (throughput, operation rate, etc.)
  3. Circuit size
  4. Energy efficiency
  5. Agility (wrt to evolving needs, changing specs, future standards)
  6. Engineering effort and time to market

Subject

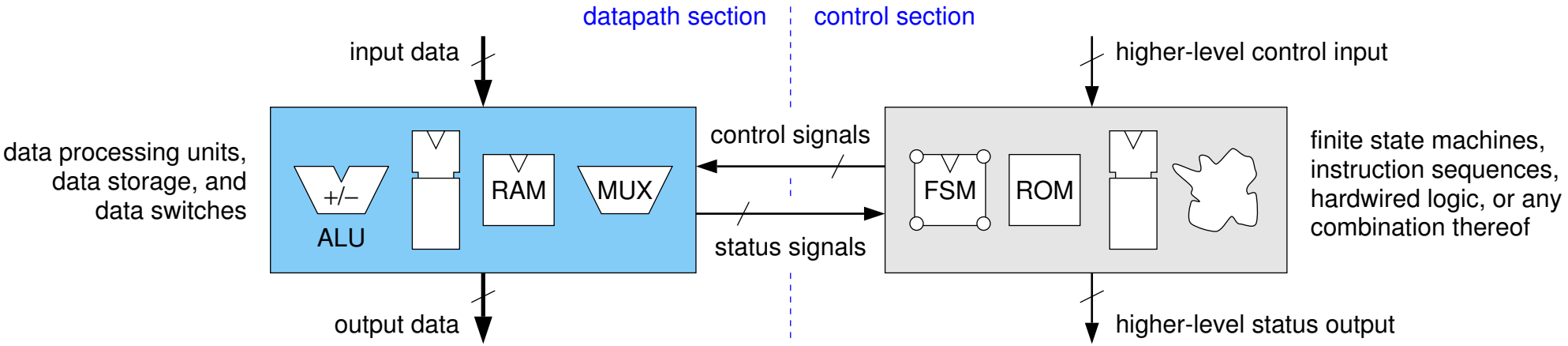
The architectural solution space

What you ought to know about microprocessors

Instruction set processors execute one program instruction after the other in consecutive fetch-load-execute-store cycles.

ALU (arithmetic-logic unit) carries out data manipulations.

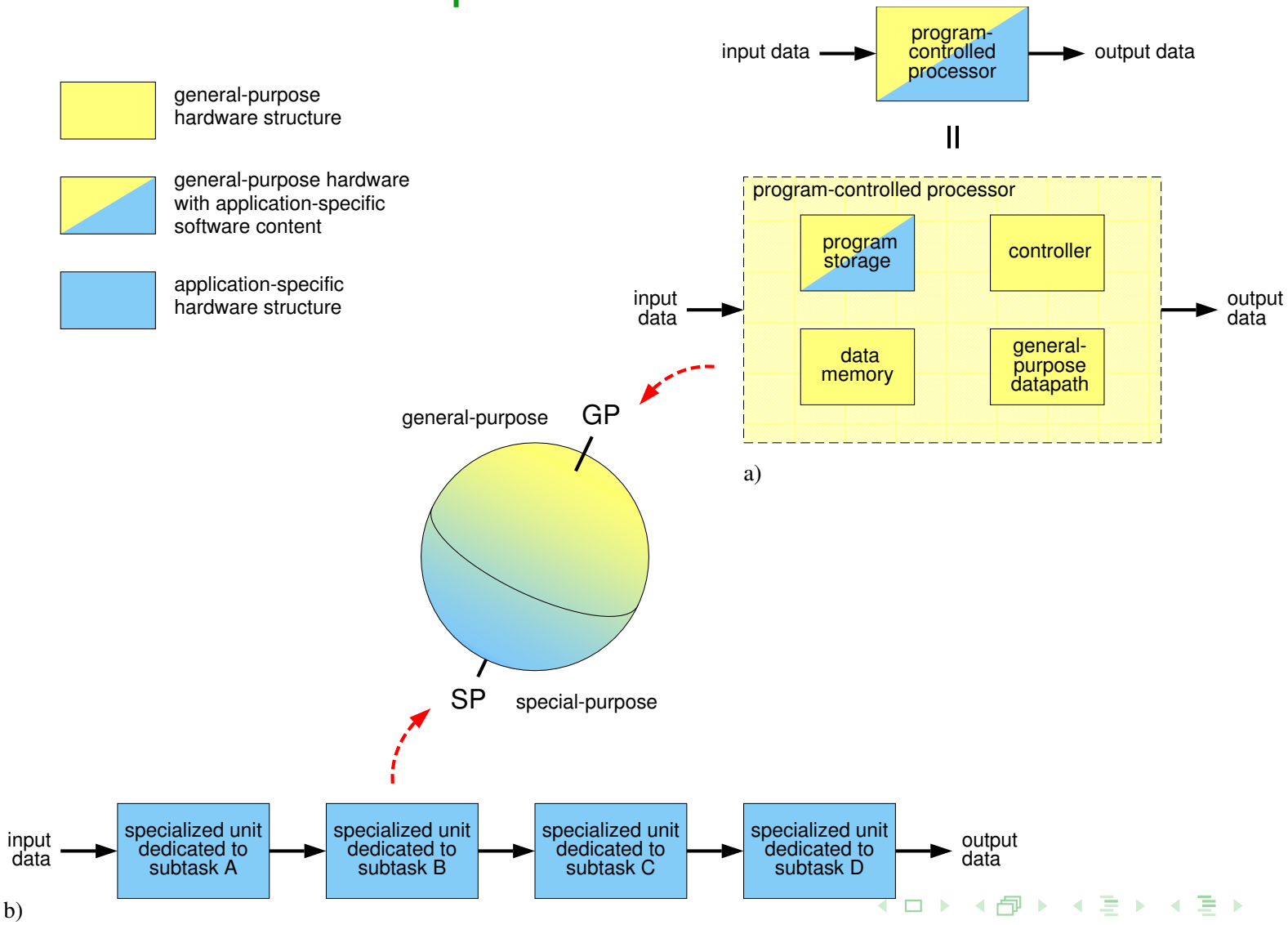
Datapath vs. Control section



von Neumann architecture common memory space, vs.

Harvard architecture separate memory spaces for data and program code.

The architectural antipodes I



The architectural antipodes II

	Hardware architecture	
	General purpose	Special purpose
Algorithm	any, not known a priori	fixed, must be known
Architecture	instruction set processor	dedicated, no single pattern
Execution model	fetch-load-execute-store "instruction-oriented"	process data item and pass on "dataflow-oriented"
Datapath	ALU(s) plus memory	customized design
Controller	with program microcode	typically hardwired
Performance indicator	instructions per second, run time of benchmarks	data throughput, can be anticipated analytically
Strengths	highly flexible, immediately available, routine design flow, low up-front costs	room for max. performance, highly energy-efficient, lean circuitry

The architectural antipodes III

Guideline

Before embarking in ASIC design, find out

- Does an architecture dedicated to the application at hand make sense
- or is a program-controlled general-purpose processor more adequate?

The architectural antipodes III

Guideline

Before embarking in ASIC design, find out

- Does an architecture dedicated to the application at hand make sense
- or is a program-controlled general-purpose processor more adequate?

- Opting for commercial microprocessors and/or FPL sidesteps many technical problems that absorb much attention when a custom IC is to be designed instead.
- Conversely, it is precisely
  - the focus on the payload computation,
  - the absence of programming and configuration overhead, and
  - the full control over architecture, circuit, and layout details

  that make it possible to optimize performance and energy efficiency.

Example: Viterbi decoder

Architecture	General purpose		Special purpose	
Key component	DSP TI TMS320C6455 without Viterbi coprocessor VCP2	DSP TI TMS320C6455 with Viterbi coprocessor VCP2	ASIC sem03w6 ETH	ASIC sem05w1 ETH
Number of chips	1	1	1	1
CMOS process	90 nm	90 nm	250 nm 5Al	250 nm 5Al
Program code	187 Kibyte	242 Kibyte	none	none
Circuit size	n.a.	n.a.	73 kGE	46 kGE
Max. throughput	45 kbit/s	9 Mbit/s	310 Mbit/s	54 Mbit/s
@ clock	1 GHz	1 GHz	310 MHz	54 MHz
Power dissipation	2.1 W	2.1 W	1.9 W	50 mW
Year	2005	2005	2004	2006

Reasons:

- DSP optimized for sustained multiply-accumulates, word width 32 bit.
- Viterbi algorithm arranged to do without multiplication.
- Viterbi algorithm arranged to do with words of 6 bit or less.
- Dedicated architectures can exploit full potential for parallelism.

Example: AES block cipher encrypter/decrypter

(Rijndael algorithm)

Architecture	General purpose	Special purpose		
Key component	CISC Processor Pentium III	FPGA Xilinx Virtex-II	ASIC (ETH) CryptoFun	ASIC (UCLA) Rijndael core
Number of chips	motherboard	1 + config.	1	1
CMOS process	n.a.	150 nm 8Al	180 nm 4Al2Cu	180 nm 4Al2Cu
Max. throughput	648 Mbit/s	1.32 Gbit/s	2.0 Gbit/s	1.6 Gbit/s
@ clock	1.13 GHz	n.a.	172 MHz	125 MHz
Power dissipation	41.4 W	490 mW	n.a.	56 mW
@ supply	n.a.	1.5 V	1.8 V	1.8 V
Year	2000	$\approx$ 2002	2007	2002

Reasons:

- Multiple LUTs included in hardware for S-Box function and inverse.
- Ciphering and subkey preparation carried out by concurrent units.
- Rijndael algorithm designed with Pentium III architecture in mind (MMX instructions, LUTs that fit into cache memory, etc.).
- Power dissipation of general-purpose processor remains daunting.

When do dedicated architectures make sense?

Dedicated architectures are favored by real-time applications such as

- Data, audio and video (de)compression
- Ciphering & deciphering (primarily for secret key ciphers)
- Error correction coding
- Digital modulation & demodulation
  (for modems, wireless communication, and disk drives)
- Adaptive channel equalization for copper lines and optical fibers
- Multipath combiners in broadband wireless access networks
- Computer graphics and video rendering
- Multimedia (e.g. MPEG, HDTV)
- Pattern recognition

Answer

"Does it make sense to consider dedicated hardware architectures?"

YES Dedicated architectures outperform program-controlled processors by orders of magnitude (wrt throughput and energy efficiency) in many transformatorial systems where data streams get processed in fairly regular ways.

Answer

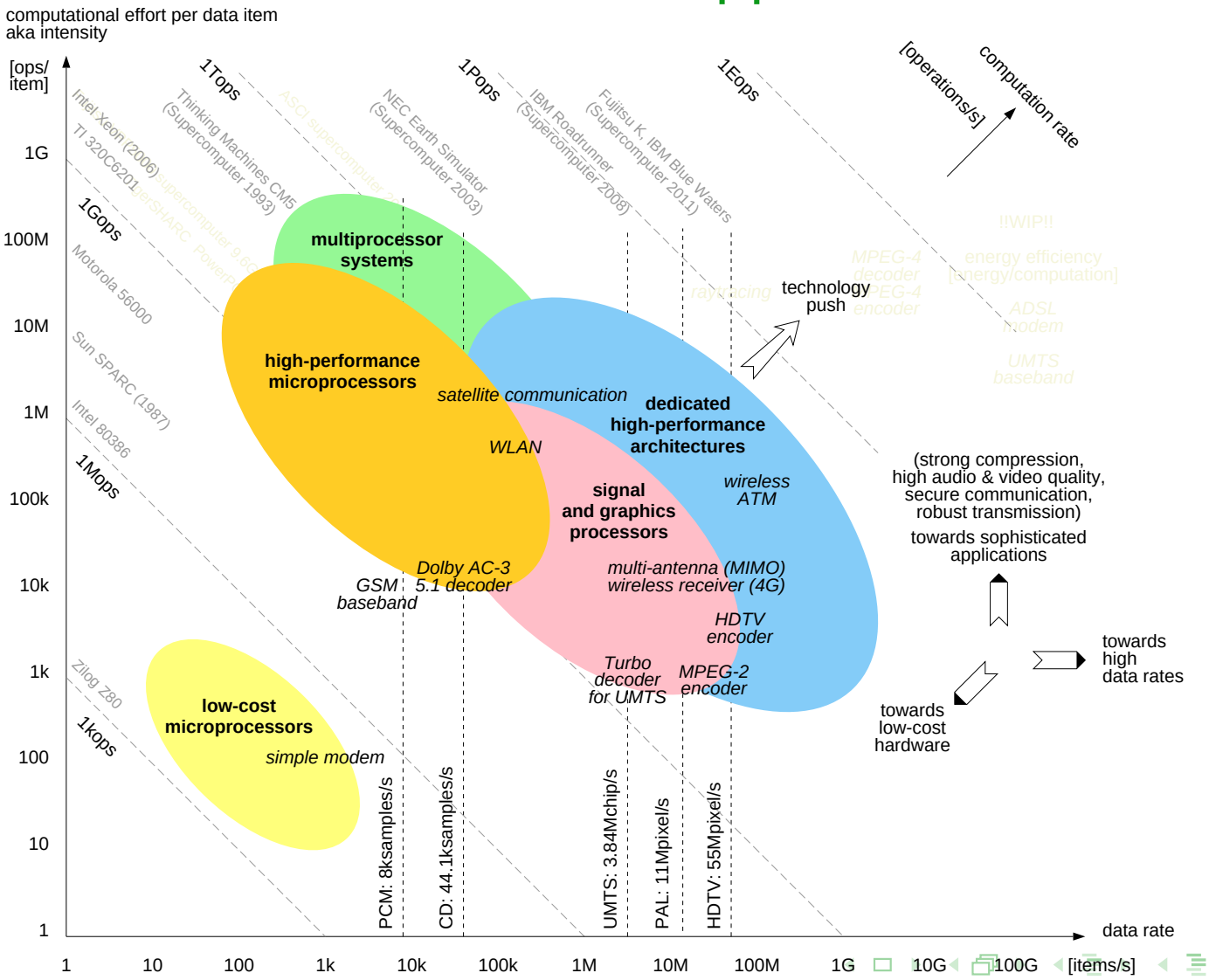
"Does it make sense to consider dedicated hardware architectures?"

YES Dedicated architectures outperform program-controlled processors by orders of magnitude (wrt throughput and energy efficiency) in many transformatorial systems where data streams get processed in fairly regular ways.

but also

NO Dedicated architectures can not rival the agility and economy of processor-type designs in applications where the computation is primarily reactive, very irregular, highly data-dependent, or memory-hungry.

Computational needs of various applications



Algorithms suitable for dedicated architectures

What makes an algorithm suitable for dedicated VLSI architectures?

Ideally:

1. Loose coupling between major processing tasks
   - Well-defined functional specification for each task.
   - Manageable interactions between them.

Algorithms suitable for dedicated architectures

What makes an algorithm suitable for dedicated VLSI architectures?

Ideally:

1. Loose coupling between major processing tasks
   - Well-defined functional specification for each task.
   - Manageable interactions between them.
2. Simple control flow
   - Course of operation does not depend on the data being processed.
   - No need for overly many modes of operations, data formats, etc.
     - Makes it possible to anticipate the datapath resources required to meet throughput goal and to design the architecture accordingly.
     - Permits control by counters and simple finite state machines.

Algorithms suitable for dedicated architectures

... continued

3. Regular data flow, recurrence of a few identical operations
   - Opens a door for sharing hardware resources in an efficient way.

Algorithms suitable for dedicated architectures

... continued

3. Regular data flow, recurrence of a few identical operations
   - Opens a door for sharing hardware resources in an efficient way.
4. Reasonable storage requirements
   - Renders on-chip memories economically possible.
   - Massive storage requirements in conjunction with moderate computational burdens place dedicated architectures at a disadvantage.

Algorithms suitable for dedicated architectures

... continued

3. Regular data flow, recurrence of a few identical operations
   - Opens a door for sharing hardware resources in an efficient way.
4. Reasonable storage requirements
   - Renders on-chip memories economically possible.
   - Massive storage requirements in conjunction with moderate computational burdens place dedicated architectures at a disadvantage.
5. Compatible with finite precision arithmetics
   - Insensitive to effects from finite precision, no need for floating-point arithmetics.
   - Area, logic delay, interconnect length, parasitic capacitances, and energy dissipation all grow with word width, they combine into a burden that multiplies at an overproportional rate.

Example: Fixed-point division

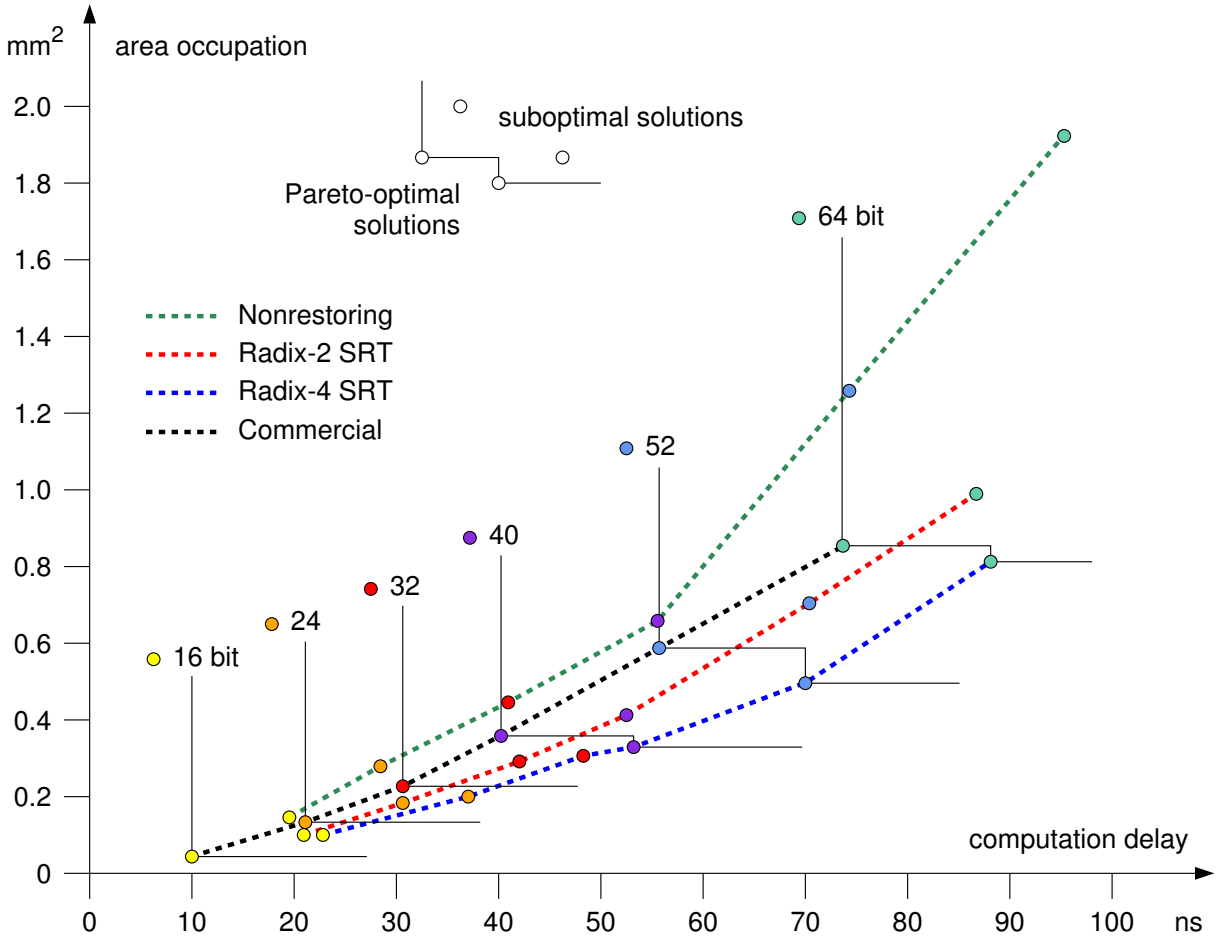


Figure: Comparison of hardware divider architectures for a 180 nm CMOS process under worst-case PTV conditions. Note the impact of quotient width.

Algorithms suitable for dedicated architectures

... continued

6. Non-recursive linear time-invariant computation
   - Opens a door for reorganizing the data processing in many ways.
   - High-speed operation, in particular, is much easier to obtain.

Algorithms suitable for dedicated architectures

... continued

6. Non-recursive linear time-invariant computation
   - Opens a door for reorganizing the data processing in many ways.
   - High-speed operation, in particular, is much easier to obtain.
7. No transcendental functions
   - Roots, logarithmic, exponential, or trigonom. functions, translations between incompatible number systems are expensive in hardware.
     - Results must either be stored in large lookup tables (LUTs) or
     - get calculated on-line in lengthy computation sequences.

Algorithms suitable for dedicated architectures

... continued

6. Non-recursive linear time-invariant computation
   - Opens a door for reorganizing the data processing in many ways.
   - High-speed operation, in particular, is much easier to obtain.
7. No transcendental functions
   - Roots, logarithmic, exponential, or trigonom. functions, translations between incompatible number systems are expensive in hardware.
     - Results must either be stored in large lookup tables (LUTs) or
     - get calculated on-line in lengthy computation sequences.
8. Extensive usage of operations unavailable from instruction sets
   - Replace lengthy instruction sequences by dedicated datapath units.
   - Fixed arguments often allow for some form of preprocessing, e.g.
     - drop unit factors and/or zero sum terms,
     - adopt special number representation schemes,
     - take advantage of symmetries and precomputed lookup tables.

Algorithms suitable for dedicated architectures

... continued

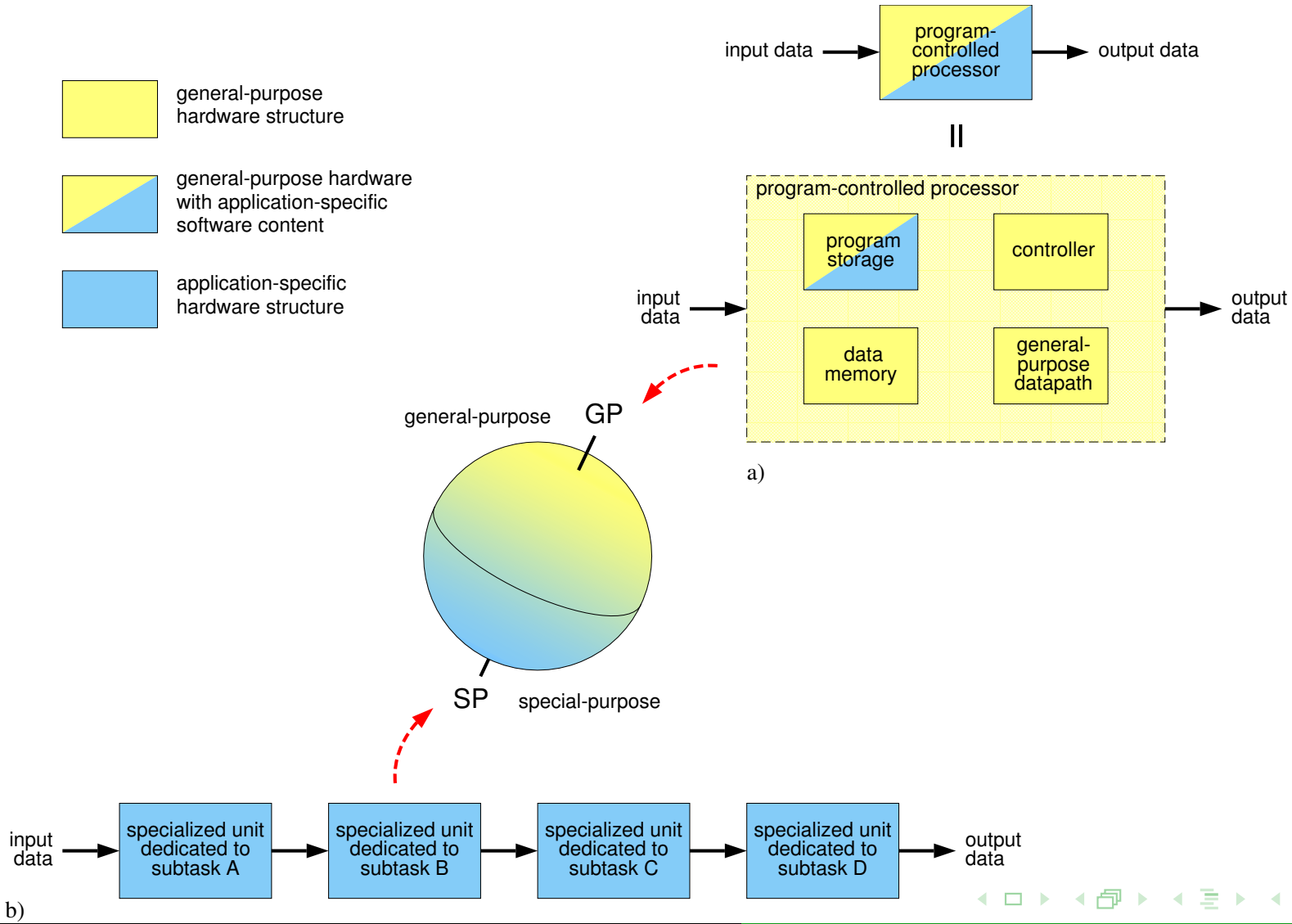
9. No divisions and multiplications on very wide data words
   - Much more expensive than addition and subtraction.
   - Vast numerical range of results gives rise to scaling issues.
   - Matrix inversion is a particularly nasty case in point as it involves divisions and often brings about numerical instability.

Algorithms suitable for dedicated architectures

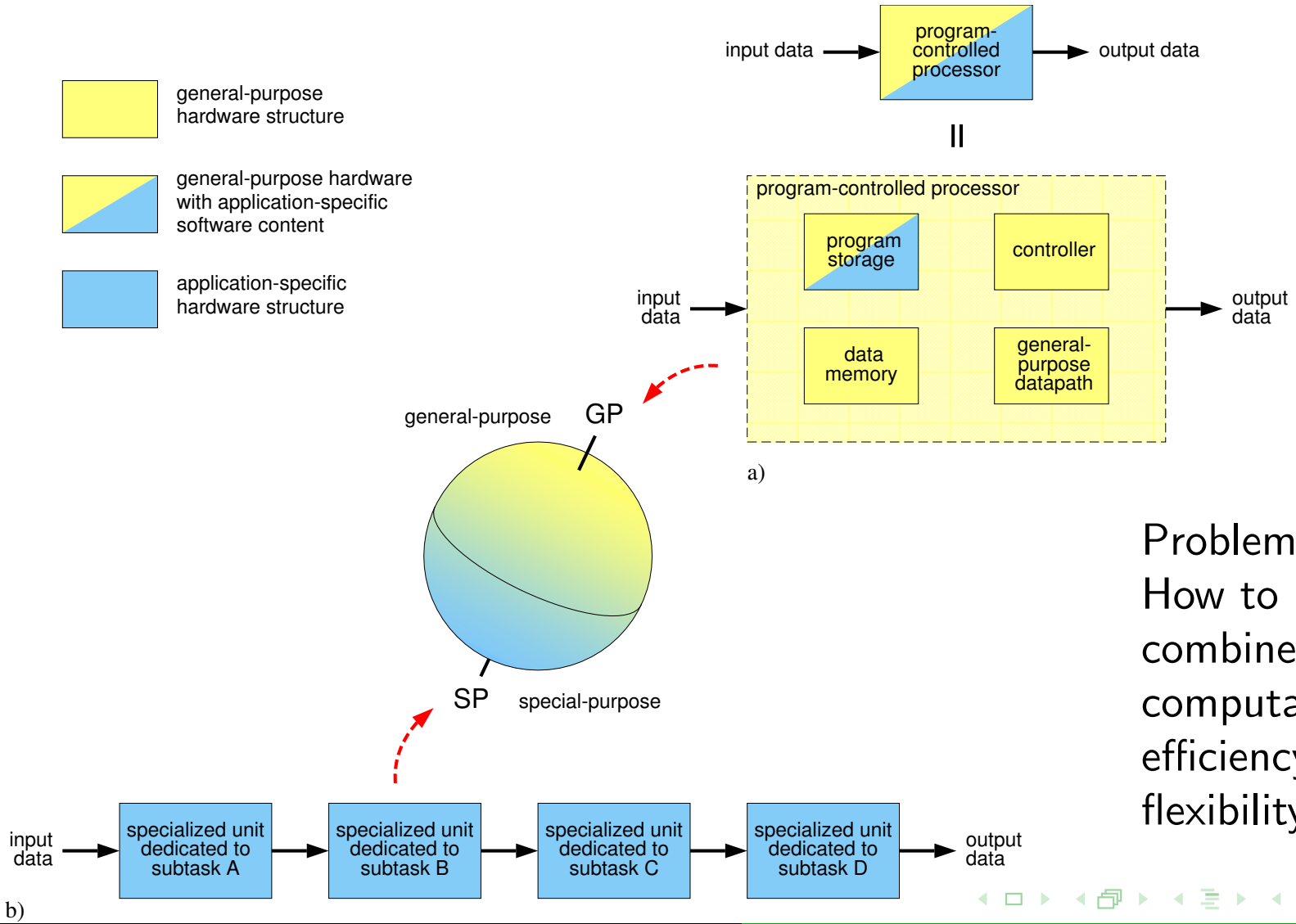
... continued

9. No divisions and multiplications on very wide data words
   - Much more expensive than addition and subtraction.
   - Vast numerical range of results gives rise to scaling issues.
   - Matrix inversion is a particularly nasty case in point as it involves divisions and often brings about numerical instability.
10. Throughput rather than latency is what matters
    - Tight latency requirements rule out pipelining
    - but are not in favor of microprocessors either as program-controlled operation can not normally guarantee fixed response times, even less so when a complex operating system is involved.

The architectural solution space



The architectural solution space



Problem:
How to combine computational efficiency with flexibility?

Have a look at typical electronic devices

Application	Subfunctions primarily characterized by irregular control flow and/or need for flexibility	Subfunctions primarily characterized by repetitive control flow and need for comput. efficiency
DVD player	user interface, track seeking, tray and spindle control, processing of non-video data (directory, title, author, subtitles, region codes)	16-to-8 bit demodulation, error correction, MPEG-2 decompression (discrete cosine transform), video signal processing
Smartphone	user interface, SMS, directory management, battery monitoring, communication protocol, channel allocation, roaming, accounting	intermediate frequency filtering, (de)modulation, channel (de)coding, error correction (de)coding, (de)ciphering, speech and video (de)compression

Have a look at typical electronic devices

Application	Subfunctions primarily characterized by irregular control flow and/or need for flexibility	Subfunctions primarily characterized by repetitive control flow and need for comput. efficiency
DVD player	user interface, track seeking, tray and spindle control, processing of non-video data (directory, title, author, subtitles, region codes)	16-to-8 bit demodulation, error correction, MPEG-2 decompression (discrete cosine transform), video signal processing
Smartphone	user interface, SMS, directory management, battery monitoring, communication protocol, channel allocation, roaming, accounting	intermediate frequency filtering, (de)modulation, channel (de)coding, error correction (de)coding, (de)ciphering, speech and video (de)compression

Guideline

Segregate the needs for computational efficiency from those of agility!

1. Dedicated satellites and 2. Host with helper engines

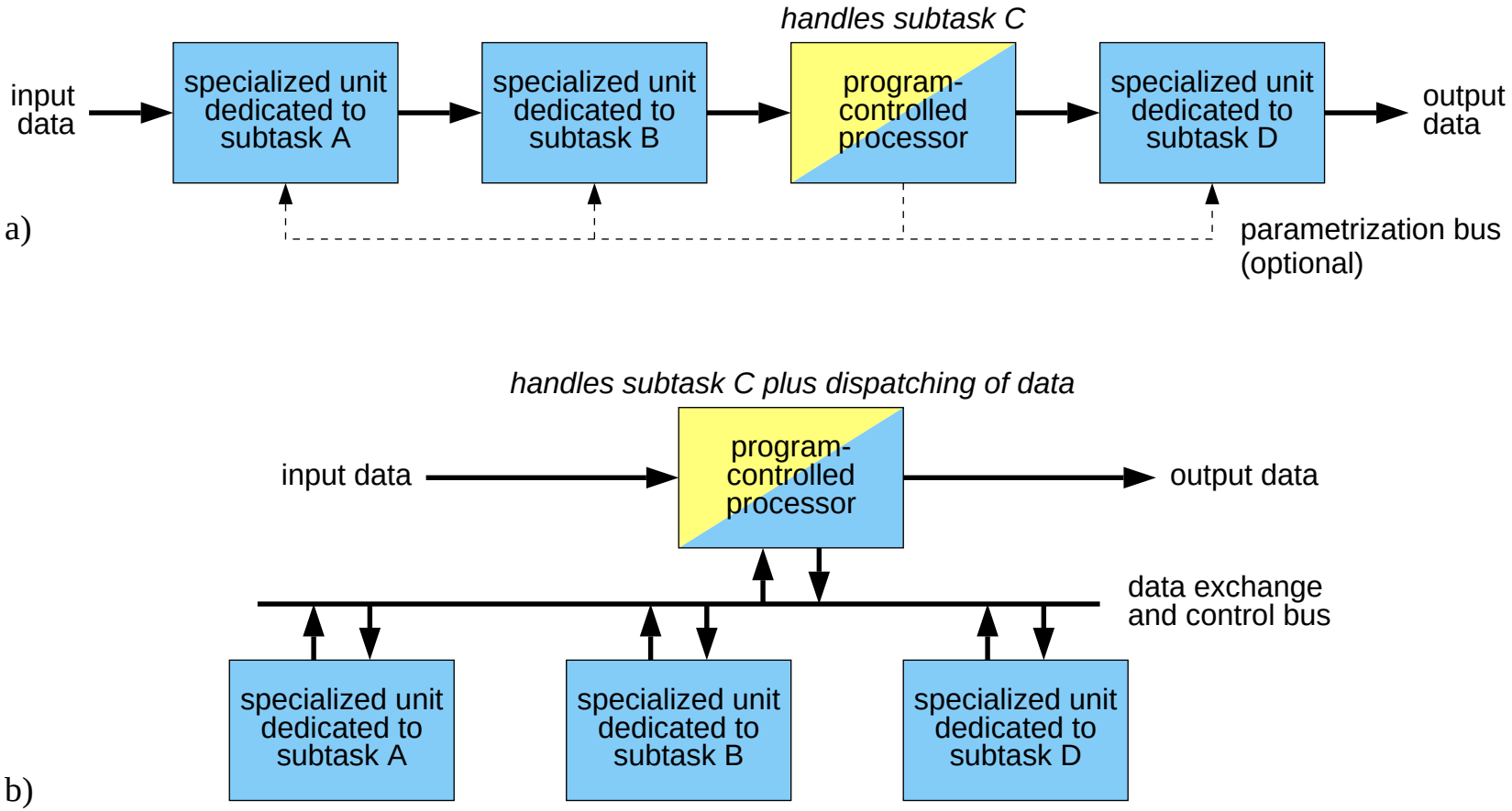
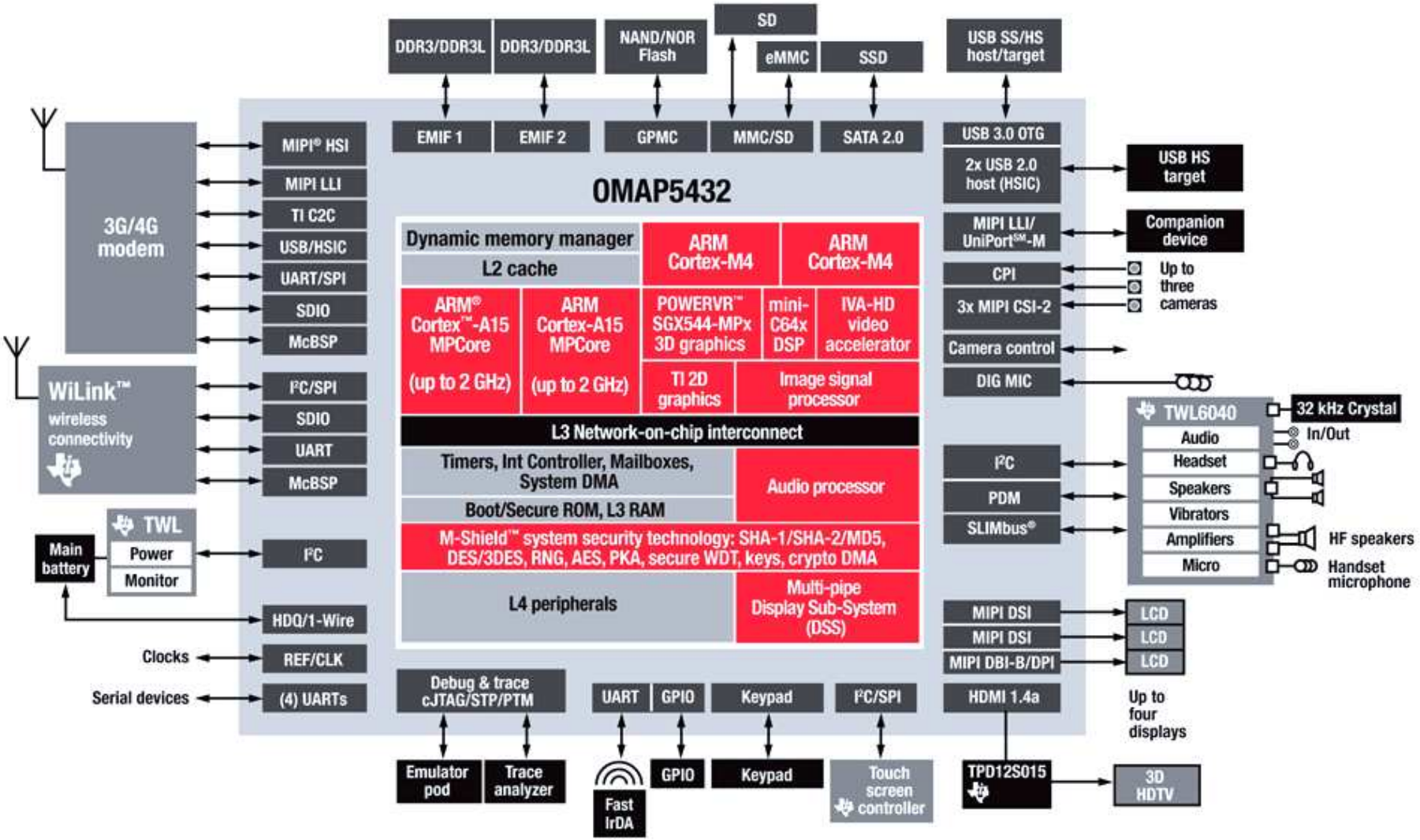
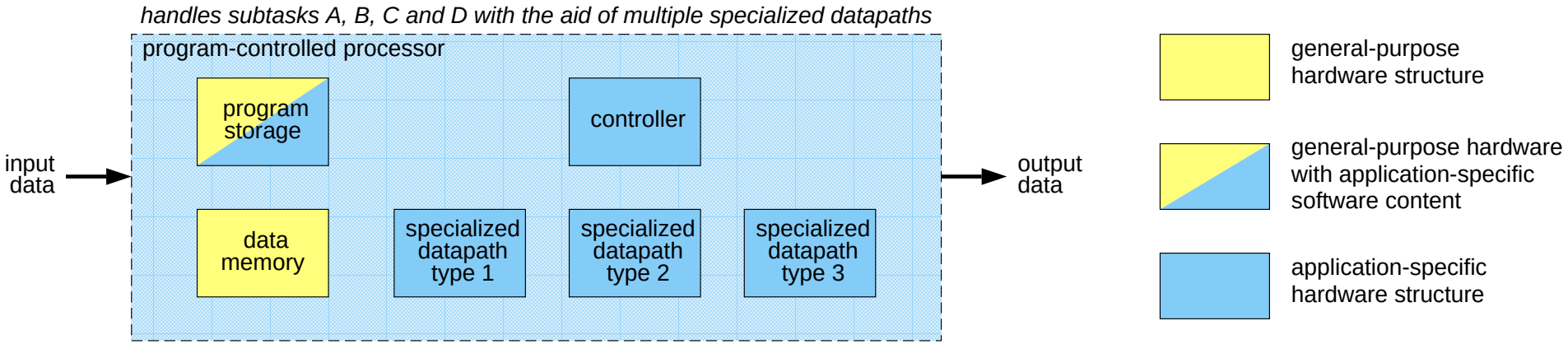


Figure: Chain of general-purpose processor and dedicated satellites (a), host computer with specialized fixed-function blocks or coprocessors (b).

Example: System on a chip for smartphones (by Texas Instr.)



3. Application-specific instruction set processor (ASIP)



- Program-controlled operation $\rightsquigarrow$ highly flexible
- Application-specific features confined to datapath circuitry
- Single thread of execution (concurrency limited to SIMD), easily extended to multiple threads (by including multiple ASIP cores)

Example: AES cipher encrypter/decrypter revisited

General purpose	Special purpose			ASIP
CISC Processor Pentium III	FPGA Xilinx Virtex-II	ASIC (ETH) CryptoFun	ASIC (UCLA) Rijndael core	Cryptoprocessor core UCLA
motherboard	1 + config.	1	1	1
Assembler	none	none	none	Assembler
n.a.	n.a.	76 kGE	173 kGE	73.2 kGE
n.a.	150 nm 8Al	180 nm 4Al2Cu	180 nm 4Al2Cu	180 nm 4Al2Cu
648 Mbit/s	1.32 Gbit/s	2.0 Gbit/s	1.6 Gbit/s	3.43 Gbit/s
1.13 GHz	n.a.	172 MHz	125 MHz	295 MHz
41.4 W	490 mW	n.a.	56 mW	86 mW
n.a.	1.5 V	1.8 V	1.8 V	1.8 V
2000	$\approx$ 2002	2007	2002	2004

Observation

ASIP combines excellent throughput and low power with the agility of a program-controlled architecture.

Example: AES cipher encrypter/decrypter revisited

General purpose	Special purpose			ASIP
CISC Processor Pentium III	FPGA Xilinx Virtex-II	ASIC (ETH) CryptoFun	ASIC (UCLA) Rijndael core	Cryptoprocessor core UCLA
motherboard	1 + config.	1	1	1
Assembler	none	none	none	Assembler
n.a.	n.a.	76 kGE	173 kGE	73.2 kGE
n.a.	150 nm 8Al	180 nm 4Al2Cu	180 nm 4Al2Cu	180 nm 4Al2Cu
648 Mbit/s	1.32 Gbit/s	2.0 Gbit/s	1.6 Gbit/s	3.43 Gbit/s
1.13 GHz	n.a.	172 MHz	125 MHz	295 MHz
41.4 W	490 mW	n.a.	56 mW	86 mW
n.a.	1.5 V	1.8 V	1.8 V	1.8 V
2000	$\approx$ 2002	2007	2002	2004

Observation

ASIP combines excellent throughput and low power with the agility of a program-controlled architecture.

Catch: proprietary instruction set $\rightsquigarrow$ special assembler, libraries, debuggers, ...

A framework for accelerating ASIP design

LISA = Language for Instruction Set Architectures
(developed by CoWare Inc. acquired by Synopsys in 2010)

The design flow essentially goes

1. Define the most adequate instruction set for a target application,
2. Refine the architecture into a cycle-accurate model (optional),
3. Cast your architecture into an RTL-type model (optional)

using the LISA language.

System-level software tools then generate

- Assembler, linker, and simulator tools.
- VHDL synthesis code (from the RTL model).

Predefined processor templates also available.

4. Reconfigurable computing (promoted by FPL vendors)

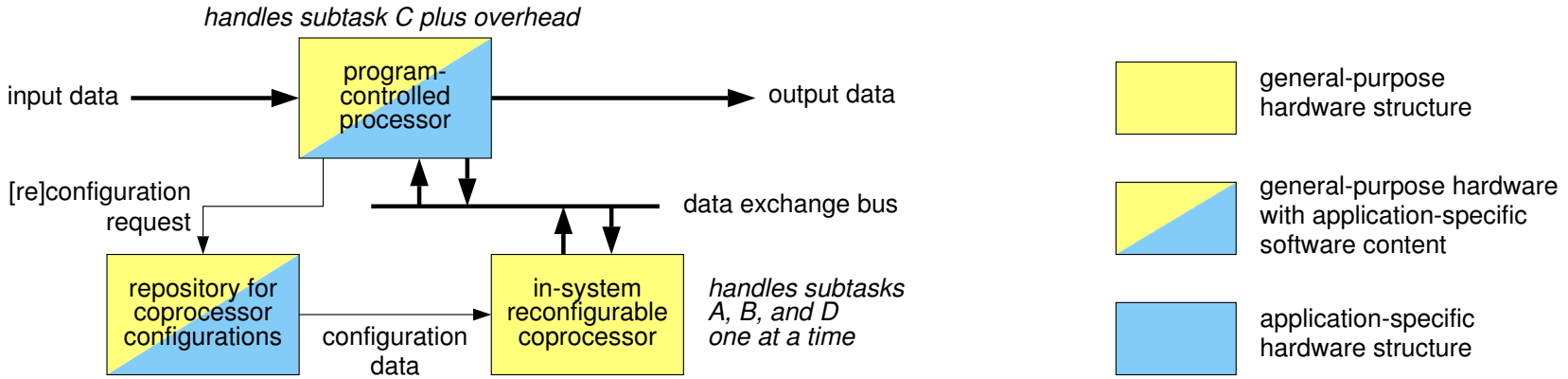


Figure: General-purpose processor with juxtaposed reconfigurable coprocessor.

General procedure:

1. Designers come up with a specific circuit structure for each major piece of suitable computation.
2. All configurations get stored in memory.
3. Whenever the host encounters a call to one of those computations, it downloads the pertaining configuration file into the FPL
4. Host feeds coprocessor with data and fetches results.
5. Host proceeds after computation completes.

4. Reconfigurable computing (promoted by FPL vendors)

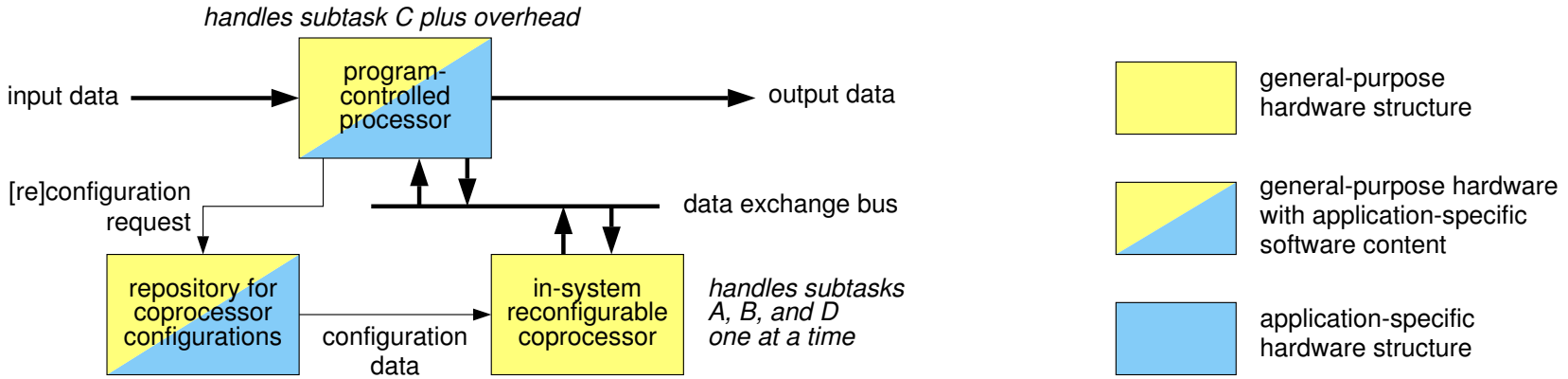
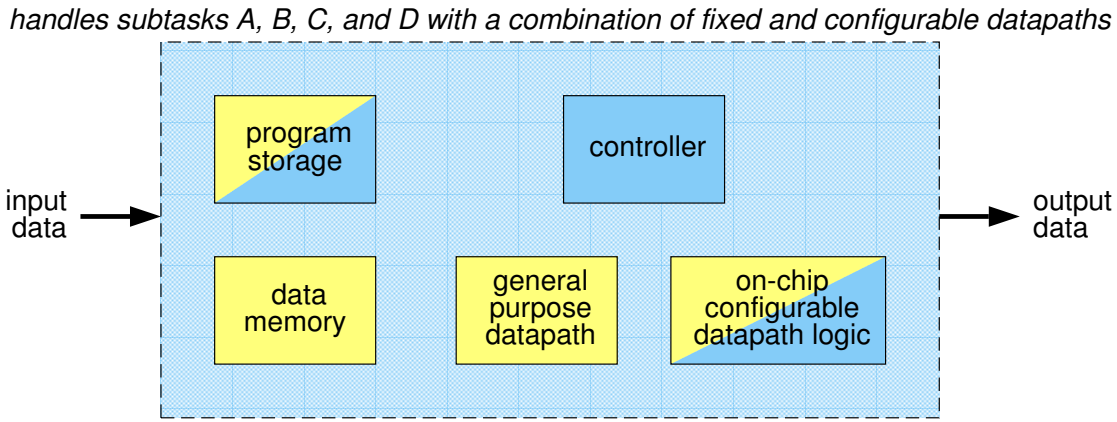


Figure: General-purpose processor with juxtaposed reconfigurable coprocessor.

General procedure:

1. Designers come up with a specific circuit structure for each major piece of suitable computation.
2. All configurations get stored in memory.
3. Whenever the host encounters a call to one of those computations, it downloads the pertaining configuration file into the FPL $\rightsquigarrow$ dead time!
4. Host feeds coprocessor with data and fetches results.
5. Host proceeds after computation completes.

5. Extendable instruction set processor (by Stretch Inc.)

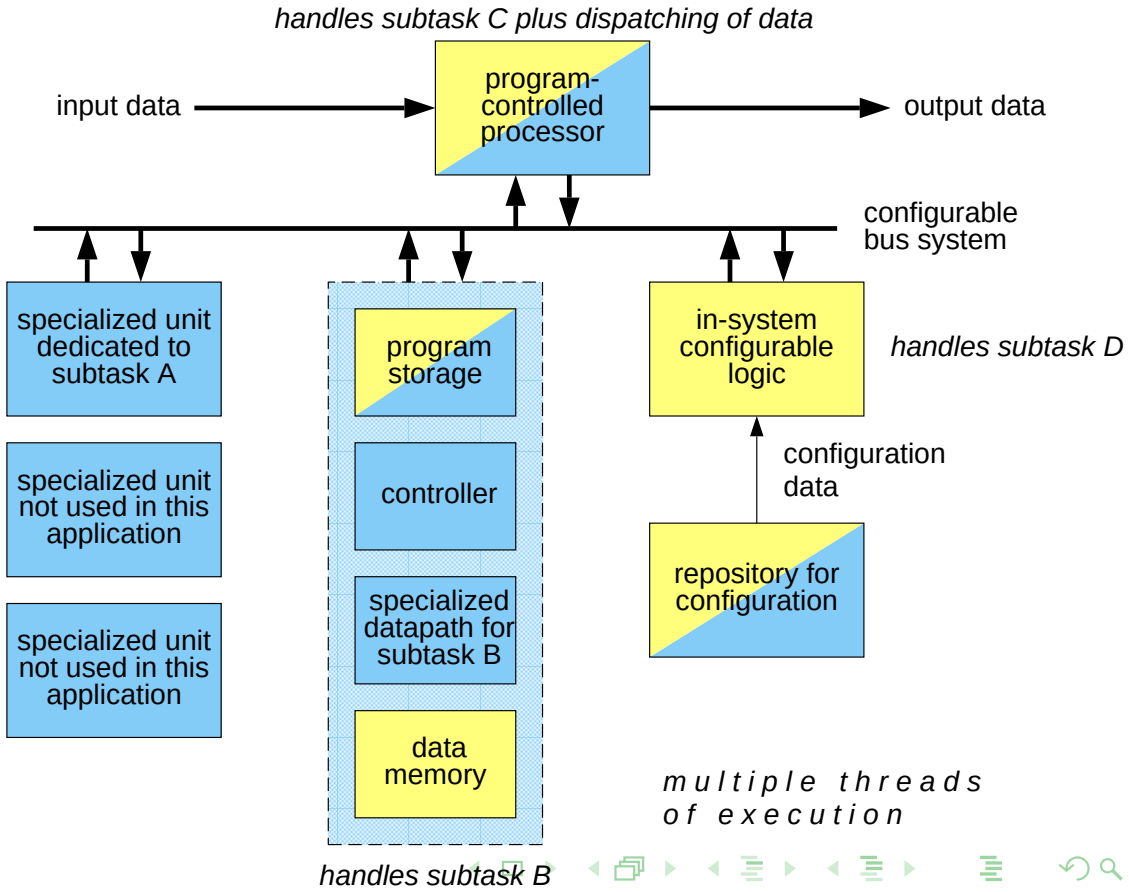


General procedure:

1. System developers write application programs in C or C++.
2. Proprietary EDA tools identify instruction sequences that are executed many times over (hot spots).
3. For each such sequence, reconfigurable logic is synthesized into a parallel computation network that completes within one clock cycle.
4. Each occurrence of the original instruction sequence gets replaced by a function call that activates the custom-made logic.

6. Domain-specific programmable platform (DSPP) (new)

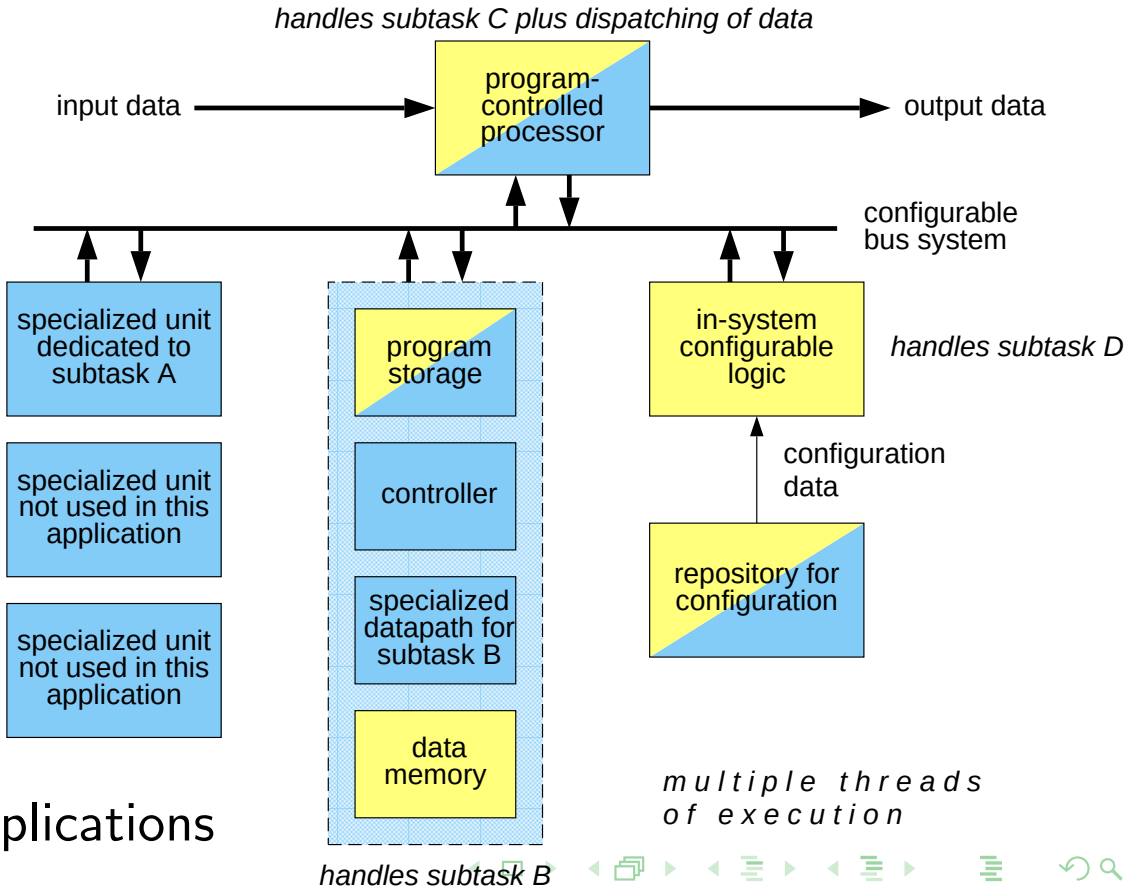
- Generous and heterogenous circuit resources in one malleable platform
- Specification using a domain-specific high-level language
- Developer tools assign most adequate execution units such as to meet performance target at minimum energy
- Little or no on-the-fly reconfiguration



6. Domain-specific programmable platform (DSPP) (new)

- Generous and heterogenous circuit resources in one malleable platform
- Specification using a domain-specific high-level language
- Developer tools assign most adequate execution units such as to meet performance target at minimum energy
- Little or no on-the-fly reconfiguration

+ good performance
+ energy-efficient
+ agile, fast turnaround
+ one design covers many applications



Reality check

- Platform ICs circuitry uses transistors lavishly,
  many subcircuits may never be used in a given application or product.
- Software tools are in their infancy (but design simplifies to platform selection and assignment of subfunctions to the on-chip resources).

Reality check

- Platform ICs circuitry uses transistors lavishly,
  many subcircuits may never be used in a given application or product.
- Software tools are in their infancy (but design simplifies to platform selection and assignment of subfunctions to the on-chip resources).

Technological progress tends to make such concerns less and less relevant.

- Viability stands or falls with the tool chain.
  - specification languages under development
  - standards required to ensure code reuse and portability
- In line with trends from general-purpose computing and high-end FPGAs.
  - costs per transistor $\downarrow$ mask costs $\uparrow$ verification costs $\uparrow$
  - energy-efficient computing has become a prime concern
  - CPU + GPU + FPL + fixed-function blocks + memory all on same chip

Reality check

- Platform ICs circuitry uses transistors lavishly,
  many subcircuits may never be used in a given application or product.
- Software tools are in their infancy (but design simplifies to platform selection and assignment of subfunctions to the on-chip resources).

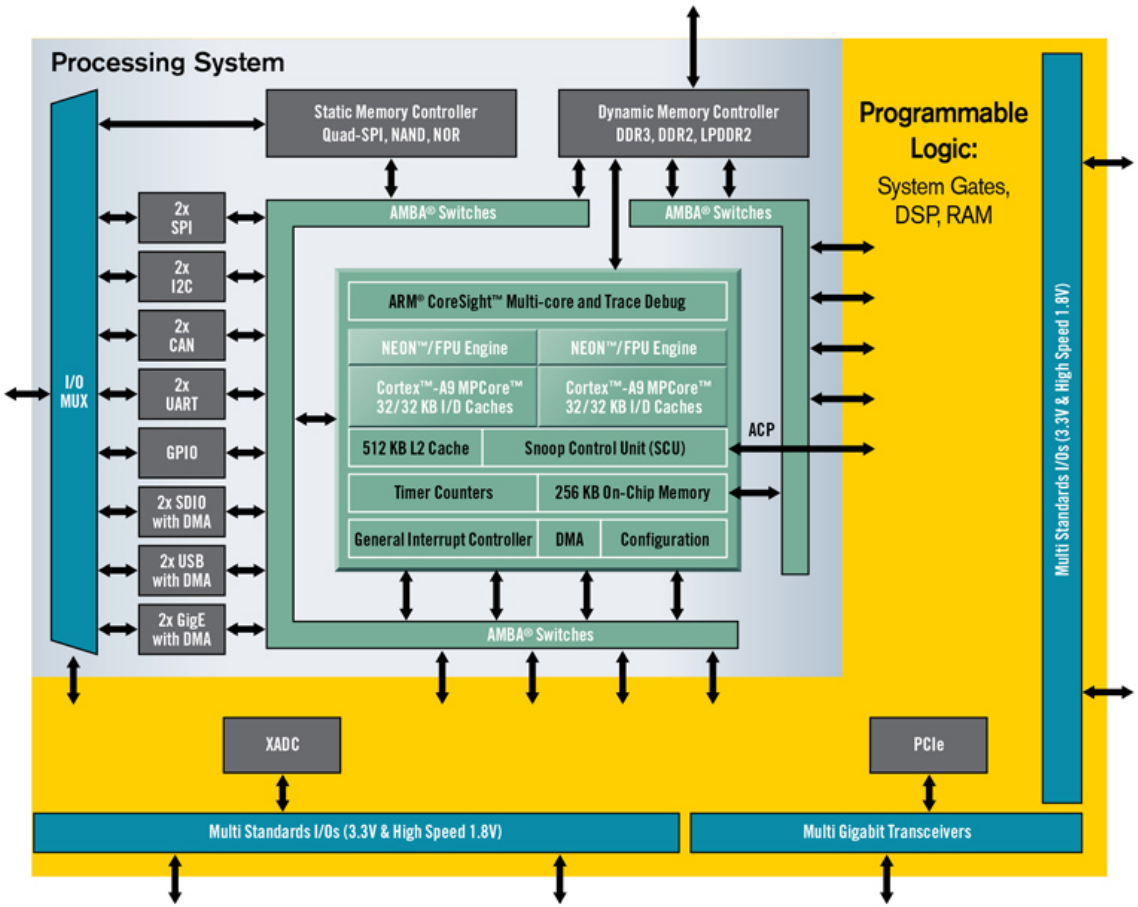
Technological progress tends to make such concerns less and less relevant.

- Viability stands or falls with the tool chain.
  - specification languages under development
  - standards required to ensure code reuse and portability
- In line with trends from general-purpose computing and high-end FPGAs.
  - costs per transistor ↓ mask costs ↑ verification costs ↑
  - energy-efficient computing has become a prime concern
  - CPU + GPU + FPL + fixed-function blocks + memory all on same chip

Conclusion

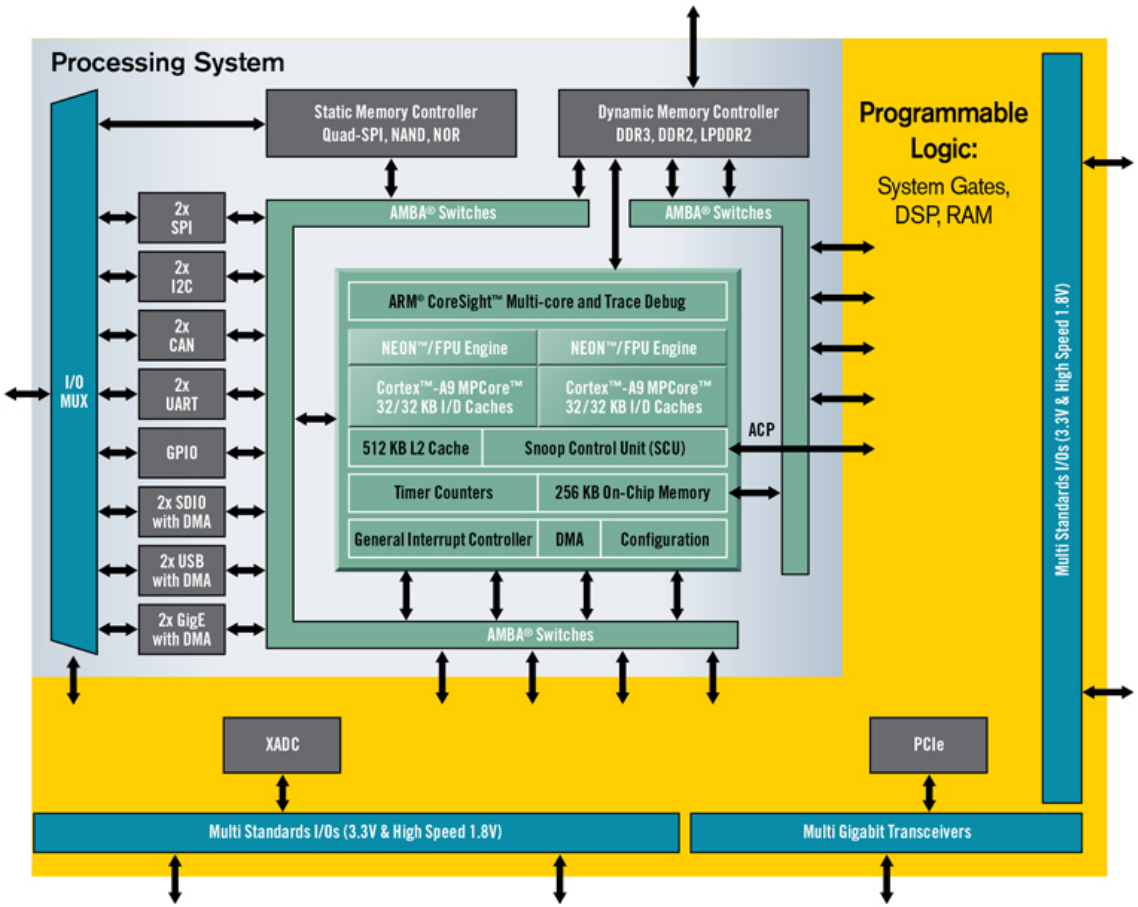
Much remains to be done before platform ICs can dominate digital VLSI,
but the concept benefits from numerous technological and economic trends.

Forerunner: Extensible Processing Platform (by Xilinx Inc.)



- Dual ARM Cortex™-A9 MPCore
  - Up to 800MHz
  - Enhanced with NEON Extension and Single & Double Precision Floating point unit
  - 32kB Instruction & 32kB Data L1 Cache
- Unified 512kB L2 Cache
- 256kB on-chip Memory
- DDR3, DDR2 and LPDDR2 Dynamic Memory Controller
- 2x QSPI, NAND Flash and NOR Flash Memory Controller
- 2x USB2.0 (OTG), 2x GbE, 2x CAN2,0B 2x SD/SDIO, 2x UART, 2x SPI, 2x I2C, 4x 32b GPIO
- AES & SHA 256b encryption engine for secure boot and secure configuration
- Dual 12bit 1Msps Analog-to-Digital converter
  - Up to 17 Differential Inputs
- Advanced Low Power 28nm Programmable Logic:
  - 28k to 235k Logic Cells (approximately 430k to 3.5M of equivalent ASIC Gates)
  - 240kB to 1.86MB of Extensible Block RAM
  - 80 to 760 18x25 DSP Slices (58 to 912 GMACS peak DSP performance)
- PCI Express® Gen2x8 (in largest devices)
- 154 to 404 User IOs (Multiplexed + SelectIO™)
- 4 to 12 10.3Gbps Transceivers (in largest devices)

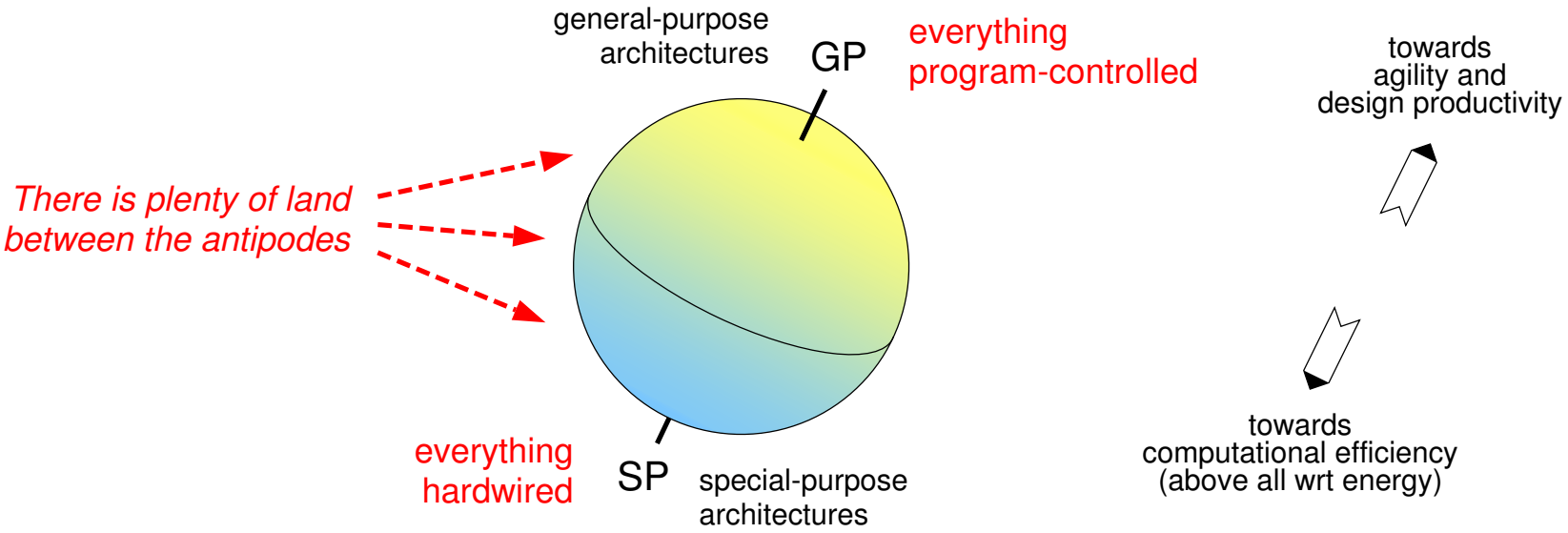
Forerunner: Extensible Processing Platform (by Xilinx Inc.)



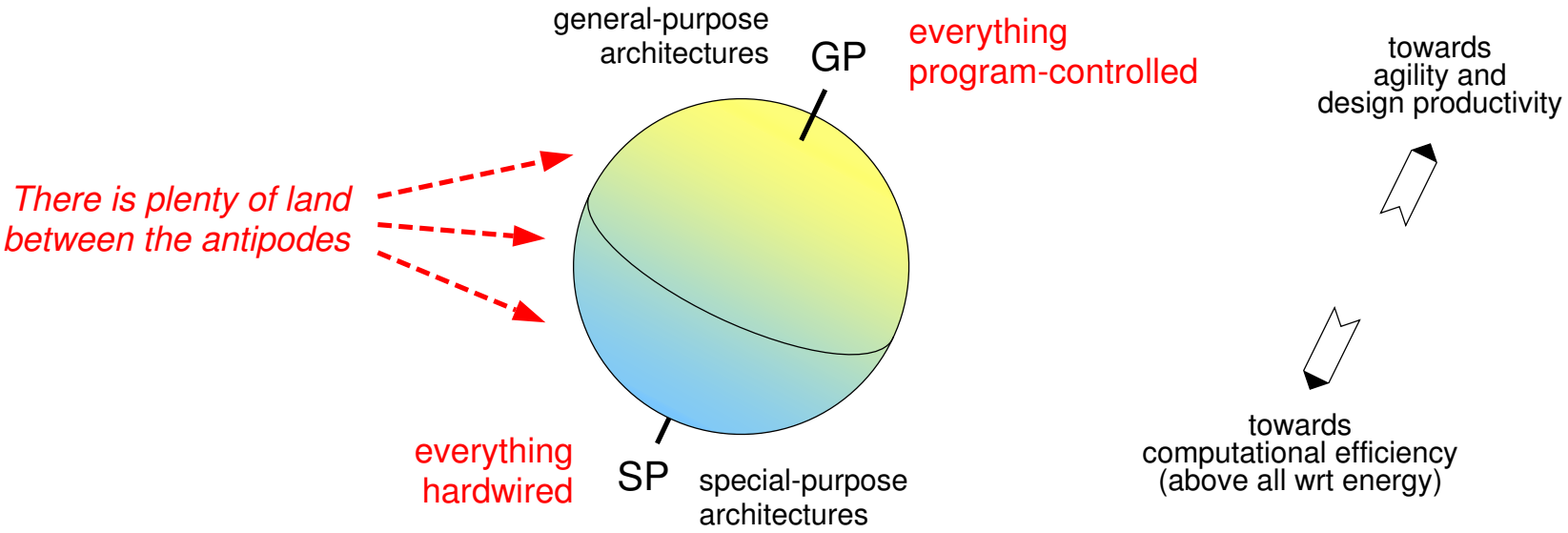
- Dual ARM Cortex™-A9 MPCore
  - Up to 800MHz
  - Enhanced with NEON Extension and Single & Double Precision Floating point unit
  - 32kB Instruction & 32kB Data L1 Cache
- Unified 512kB L2 Cache
- 256kB on-chip Memory
- DDR3, DDR2 and LPDDR2 Dynamic Memory Controller
- 2x QSPI, NAND Flash and NOR Flash Memory Controller
- 2x USB2.0 (OTG), 2x GbE, 2x CAN2,0B 2x SD/SDIO, 2x UART, 2x SPI, 2x I2C, 4x 32b GPIO
- AES & SHA 256b encryption engine for secure boot and secure configuration
- Dual 12bit 1Msps Analog-to-Digital converter
  - Up to 17 Differential Inputs
- Advanced Low Power 28nm Programmable Logic:
  - 28k to 235k Logic Cells (approximately 430k to 3.5M of equivalent ASIC Gates)
  - 240kB to 1.86MB of Extensible Block RAM
  - 80 to 760 18x25 DSP Slices (58 to 912 GMACS peak DSP performance)
- PCI Express® Gen2x8 (in largest devices)
- 154 to 404 User IOs (Multiplexed + SelectIO™)
- 4 to 12 10.3Gbps Transceivers (in largest devices)

"CPU and GPU cores are the new gates (EE Times 2011) ... and platform ICs are the new gate arrays (H. Kaeslin)."

Insight gained



Insight gained



Guideline

- Rely on dedicated hardware only for those subfunctions that are called many times and are unlikely to change.
- Keep the rest programmable (via software or reconfiguration).

The key options of architecture design

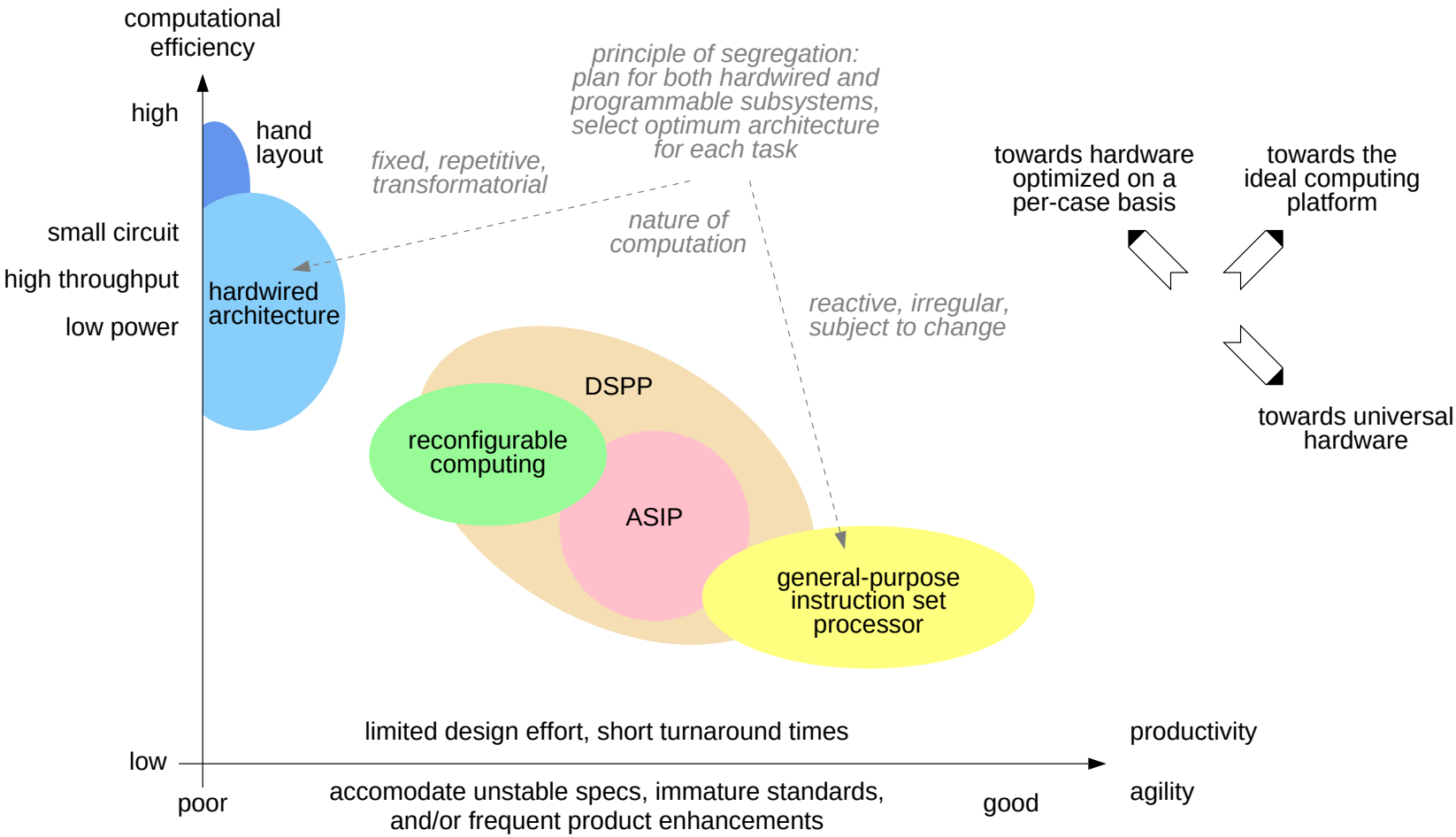


Figure: Tradeoffs between computational efficiency, agility, and design productivity.

Example: Yet another SoC

Note the coexistence of
- general-purpose processors
- ASIPs, and
- hardwired helper engines

on the same die.

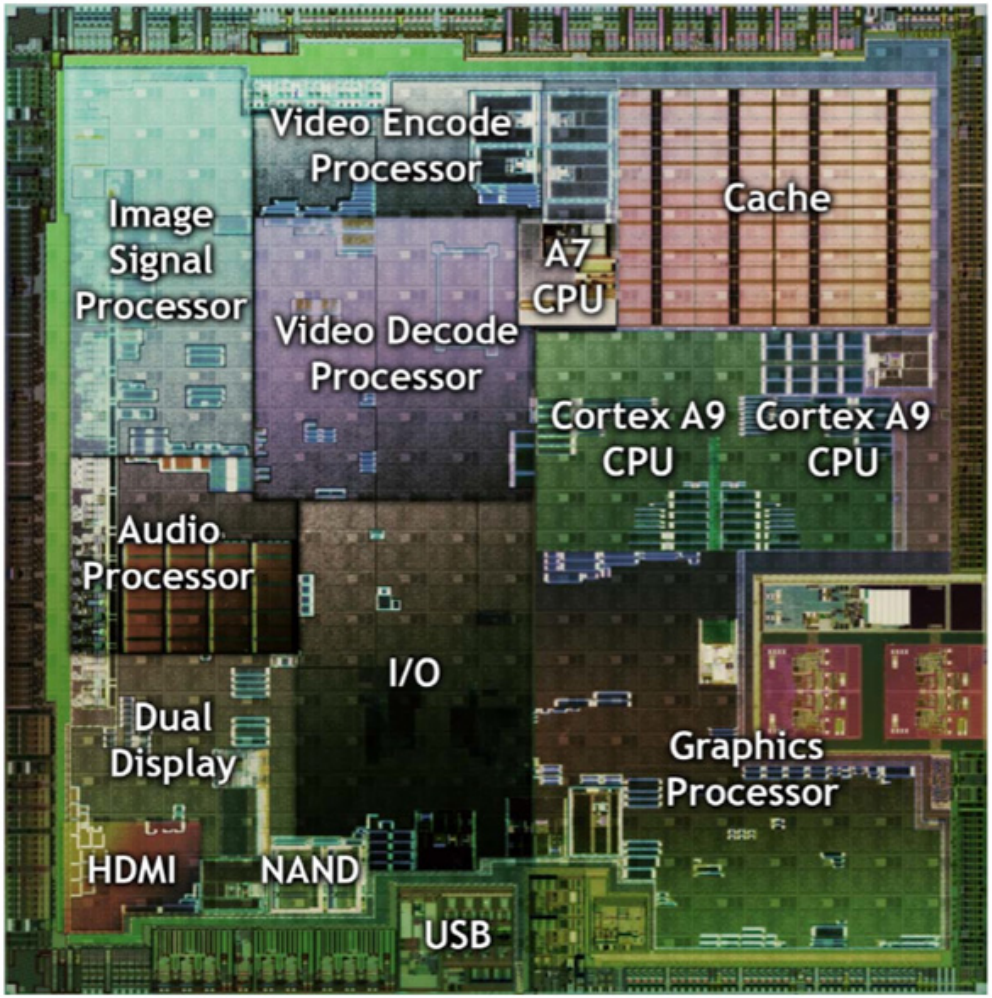


Figure: Tegra II chip for smartphones and tablet computers (source Nvidia).

Subject

How to design dedicated VLSI architectures

Why do we focus on dedicated architectures?

Many techniques for obtaining high performance at low cost are shared between general- and special-purpose architectures.

Yet, our emphasis is on dedicated architectures because

- A priori knowledge of a computational problem offers room for ideas that do not apply to instruction-set processors architectures.
- Utmost performance requirements often ask for special-purpose designs.
- Industry provides us with an extremely vast selection of micro- and signal processors so that proprietary designs are hard to justify.
- There exists a comprehensive literature on general-purpose architectures.

Most processing algos must be reworked for hardware I

Departures from some mathematically ideal algorithm are almost always necessary to arrive at an economically feasible solution. Examples follow.

Digital filter Tolerate a somewhat lower stopband suppression in exchange for a reduced computational burden.
(e.g. lower order, smaller coefficients replaced by zeros.)

Most processing algos must be reworked for hardware I

Departures from some mathematically ideal algorithm are almost always necessary to arrive at an economically feasible solution. Examples follow.

Digital filter Tolerate a somewhat lower stopband suppression in exchange for a reduced computational burden.
(e.g. lower order, smaller coefficients replaced by zeros.)

Viterbi decoder (for convolutional codes) Sacrifice 0.1 dB or so of coding gain for the benefit of doing computations in a more economic way.
(e.g. truncated dynamic range, frequent rescaling, restricted traceback.)

Most processing algos must be reworked for hardware II

Autocorrelation function

Replace computation of

$$ACF_{xx}(k) = r_{xx}(k) = \sum_{n=-\infty}^{\infty} x(n) \cdot x(n+k)$$

by the average magnitude difference function

$$AMDF_{xx}(k) = r'_{xx}(k) = \sum_{n=0}^{N-1} |x(n) - x(n+k)|$$

Most processing algos must be reworked for hardware III

Magnitude function

- Approximated with shift, add and compare.

Name	aka	Formula
lesser	$\ell^{-\infty}$-norm	$l = \min(\lvert a\rvert, \lvert b\rvert)$
sum	ℓ^1-norm	$s = \lvert a\rvert + \lvert b\rvert$
magnitude (reference)	ℓ^2-norm	$m = \sqrt{a^2 + b^2}$
greater	ℓ^∞-norm	$g = \max(\lvert a\rvert, \lvert b\rvert)$
Approximation 1		$m \approx \frac{3}{8}s + \frac{5}{8}g$
Approximation 2		$m \approx \max(g, \frac{7}{8}g + \frac{1}{2}l)$

- Simply replaced by ℓ^1- or ℓ^∞-norm.
  (finds applications in MIMO decoders, for instance.)

Finding an optimal hardware organization

Guideline

There is room for remodelling computations in two distinct domains:

- Processing algorithm.
- Hardware architecture.

Finding an optimal hardware organization

Guideline

There is room for remodelling computations in two distinct domains:

- Processing algorithm.
- Hardware architecture.

Alternative choices in the algorithmic domain. How to tailor an algorithm such as to cut the computational burden, to trim down memory requirements, and/or to speed up calculations without incurring unacceptable implementation losses?

Finding an optimal hardware organization

Guideline

There is room for remodelling computations in two distinct domains:

- Processing algorithm.
- Hardware architecture.

Alternative choices in the algorithmic domain. How to tailor an algorithm such as to cut the computational burden, to trim down memory requirements, and/or to speed up calculations without incurring unacceptable implementation losses?

Equivalence transforms in the architectural domain. How to (re)organize a computation such as to optimize throughput, circuit size, energy efficiency and overall costs while leaving the input-to-output relationship unchanged except, possibly, for latency?

Systems engineers and VLSI designers must collaborate

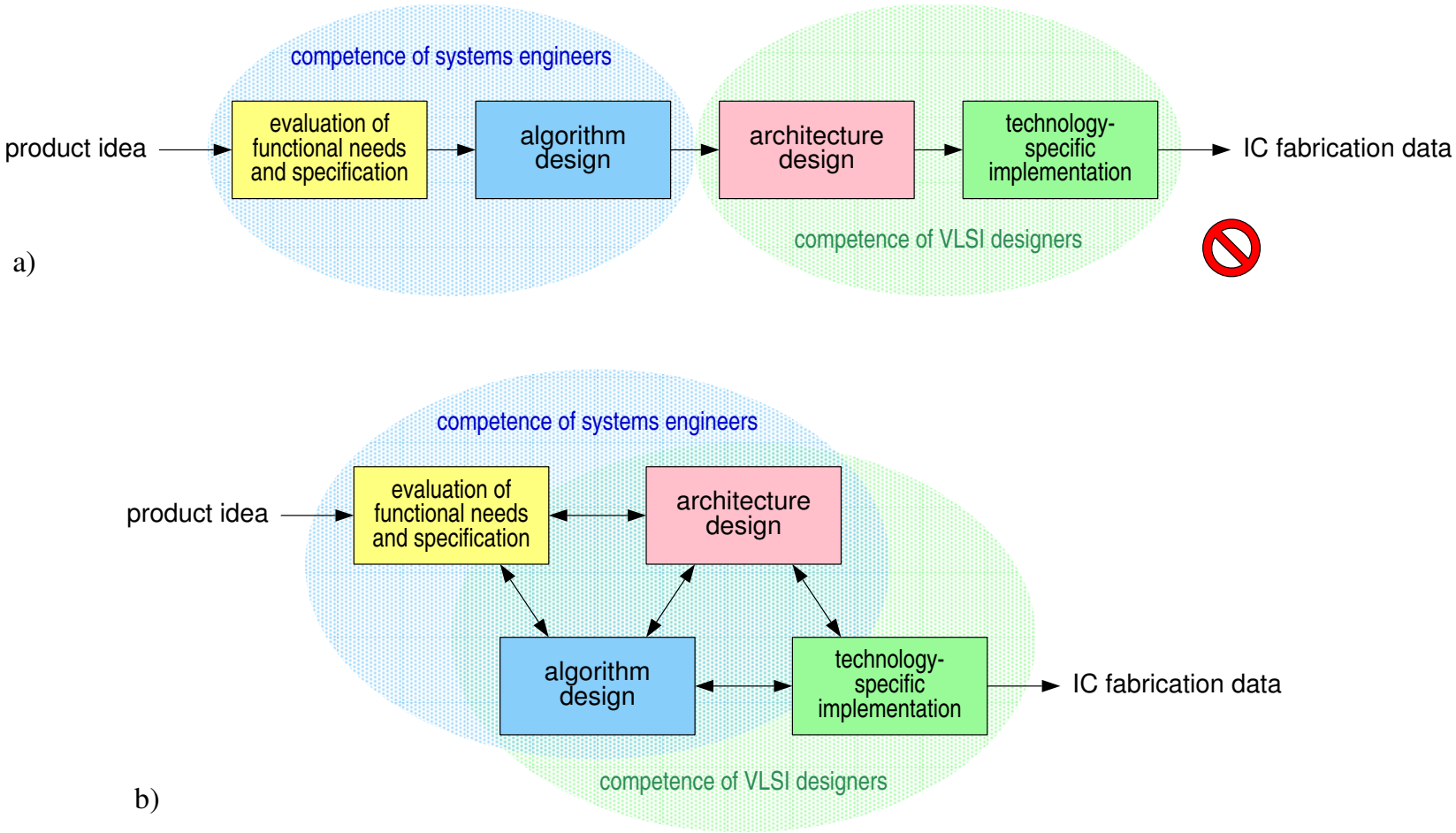


Figure: Sequential thinking (a) versus networked team (b).

Insight gained

Observation

It is always necessary to balance many contradicting requirements to arrive at a working and marketable embodiment of an algorithm.

- There is more to VLSI design than accepting a given algorithm and turning that into hardware with the aid of some HDL synthesizer.
- Algorithm design is not covered in this course, but nevertheless extremely important for VLSI design.

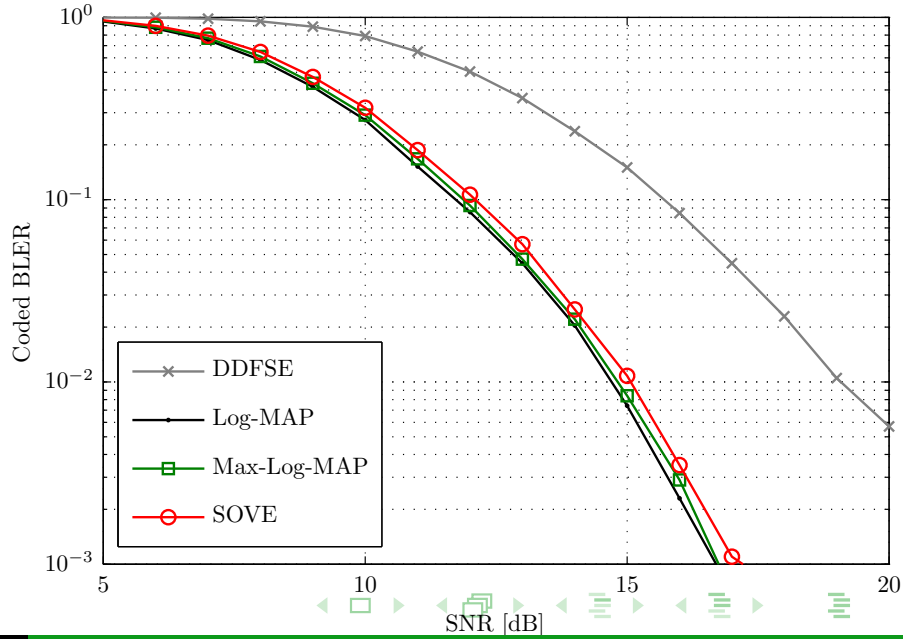
Example: Sequence estimation for EDGE receiver

Algorithm	Delayed decision feedback	Max-log-MAP	Soft output Viterbi equalizer
Soft output	no	yes	yes
Forward recursion	yes	yes	yes
Backward recursion	no	yes	no
Backtracking step	yes	no	no
Memory requirements	1x	50x	0.13x

Key design targets:

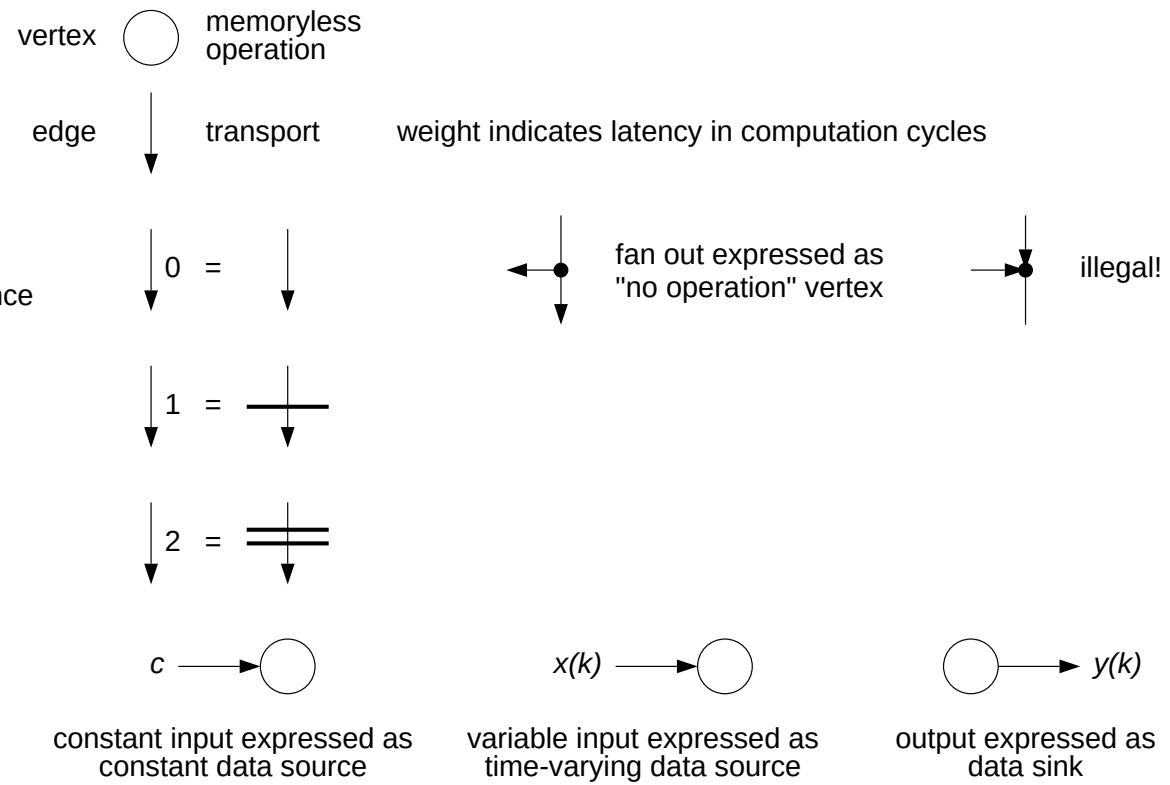
- soft output
- less than 577 μs per burst
- small circuit, low power
- min. block error rate at any given signal-to-noise ratio

Which option would you go for?



Data dependency graphs (DDG)

Definitions



Shorthand notations

introduced for convenience

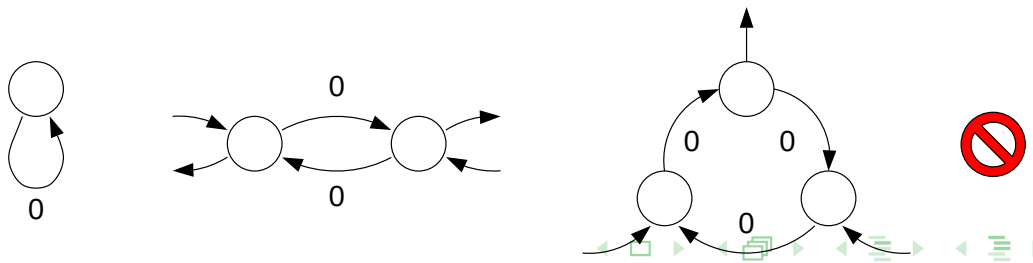
constant input expressed as constant data source

variable input expressed as time-varying data source

output expressed as data sink

Danger of race conditions

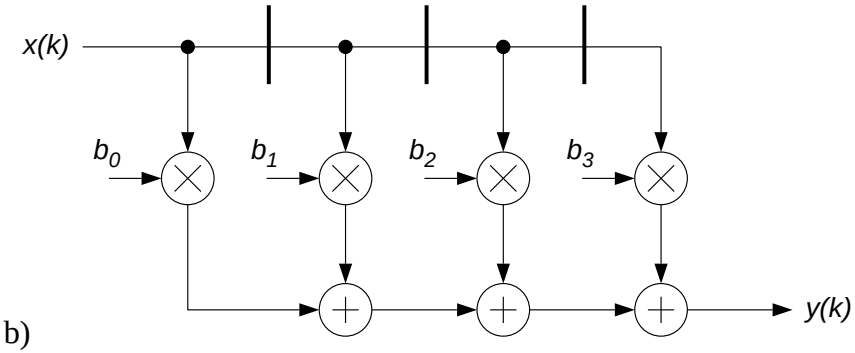
circular paths
of edge weight zero
are not admitted!



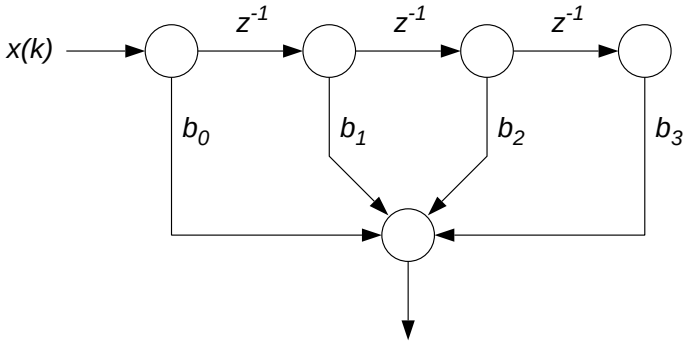
The isomorphic architecture

$$y(k) = \sum_{n=0}^{N=3} b_n x(k-n)$$

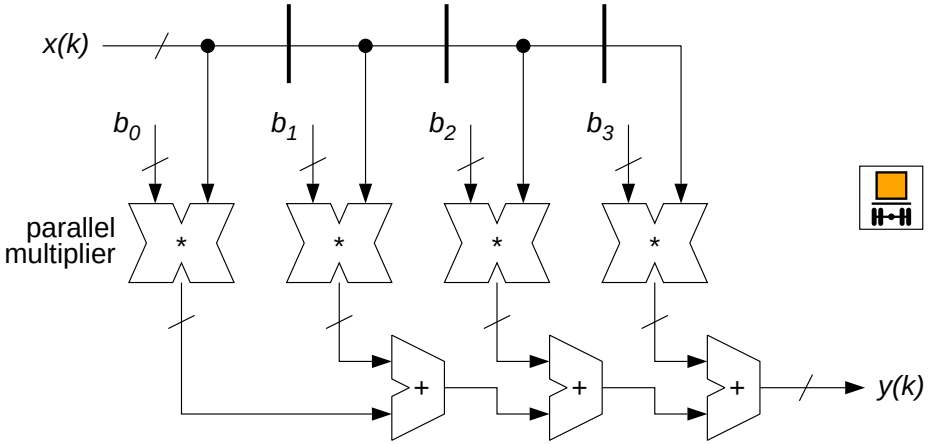
a)



b)



c)



d)

Figure: Example: A third order transversal filter in various notations. Equation (a), DDG (b), and isomorphic architecture (d). SFG for comparison (c).

Figures of merit for hardware architectures I (Perform.-related)

Cycles per data item Γ , number of computation cycles between releasing two subsequent data items.

Longest path delay t_{lp} , the lapse of time required for data to propagate along the longest path. A circuit cannot function correctly unless $t_{lp} \leq T_{cp}$.

Figures of merit for hardware architectures I (Perform.-related)

Cycles per data item Γ , number of computation cycles between releasing two subsequent data items.

Longest path delay t_{lp} , the lapse of time required for data to propagate along the longest path. A circuit cannot function correctly unless $t_{lp} \leq T_{cp}$.

Time per data item T , the lapse of time between releasing two subsequent data items, e.g. in μs/sample, ms/frame, or s/computation. $T = \Gamma \cdot T_{cp} \geq \Gamma \cdot t_{lp}$.

Data throughput $\Theta = \frac{1}{T} = \frac{f_{cp}}{\Gamma}$ expressed in pixel/s, sample/s, frame/s, record/s, FFT/s, or the like.

Figures of merit for hardware architectures I (Perform.-related)

Cycles per data item Γ , number of computation cycles between releasing two subsequent data items.

Longest path delay t_{lp} , the lapse of time required for data to propagate along the longest path. A circuit cannot function correctly unless $t_{lp} \leq T_{cp}$.

Time per data item T , the lapse of time between releasing two subsequent data items, e.g. in μs/sample, ms/frame, or s/computation. $T = \Gamma \cdot T_{cp} \geq \Gamma \cdot t_{lp}$.

Data throughput $\Theta = \frac{1}{T} = \frac{f_{cp}}{\Gamma}$ expressed in pixel/s, sample/s, frame/s, record/s, FFT/s, or the like.

Latency L , number of computation cycles from a data item entering a circuit until the pertaining result becomes available.

Figures of merit for hardware architectures II (Cost-related)

Circuit size A expressed in mm^2, F^2 or GE (gate equivalent).

Size-time product AT , the hardware resources spent to obtain a given throughput. $AT = \frac{A}{\Theta}$.

Figures of merit for hardware architectures II (Cost-related)

Circuit size A expressed in mm^2, F^2 or GE (gate equivalent).

Size-time product AT , the hardware resources spent to obtain a given throughput. $AT = \frac{A}{\Theta}$.

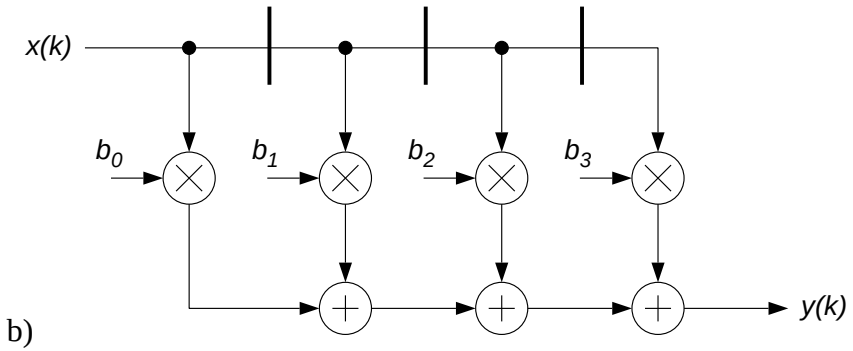
Energy per data item E , the amount of energy dissipated for a given computation on a data item e.g. in pJ/MAC, nJ/sample, μJ/datablock, or in mWs/frame.

Can also be understood as power-per-throughput ratio $E = \frac{P}{\Theta}$ measured in $\mathrm{mW}/\frac{\mathrm{Mbit}}{\mathrm{s}}$ or W/GOPS.

because $\frac{\text{energy}}{\text{data item}} = \frac{\text{energy per second}}{\text{data item per second}} = \frac{\text{power}}{\text{throughput}}$

Energy-time product ET indicates how much energy gets spent for achieving a given throughput (synonym "energy-per-throughput ratio"). $ET = \frac{E}{\Theta} = \frac{P}{\Theta^2}$, e.g. in $\mu\mathrm{J}/\frac{\mathrm{datablock}}{\mathrm{s}}$ or mWs^2/videoframe.

Example



b)

Approximations

- Interconnect delays neglected (overly optimistic).
- Delays of arithmetic operations summed up (sometimes pessimistic).
- Glitching ignored (optimistic).

$$A = 3A_{reg} + 4A_{*} + 3A_{+}$$
$$\Gamma = 1$$
$$t_{lp} = t_{reg} + t_{*} + 3t_{+}$$
$$AT = (3A_{reg} + 4A_{*} + 3A_{+})(t_{reg} + t_{*} + 3t_{+})$$
$$L = 0$$
$$E = 3E_{reg} + 4E_{*} + 3E_{+}$$

A symbolic representation of hardware

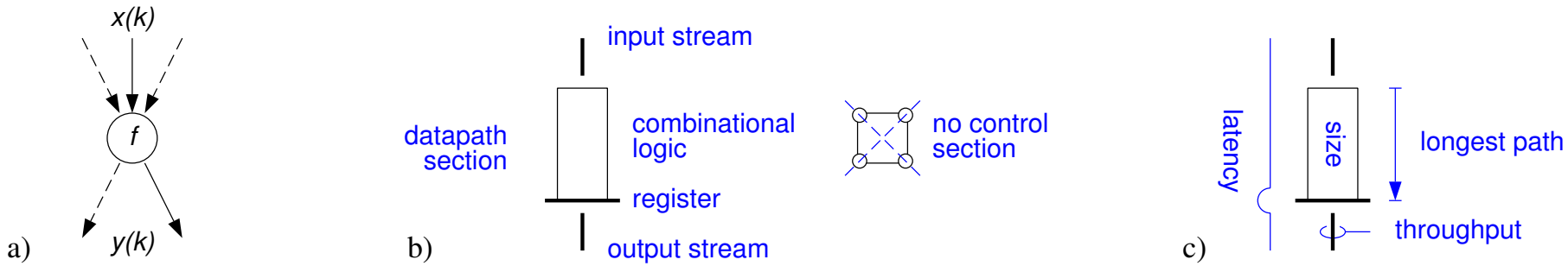


Figure: DDG (a), reference hardware configuration (b), key characteristics (c).

Reference hardware = isomorphic architecture + output register(s)

Computation cycle versus clock period

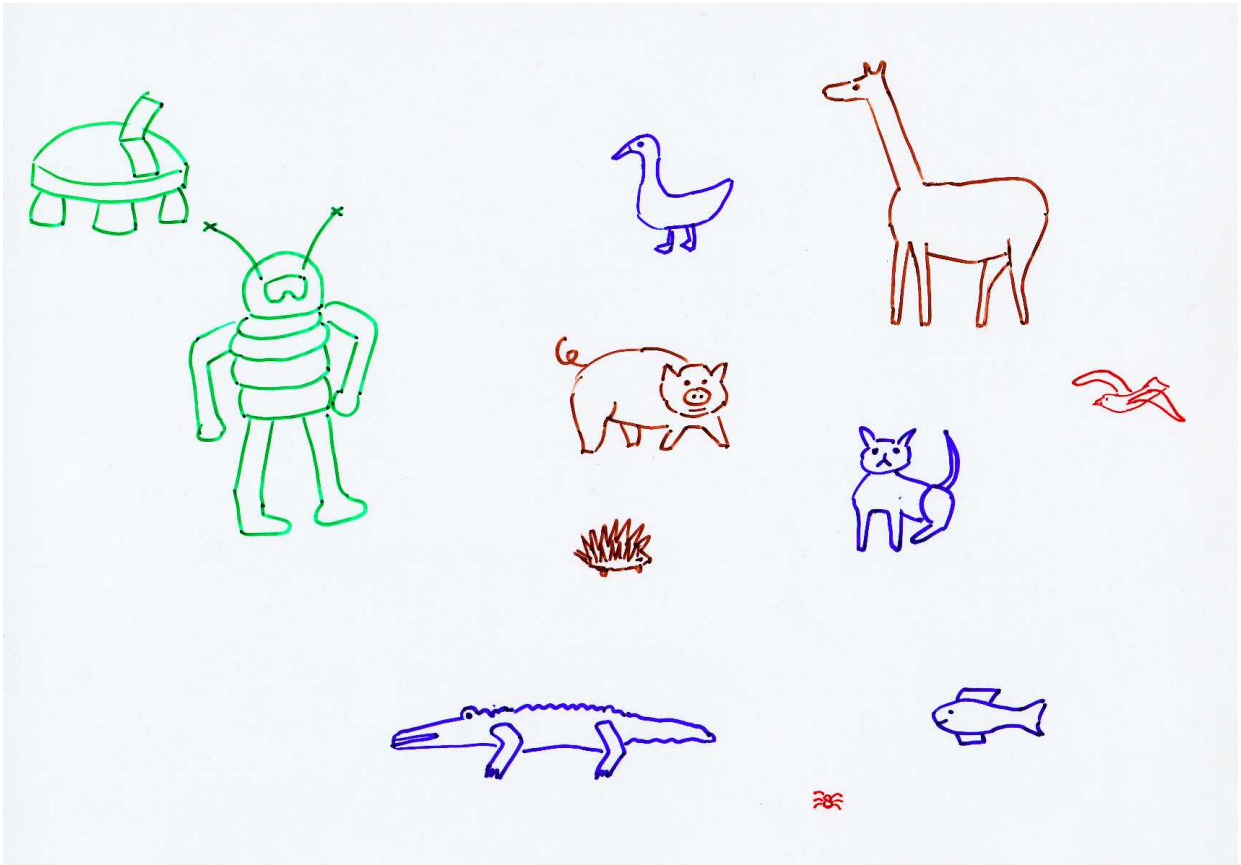
- A computation period T_{cp} is the time span that separates two consecutive computation cycles.
- During each computation cycle, fresh data emanate from a register, propagate through combinational circuitry before the result gets stored in the next analogous register.
- It is the combinational circuitry that performs all arithmetic, logic, and data routing operations.
- Computation rate $f_{cp} = \frac{1}{T_{cp}}$ denotes the inverse.
- For all circuits that adhere to single-edge-triggered one-phase clocking, computation cycle and clock period are the same.

$$f_{cp} = f_{clk} \quad \Leftrightarrow \quad T_{cp} = T_{clk}$$

Subject

Transforms for combinational computations

Darwin stepping off the boat at Galapagos



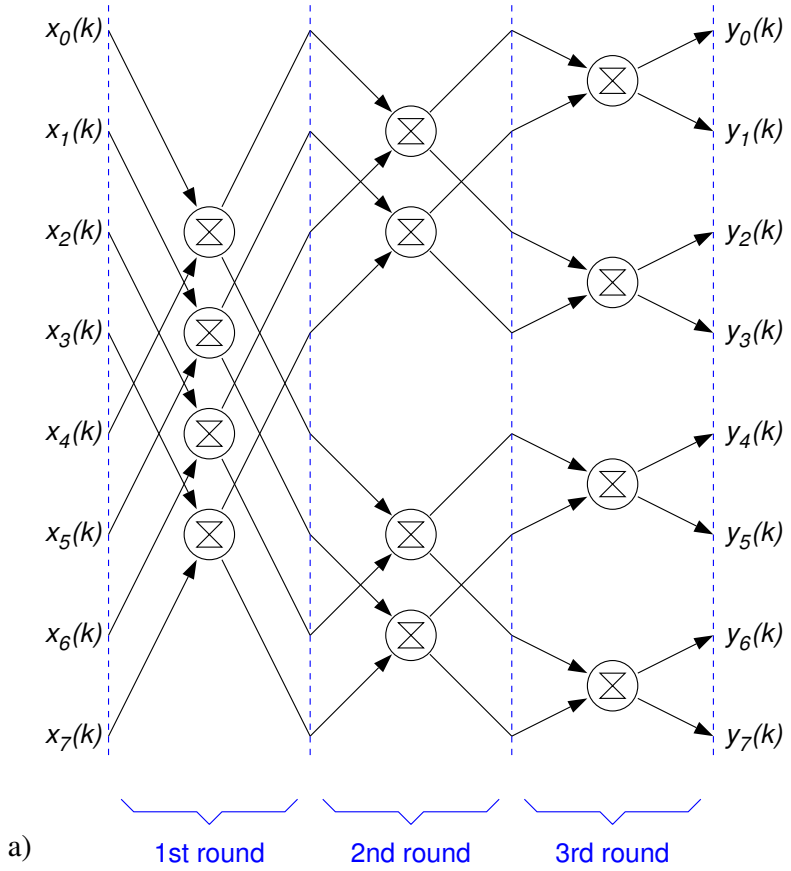
⇝ Diversity and evolution in biology suggest a transform approach to VLSI architecture design.

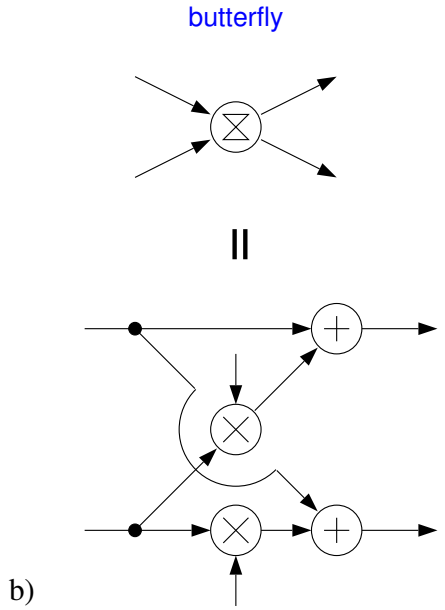
What do we mean by combinational computation?

A computation is termed combinational if

- Result depends on the present arguments exclusively.
- All edges in the DDG have weight zero.
- DDG is free of circular paths.

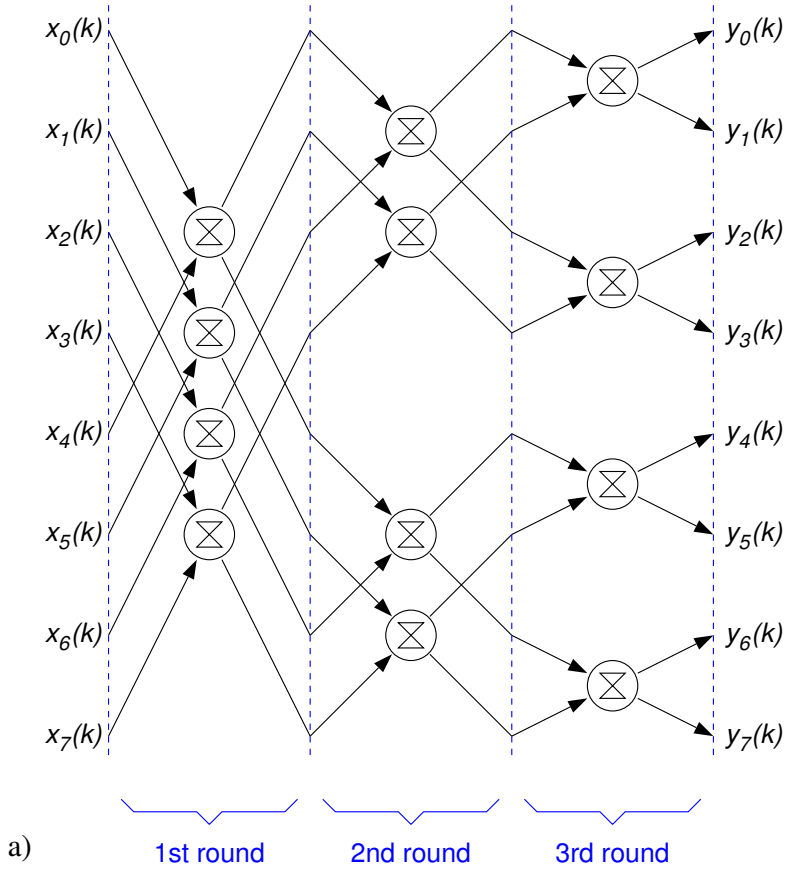
Example: 8-point FFT

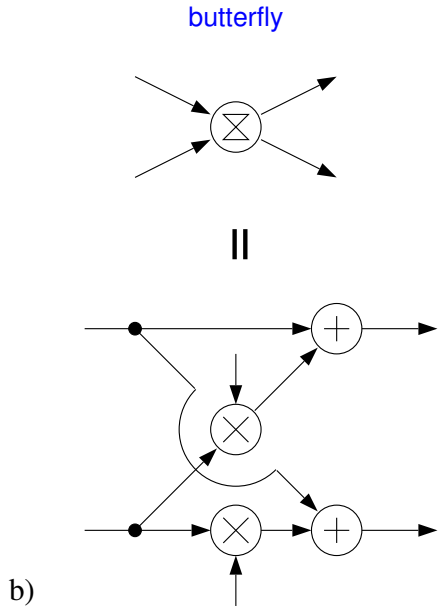




If the combinational function f complex ($8 \ll n$-point FFT, AES, JPEG) then

Example: 8-point FFT





If the combinational function f complex ($8 \ll n$-point FFT, AES, JPEG) then the isomorphic architecture is a rather expensive proposition.

Architectural options

Three options for improving this unsophisticated arrangement exist:

Decomposing function f into a sequence of subfunctions that get executed one after the other on same hardware.

Pipelining of the functional unit for f to improve computation rate by cutting down combinational depth.

Replicating the hardware for f and having all units work concurrently.

Architectural options

Three options for improving this unsophisticated arrangement exist:

Decomposing function f into a sequence of subfunctions that get executed one after the other on same hardware.

Pipelining of the functional unit for f to improve computation rate by cutting down combinational depth.

Replicating the hardware for f and having all units work concurrently.

Open questions:

- Does it make sense to combine pipelining with iterative decomposition in spite of their contrarian effects?
- How do replication and pipelining compare?
  Are there situations where one should be preferred over the other?

Iterative decomposition

Paradigm: Step-by-step execution

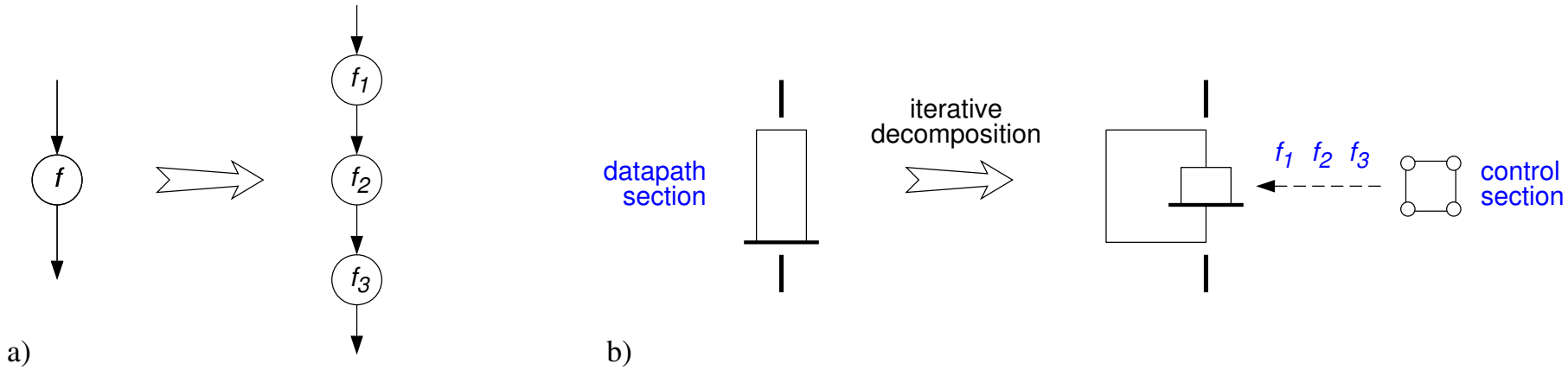


Figure: DDG (a) and hardware configuration for $d = 3$ (b).

Performance and cost analysis

As a first-order approximation, iterative decomposition by a factor of d leads to the following figures of merit:

$$\frac{A_f}{d} + A_{reg} + A_{ctl} \leq A(d) \leq A_f + A_{reg} + A_{ctl}$$

$$\Gamma(d) = d$$

$$t_{lp}(d) \approx \frac{t_f}{d} + t_{reg}$$

$$d(A_{reg} + A_{ctl})t_{reg} + (A_{reg} + A_{ctl})t_f + A_f t_{reg} + \frac{1}{d}A_f t_f \leq AT(d) \leq d(A_f + A_{reg} + A_{ctl})t_{reg} + (A_f + A_{reg} + A_{ctl})t_f$$

$$L(d) = d$$

$$E(d) \geqslant E_f + E_{reg}$$

Insight gained

Iterative decomposition

- Is attractive when a computation makes repetitive use of a single subfunction because a lot of area can then be saved.

  Example: multiplication $\mapsto$ repeated shift & add operations

- Is unattractive when subfunctions are very disparate and, therefore, cannot be made to share much hardware resources.

  Example: square root, logarithm, multiplication modulo some prime

Insight gained

Iterative decomposition

- Is attractive when a computation makes repetitive use of a single subfunction because a lot of area can then be saved.

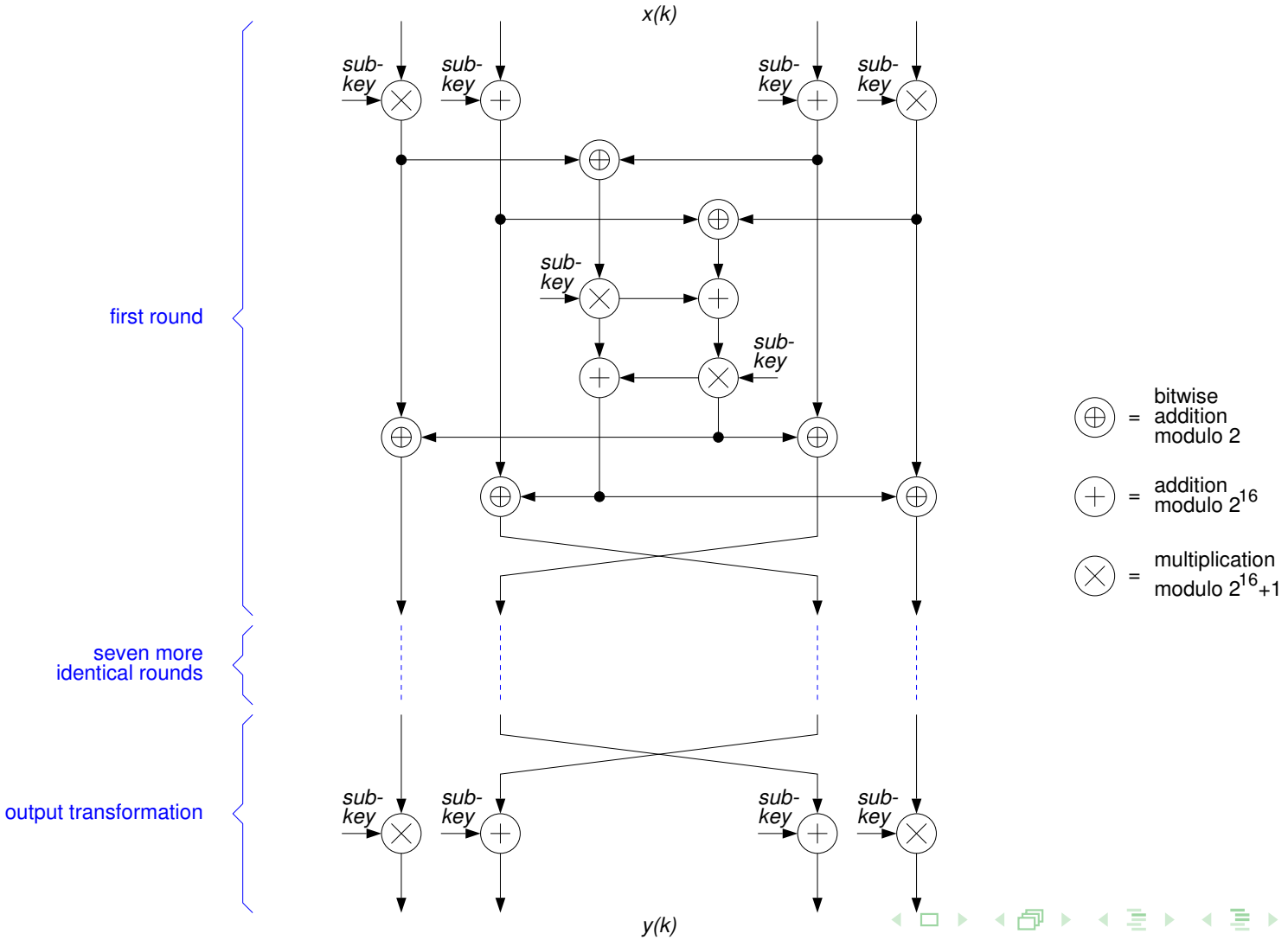
  Example: multiplication $\mapsto$ repeated shift & add operations

- Is unattractive when subfunctions are very disparate and, therefore, cannot be made to share much hardware resources.

  Example: square root, logarithm, multiplication modulo some prime

- Does not impact throughput much as long as $t_{reg} \ll t_{lp}$.
- May or may not improve energy efficiency.
  - yes, if cutting overly long signal propagation paths mitigates excessive glitching and the associated energy losses.
  - no, if the extra activity of data registers, control logic, and data recycling circuitry dominates.

Example: block cipher IDEA



Pipelining

Paradigm: Assembly line operated by specialized workers

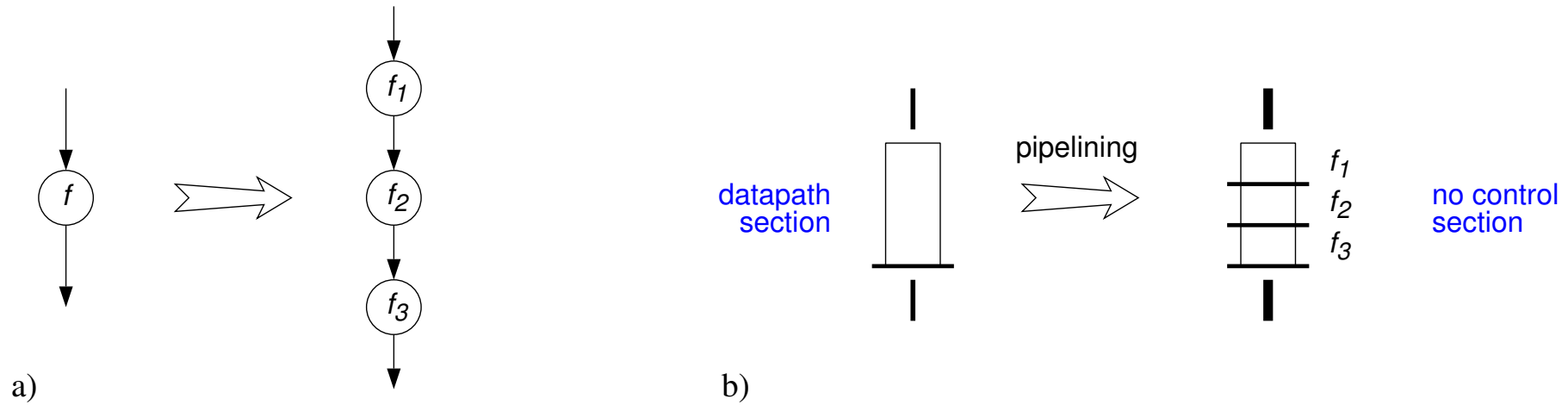


Figure: DDG (a) and hardware configuration for $p = 3$ (b).

Performance and cost analysis

Pipelining by a factor of p changes performance and cost figures as follows

$$A(p) = A_f + pA_{reg}$$

$$\Gamma(p) = 1$$

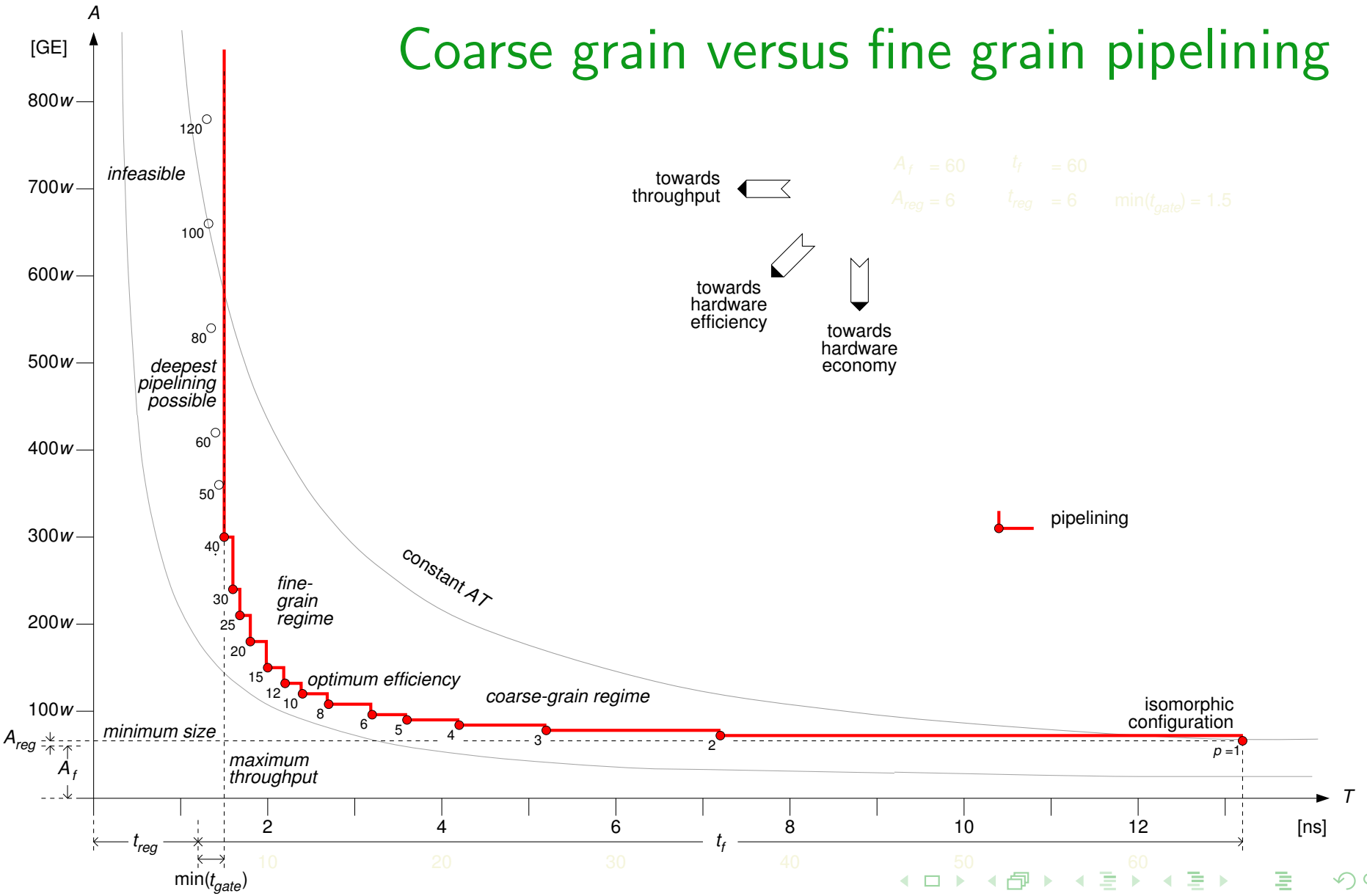
$$t_{lp}(p) \approx \frac{t_f}{p} + t_{reg}$$

$$AT(p) \approx pA_{reg}\,t_{reg} + (A_{reg}\,t_f + A_f\,t_{reg}) + \frac{1}{p}A_f\,t_f$$

$$L(p) = p$$

$$E(p) \underset{\text{coarse grain}}{\overset{\text{fine grain}}{\gtrless}} E_f + E_{reg}$$

Coarse grain versus fine grain pipelining



Insight gained

Must distinguish between two regimes of pipelining:

Coarse grain pipelining.

Few registers evenly inserted into a deep combinational network.

- \+ Little extra area for much better throughput.
- \+ AT-product lowered dramatically.
- \+ Long reconvergent fanout paths cut $\rightsquigarrow$ reduced glitching.

Insight gained

Must distinguish between two regimes of pipelining:

Coarse grain pipelining.

Few registers evenly inserted into a deep combinational network.

- $+$ Little extra area for much better throughput.
- $+$ AT-product lowered dramatically.
- $+$ Long reconvergent fanout paths cut $\rightsquigarrow$ reduced glitching.

Fine grain pipelining.

Combinational delay in each stage approaches register delay.

- $\sim$ Diminishing speedup for more and more overhead.
- $-$ AT-product augments significantly.
- $-$ Significant register activity added $\rightsquigarrow$ waste of energy.

Theoretical bound

- Pipeline stage must accomodate at least one 2-input NAND or NOR.
  $\rightsquigarrow$ Computation rate and clock frequency are bounded.

$$T_{cp} \geq \min(t_{lp}) = \min(t_{gate}) + t_{reg} = \min(t_{nand}, t_{nor}) + t_{su\,ff} + t_{pd\,ff}$$

Numerical example:

- Standard cell library for a 130 nm CMOS process.
- Computation period bounded from below to

$$T_{cp} \geq t_{\mathrm{NAN2D1}} + t_{\mathrm{DFFPB1}} = 18\ \mathrm{ps} + 249\ \mathrm{ps} \approx 267\ \mathrm{ps}$$

  $\rightsquigarrow$ Absolute maximum computation rate $\approx$ 3.7 GHz.

A side glance at microprocessors I

CPU	year	clock [MHz]	FO4 inverter delays per pipeline stage
Intel 80386	1989	33	≈ 80
Intel Pentium 4	2003	3200	12...16
Core 2 Duo	2007	2167	≈ 40
Core i7 980X	2011	3333...3600	42...46
IBM POWER5	2004	1650...1900	22
IBM POWER6	2007	3500...5000	13
IBM Cell Processor	2006	3200	11

FO4 = fanout 4

A side glance at microprocessors I

CPU	year	clock [MHz]	FO4 inverter delays per pipeline stage
Intel 80386	1989	33	≈ 80
Intel Pentium 4	2003	3200	12...16
Core 2 Duo	2007	2167	≈ 40
Core i7 980X	2011	3333...3600	42...46
IBM POWER5	2004	1650...1900	22
IBM POWER6	2007	3500...5000	13
IBM Cell Processor	2006	3200	11

FO4 = fanout 4

FO4 is a delay metric defined as the delay of an inverter, driven by an inverter 4x smaller than itself, and driving an inverter 4x larger than itself.

Observations

- Pipelining has been instrumental in pushing processor clock frequencies.
- 12 or so FO4 inverter delays per stage is close to practical limit.
- Trend towards ever deeper pipelines reversed in the Intel Core family to reclaim energy efficiency.

A side glance at microprocessors II

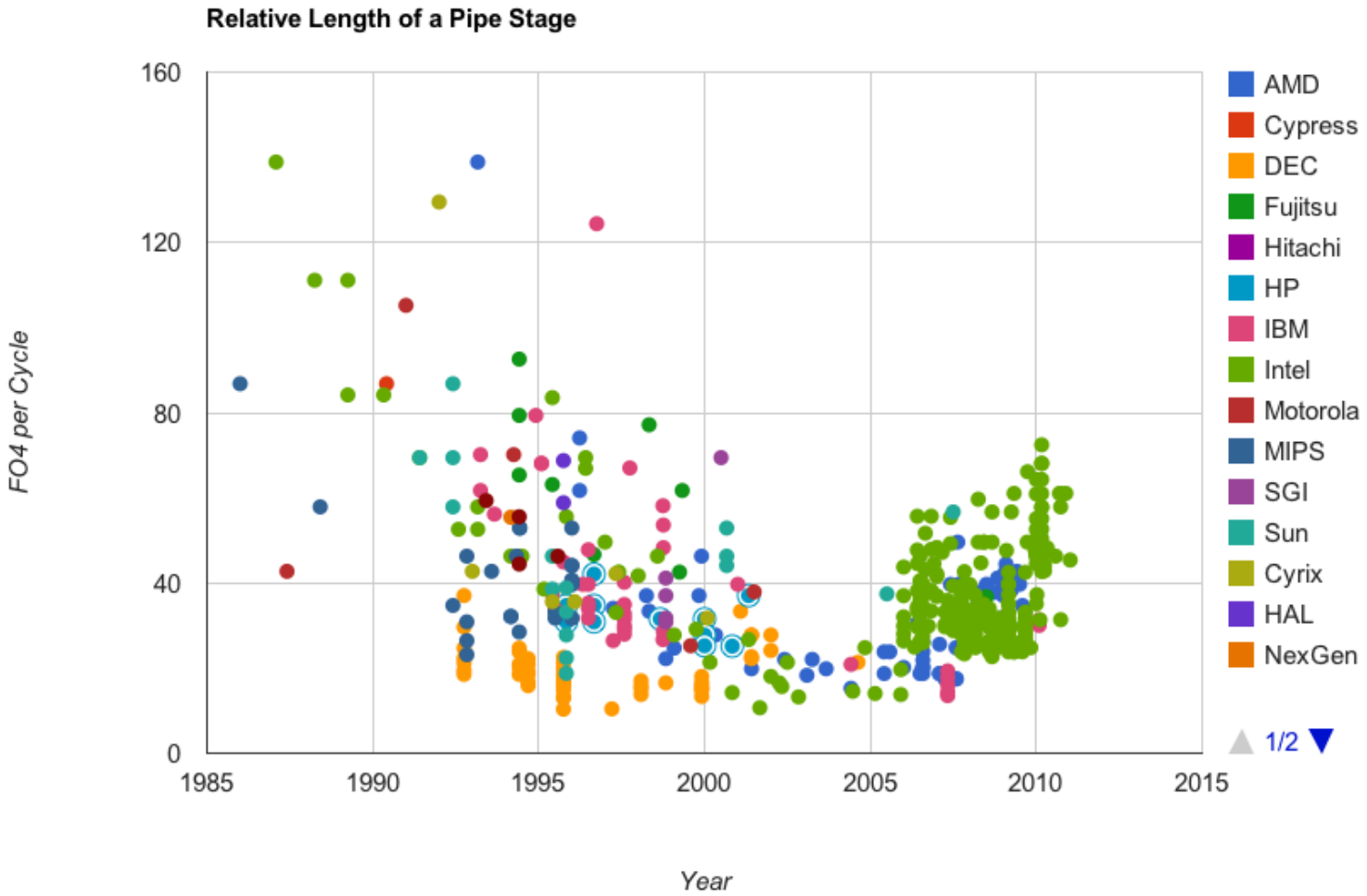


Figure: Evolution of pipeline depth over the years (source Stanford CPU database).

Pipelining in the presence of multiple feedforward paths

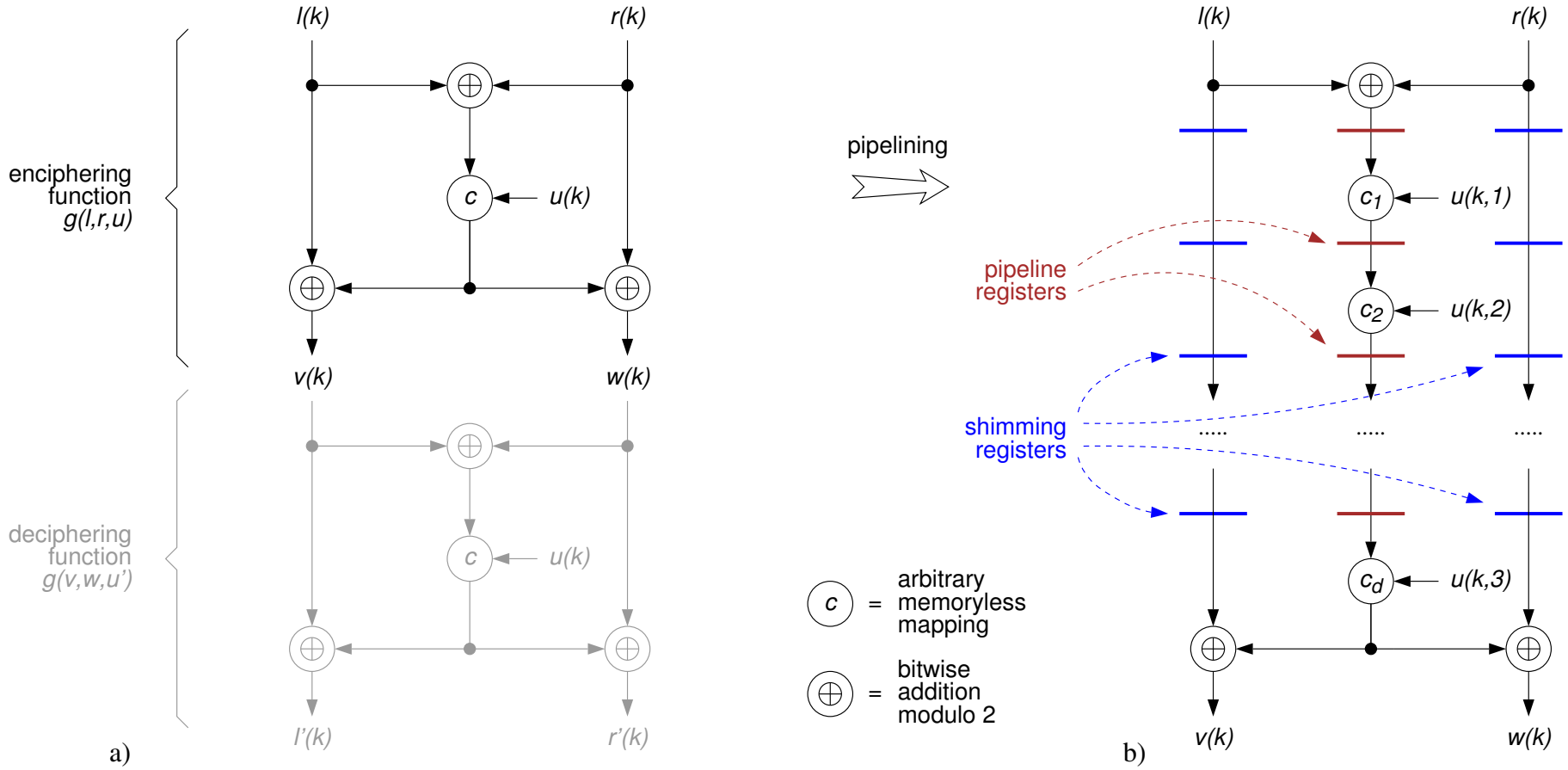


Figure: Involutory cipher algorithm. DDG before (a) and after pipelining (b).

A brute force approach to performance I

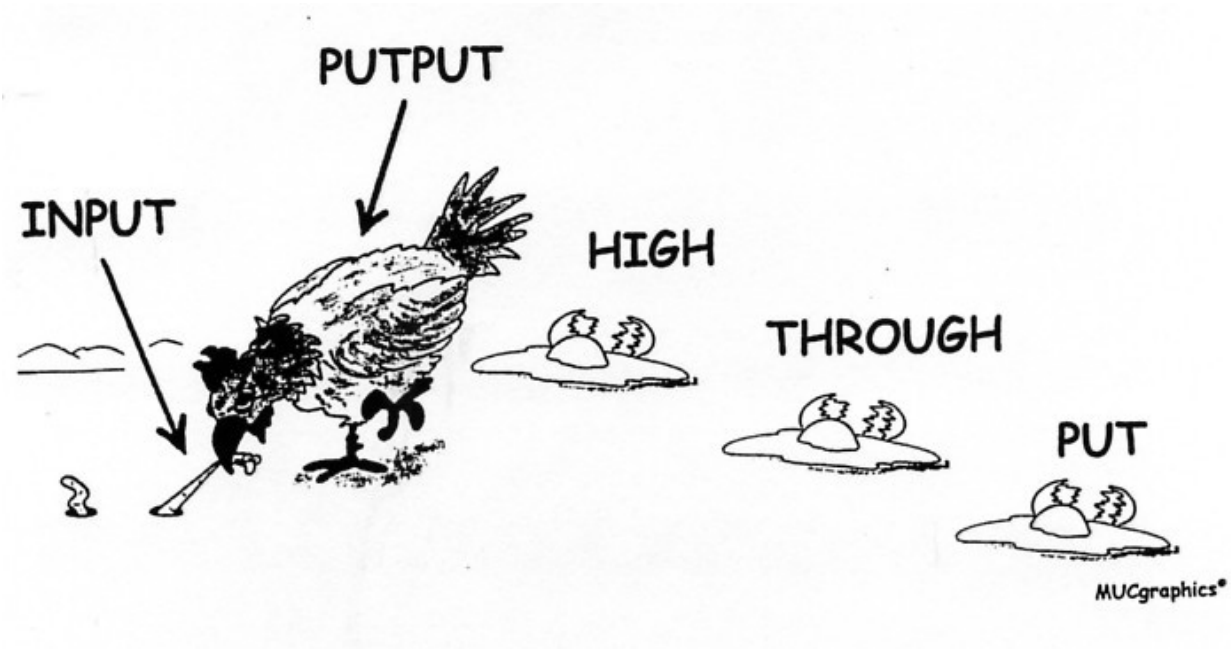


Figure: If one functional unit does not meet your performance goals ...

A brute force approach to performance II

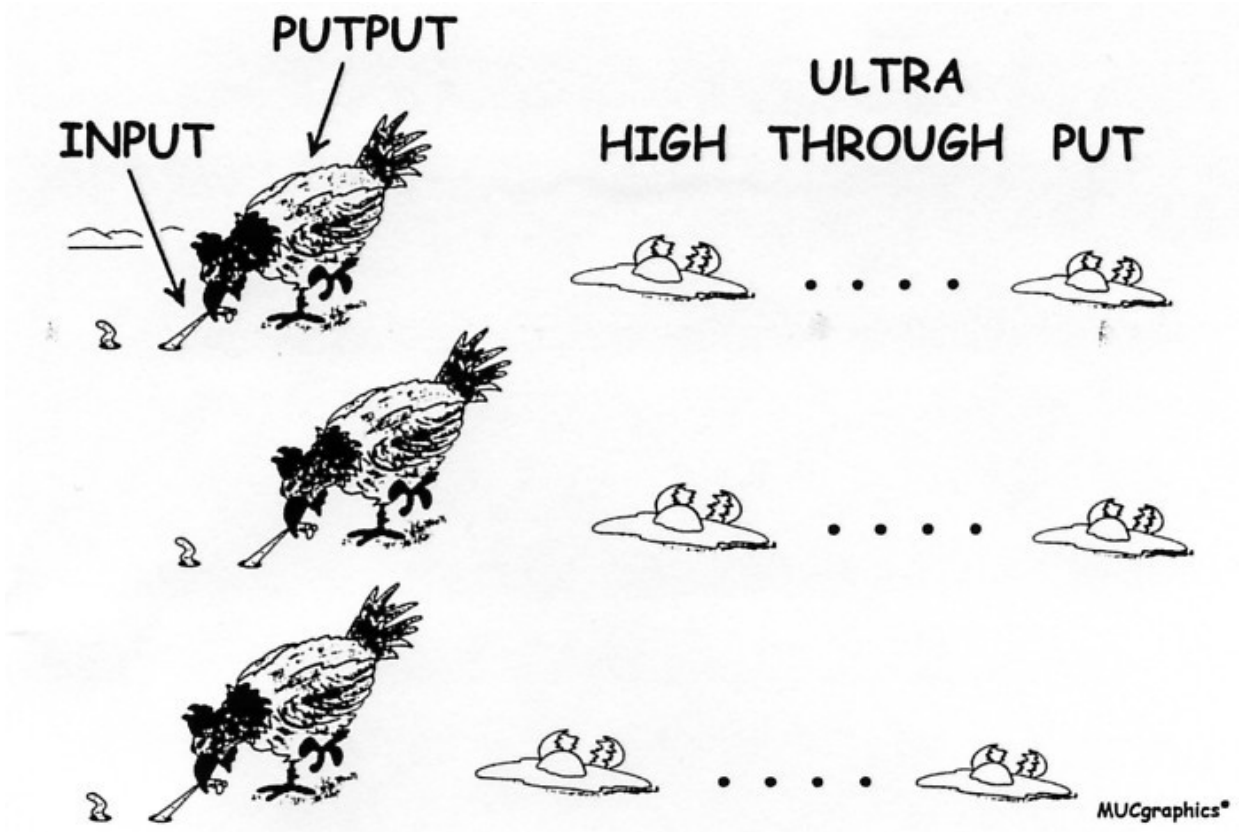


Figure: ... then try to get more of them.

Replication

Paradigm: Multi-piston pump

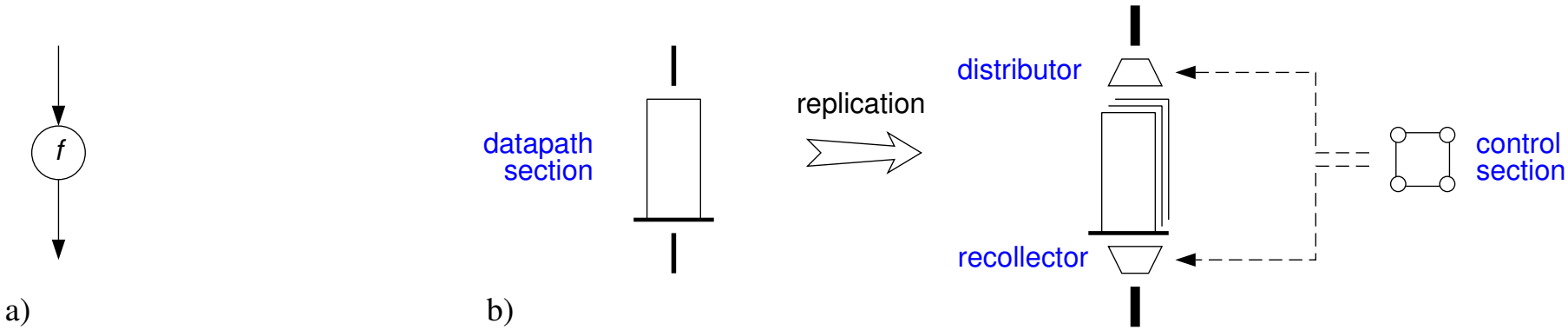


Figure: DDG (a) and hardware configuration for $q = 3$ (b).

Performance and cost analysis

The key characteristics of replication by a factor of q are

$$A(q) = q(A_f + A_{reg}) + A_{ctl}$$

$$\Gamma(q) = \frac{1}{q}$$

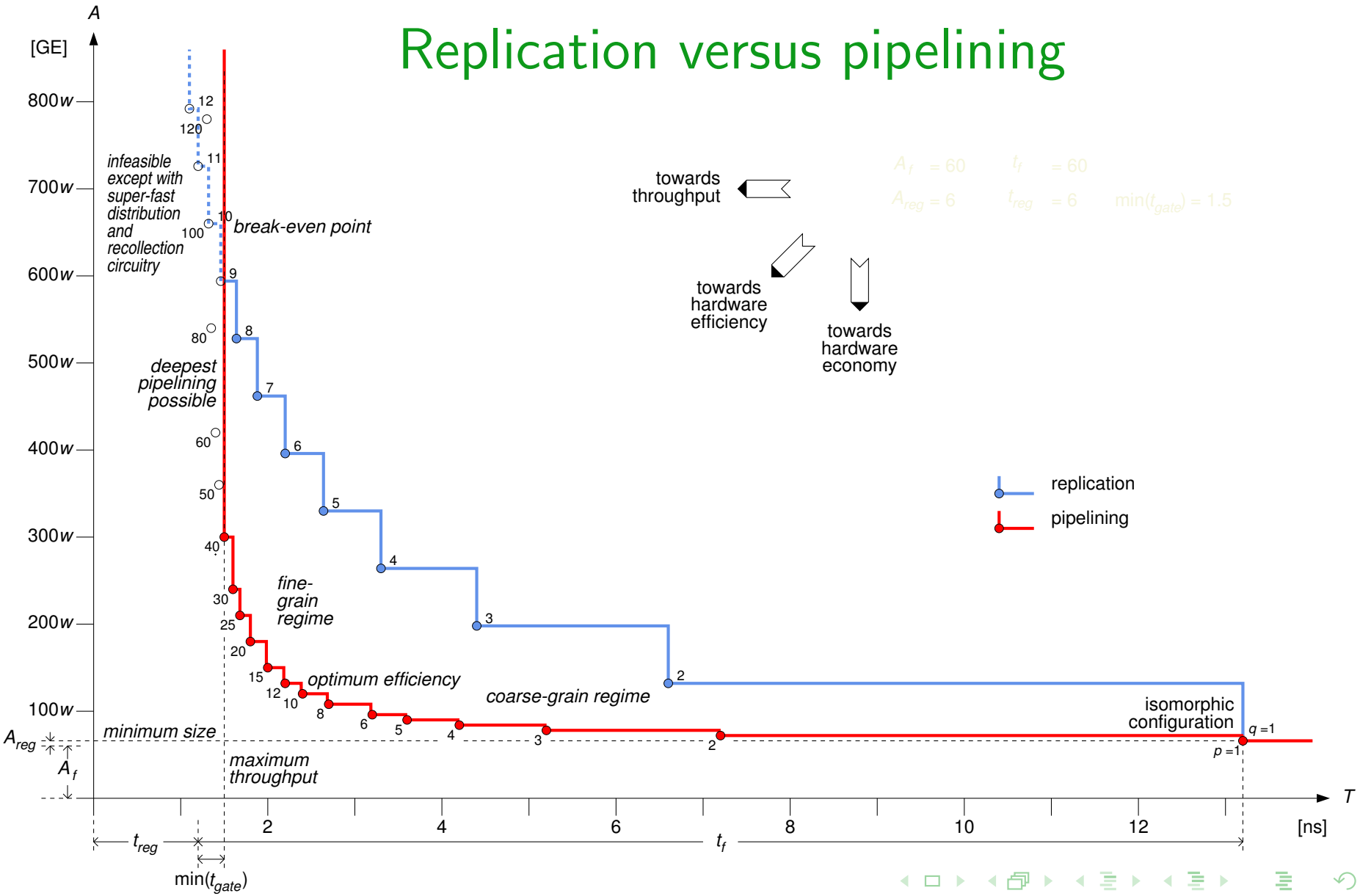
$$t_{lp}(q) \approx t_f + t_{reg}$$

$$AT(q) \approx (A_f + A_{reg} + \frac{1}{q}A_{ctl})(t_f + t_{reg}) \approx (A_f + A_{reg})(t_f + t_{reg})$$

$$L(q) = 1$$

$$E(q) \approx E_f + E_{reg} + E_{ctl}$$

Replication versus pipelining



Example: Microprocessor architectures I

- Superscalar $\mapsto$ multiple ALUs, FPUs, etc. under common control.
- Multicore $\mapsto$ multiple processor cores working independently.

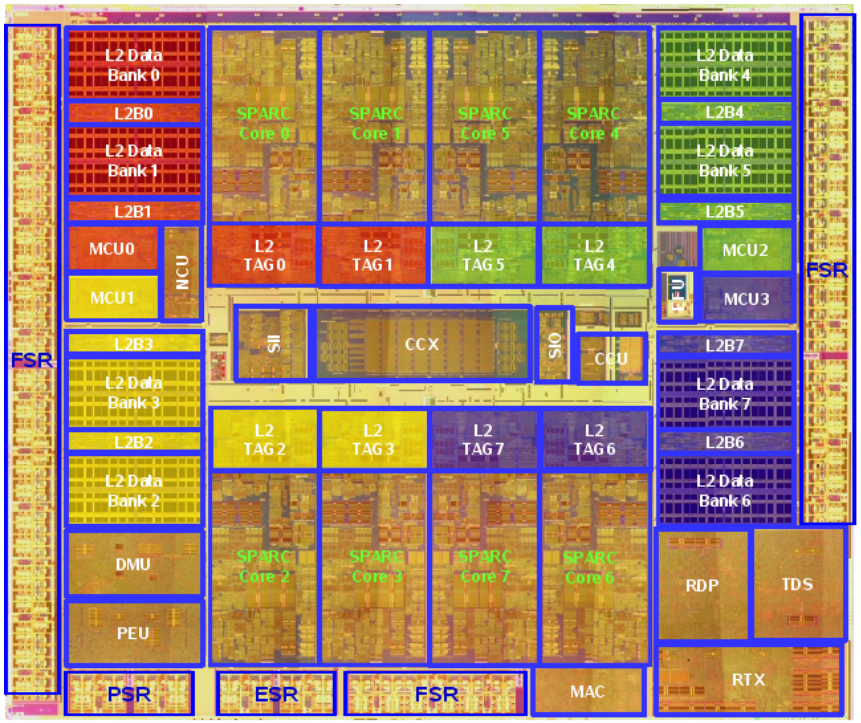


Figure: Floorplan of a Sun Microsystems UltraSPARC T2 CPU (Niagara 2) that combines 8 cores on a single die (separate integer and floating point units in each core, 8 threads/core, 1831 pins, 65 nm CMOS, 342 mm^2, 1.4 GHz).

Example: Microprocessor architectures II

Computer industry has been pushed towards replication because

- CMOS offered more room for increasing circuit complexity than for pushing clock frequencies higher.
- The faster the clock, the smaller the region on a semiconductor die that can be reached within a single clock period.
- Fine grain pipelines dissipate a lot of energy for relatively little computation.
- Reusing a well-tried subsystem benefits design productivity and lowers risks.
- A multicore processor can still be of commercial value even if one of its CPUs is found to be defective.

Time sharing

- Many applications ask for the simultaneous processing of multiple parallel data streams.

Paradigm: Student sharing his time between various subjects

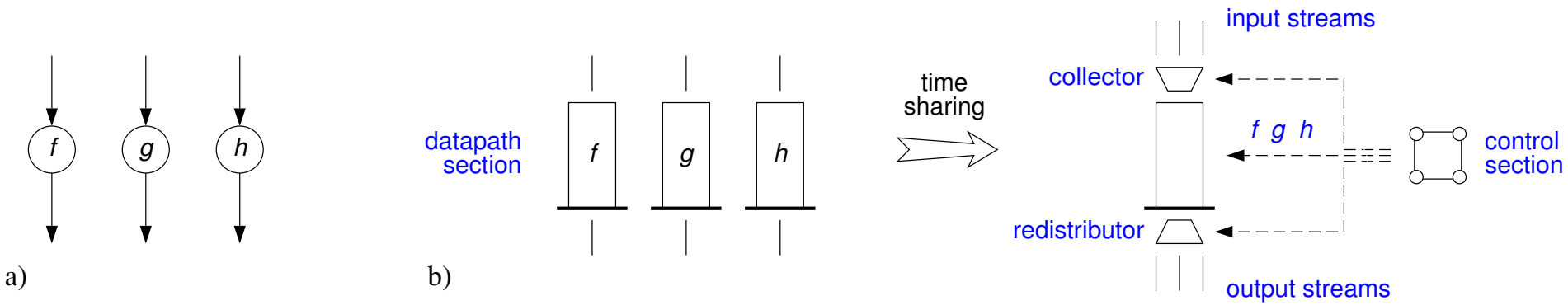


Figure: DDG (a) and hardware configuration for $s = 3$ (b).

Performance and cost analysis

Time sharing by a factor of s yields the following picture

$$\max_{f,g,h}(A) + A_{reg} + A_{ctl} \leq A(s) \leq \sum_{f,g,h} A + A_{reg} + A_{ctl}$$

$$\Gamma(s) = s$$

$$t_{lp}(s) \approx \max_{f,g,h}(t) + t_{reg}$$

$$s(\max_{f,g,h}(A) + A_{reg} + A_{ctl})(\max_{f,g,h}(t) + t_{reg}) \leq AT(s) \leq$$

$$s(\sum_{f,g,h} A + A_{reg} + A_{ctl})(\max_{f,g,h}(t) + t_{reg})$$

$$L(s) = s$$

$$E(s) \approx s \max_{f,g,h}(E) + E_{reg} + E_{ctl}$$

Insight gained

Time sharing

- is most favorable when one monofunctional datapath proves sufficient because all streams are to be processed in exactly the same way
- is unattractive when subfunctions are very disparate because no substantial savings can be obtained from concentrating their processing into one multifunctional datapath
- refrains from taking advantage of the parallelism inherent in the original problem
- may be viewed as complementary to replication

Example: 8-point FFT

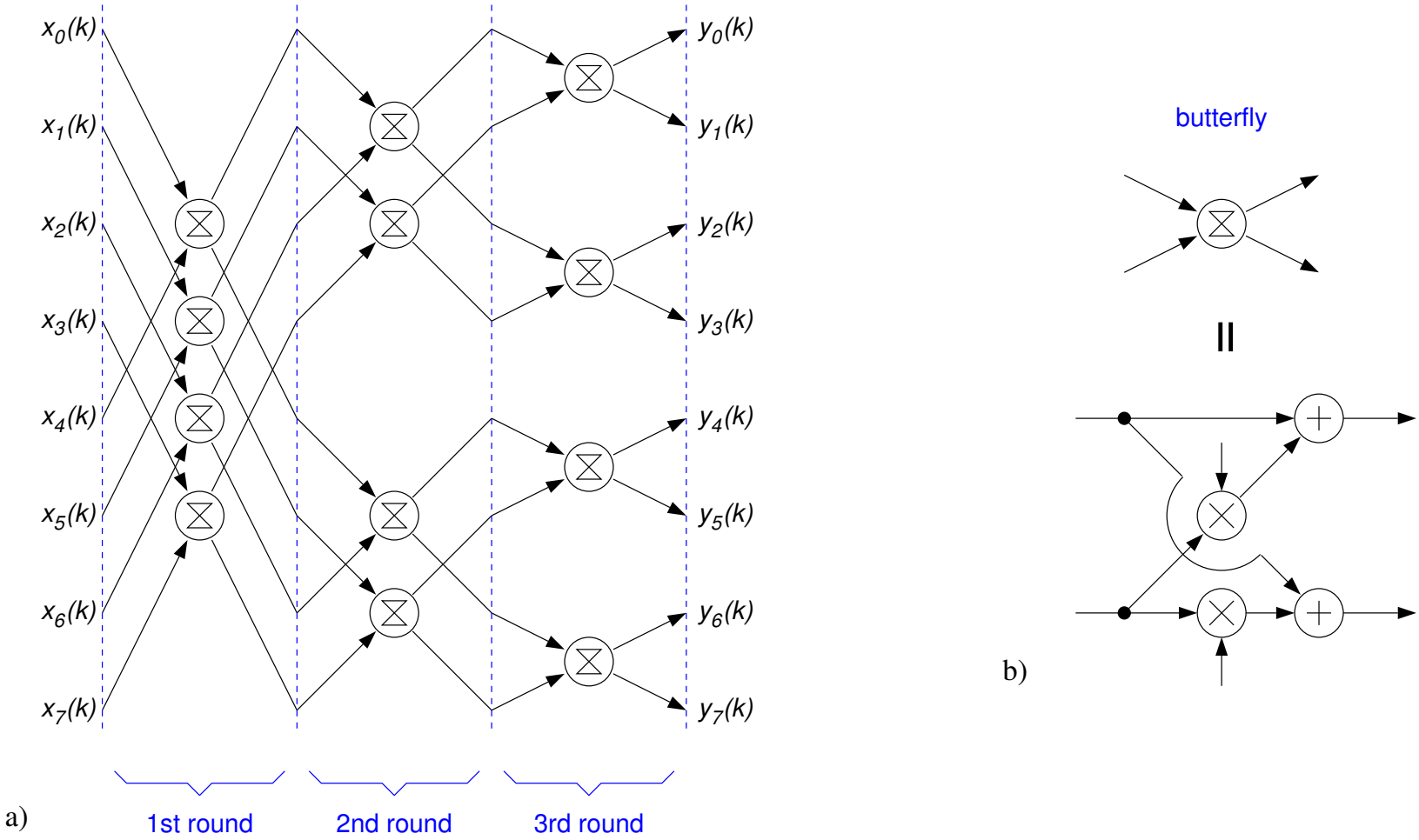
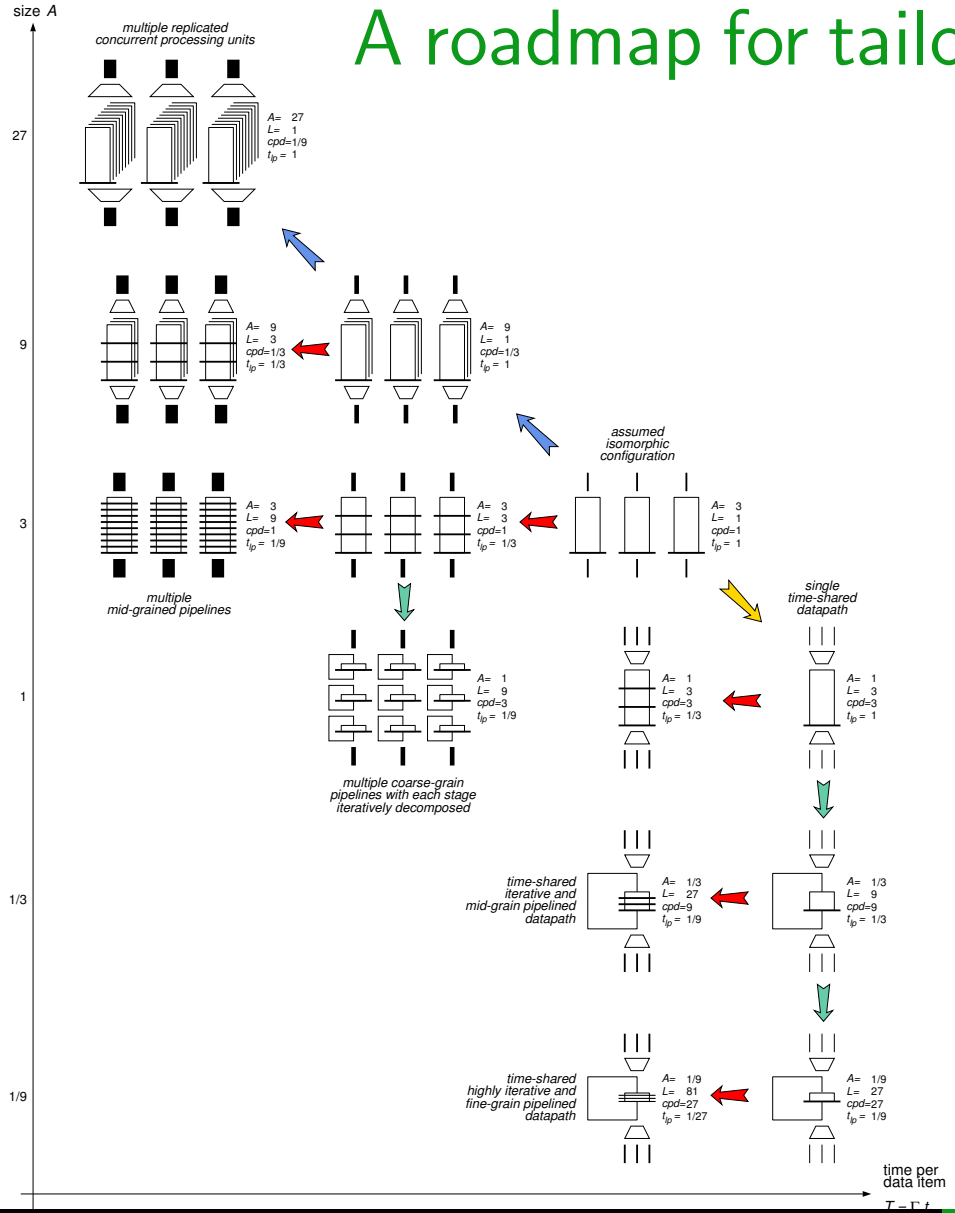
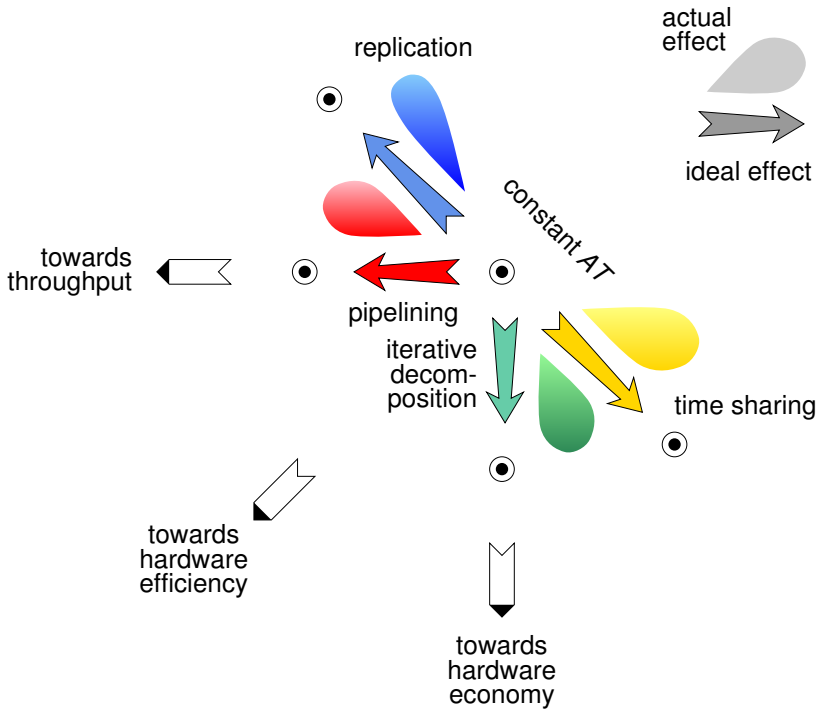


Figure: DDG of 8-point FFT (a) and DDG of butterfly operator (b).

A roadmap for tailoring combinational hardware





Example: Two cryptochip architectures compared

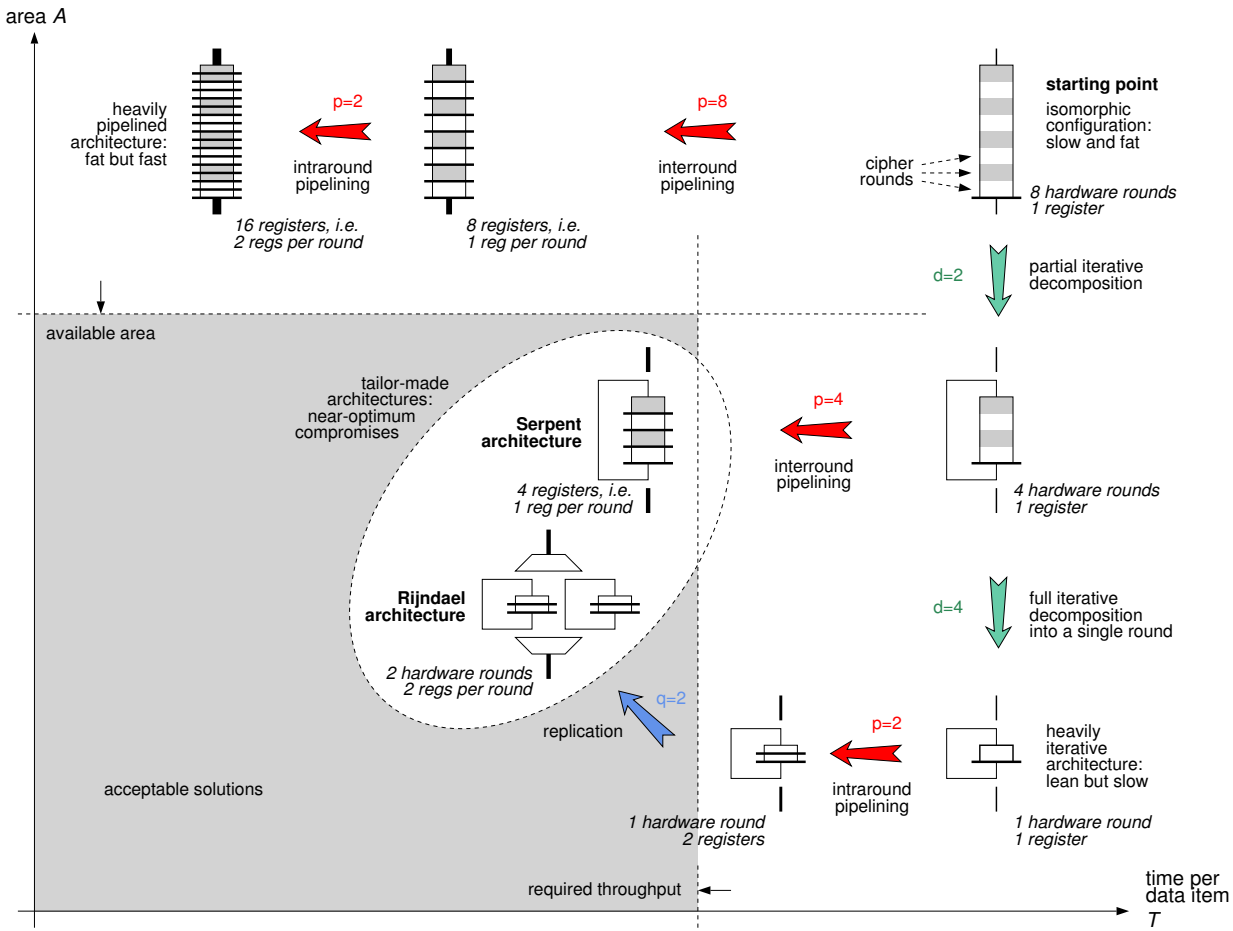


Figure: Two competing teams have taken different routes but have arrived at similar compromises between throughput and area (ETH CHES 2002).

Universal versus algebraic transforms

Universal transforms. Whether and how to apply them can be decided from a DDG's connectivity and weights alone, no matter what operations the vertices stand for.

Examples: Iterative decomposition, pipelining, replication, time sharing, more to follow.

Universal versus algebraic transforms

Universal transforms. Whether and how to apply them can be decided from a DDG's connectivity and weights alone, no matter what operations the vertices stand for.

Examples: Iterative decomposition, pipelining, replication, time sharing, more to follow.

Algebraic transforms. Take advantage of specific algebraic properties of the operations involved.

Examples: Associativity transform, commutativity transform, Horner's scheme, method of finite differences (Charles Babbage, 1822), etc.

Example: Associativity transform

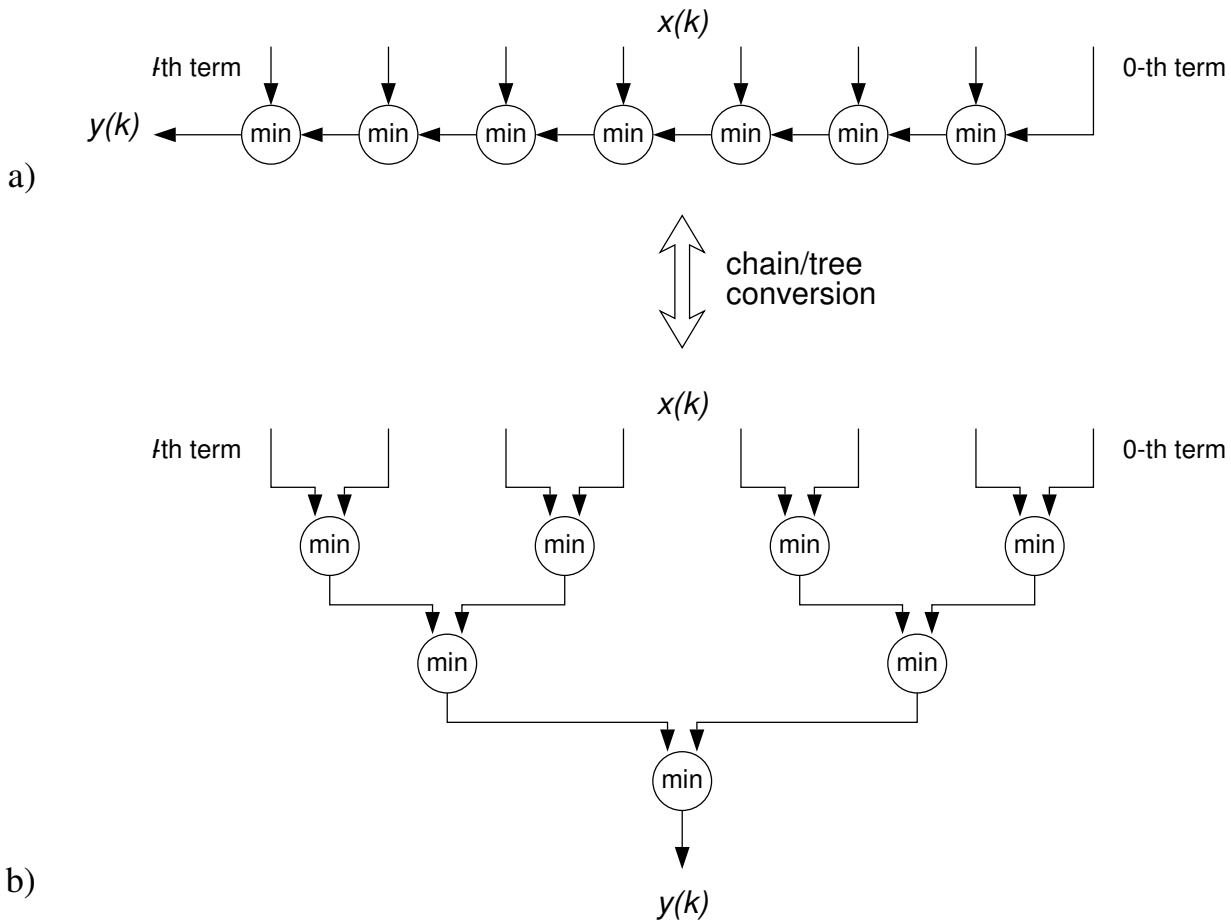


Figure: 8-way minimum function. Chain-type DDG (a), tree-type DDG (b).

Recapitulation

Equivalence transforms that help optimize combinational computations

Iterative decomposition, pipelining, replication and algebraic transforms, plus time sharing in the presence of parallel data streams.

- Iterative decomposition and time sharing are most effective when a computational unit can be reused several times.
- Pipelining is generally superior to replication.
  While coarse grain pipelining improves throughput dramatically, benefits decline as more and more stages are included.
- Pipelining and iterative decomposition are complementary in that they both can contribute to lowering the size-time product.
- Lowering the size-time product AT always implies cutting down the longest path t_{lp}.

Subject

Options for temporary storage of data

Why and when do we need to stora data?

Except for trivial SSI/MSI circuits, any IC includes some form of memory.

This is either because

- the data processing algorithm is of sequential nature and, therefore, asks for functional memory,

or because

- nonfunctional storage got introduced into the circuit as a consequence from architectural transformations.

Options for temporary storage of data

Architectural options for temporary storage of data:

On-chip registers built from individual flip-flops or latches.

On-chip memory i.e. SRAM macrocell (or possibly embedded DRAM).

Off-chip memory i.e. SRAM or DRAM catalog part.

Options for temporary storage of data

Architectural options for temporary storage of data:

On-chip registers built from individual flip-flops or latches.

On-chip memory i.e. SRAM macrocell (or possibly embedded DRAM).

Off-chip memory i.e. SRAM or DRAM catalog part.

Differences that impact high-level design decisions:

- One-at-a-time versus all-at-a-time data access patterns
- Available memory configurations and area occupation
- Storage capacities
- Wiring and the costs of going off-chip
- Energy efficiency
- Latency and timing

Data access patterns

RAMs impose access one data word after the other

Fine in architectures obtained from

- iterative decomposition and
- time sharing.

Perfect match for microprocessors
("fetch, load, execute, store").

Registers allow for simultaneous access to all data words stored

Mandatory in high-throughput architectures obtained from

- pipelining,
- retiming, *to be introduced later in this chapter*
- loop unfolding *idem*

where data are kept moving in every computation cycle.

Available memory configurations

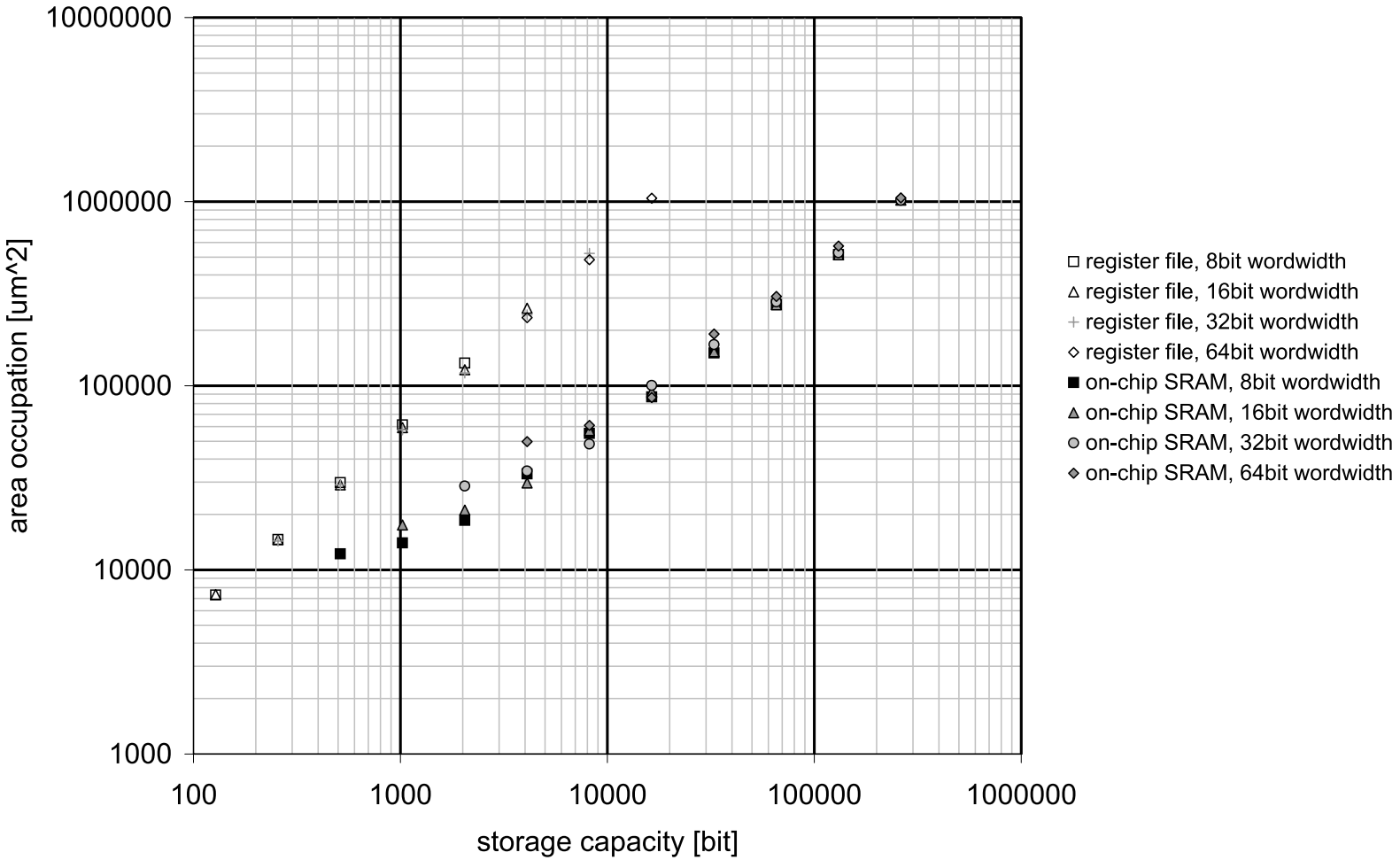


Figure: Area occupation of registers and on-chip RAMs for a 130 nm CMOS.

Wiring and the costs of going off-chip

Off-chip memories add to pin count, package count, and board space.

- Extra parasitic capacitances
- Extra delays
- Extra energy dissipation

Wiring and the costs of going off-chip

Off-chip memories add to pin count, package count, and board space.

- Extra parasitic capacitances
- Extra delays
- Extra energy dissipation
- Commodity RAMs impose bidirectional pads which require special attention.
  - Stationary and transient drive conflicts must be avoided.
  - ATE must be made to alternate between read and write modes with no physical access to any control signal within the chip.
  - Test patterns must address bidirectional operation and high-impedance states.
  - Electrical and timing measurements become more complicated.

Conclusion

Off-chip data storage is associated with important penalties.

Options for temporary data storage compared

architectural option	on-chip				off-chip
	bistables		embedded		commodity
	flip-flop	latch	SRAM	DRAM	DRAM
fabrication process	compatible with logic				optimized
devices in each cell	20...30T	12...16T	6T	1T1C	1T1C
cell area per bit [F^2]	1700...2800	1100...1800	135...170	18...30*	6...8
extra circuit overhead	none		1.3 $\leq$ factor $\leq$ 2		off-chip
memory refresh cycles	none			yes	
extra package pins	none		none		addr. & data bus
nature of wiring	multitude of local lines		on-chip busses		package & board
bidirectional busses	none		optional		mandatory
access to data words	all at a time		one at a time		
available configurations	any		restricted		
energy efficiency	good		fair	poor	very poor
latency and paging	none		no fixed rules		yes
impact on clock period	minor		substantial		severe

* As low as 6...8 for processes that accomodate 3D capacitors (4 to 6 extra masks)

Example: RAMs in a CMOS ASIC technology

Cu-11 is an ASIC technology by IBM (2002)

- gate length 110 nm, supply voltage 1.2 V
- Cu interconnect combined with low-k interlevel dielectrics

SRAM macrocell generator from 128 bit to 1 Mibit

Embedded DRAM megacells up to 16 Mibit (with trench caps)

- cycle time of 1 Mibit eDRAM is 15 ns
  (equivalent to $555 \cdot t_{pd}$ of a 2-input NAND)
- eDRAM bit cell area is 0.31 μm^2
- 1 Mibit eDRAM occupies an area of 2.09 mm^2 (84% overhead)
- 16 Mibit eDRAM occupies 14.1 mm^2 (63% overhead)

Recapitulation

Observation

There is no such thing as an optimal solution for temporary storage of data, what is best strongly depends on the situation and requirements.

- Only registers allow for simultaneous access to all data, but occupy a lot of die area per bit.
- SRAMs can hold more significant quantities of data than registers but are slower than registers, yet faster than DRAMs.
- DRAMs require periodical refresh $\rightsquigarrow$ power dissipated even when idle.
- DRAM and Flash memories are cost-efficient for large data quantities.
- Flash is used for permanent storage, but is much slower than RAM.
- Commodity memories offer virtually unlimited capacities at low costs, but are is associated with speed, energy and other penalties.

Subject

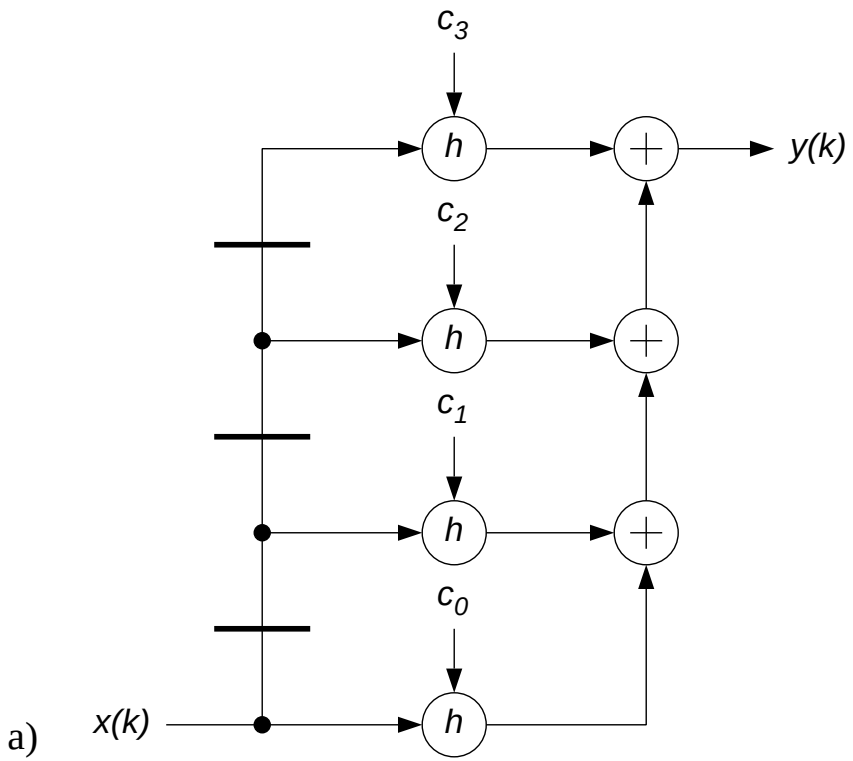
Transforms for non-recursive computations

What do we mean by non-recursive computation?

A computation is termed (sequential and) non-recursive if

- Result is dependent on past arguments, not just present.
- Edges with weights greater than zero are present in the DDG.
- DDG is free of circular paths.

Example: Nonlinear time-invariant third order correlator



Can you do better
in terms of speed and area?

- Pipelining helps boost throughput but is rather inefficient in this case.

Retiming

Paradigm: Repartition workloads evenly for all workers on an assembly line

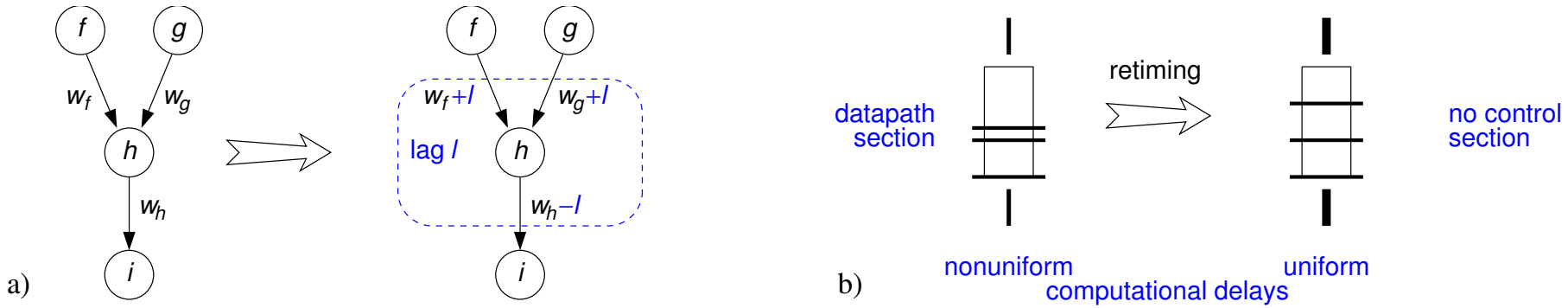


Figure: DDG (a) and hardware configuration for $l = 1$ (b).

Formal rules

To be legal, any retiming must observe the following rules:

1. Neither outputs nor sources of time-varying inputs may be part of a supervertex that is to be retimed.
2. When a supervertex is assigned a lag (lead) by l computation cycles, the weights of all its incoming edges are in- (de-)cremented by l and the weights of all its outgoing edges are de- (in-)cremented by l.
3. No edge weight may be changed to assume a negative value.
4. Any circular path must always include at least one edge of strictly positive weight (roundtrip weights will never change).

Pipelining revisited

Same rules as for retiming except

1. Any supervertex to be assigned a lag (lead) must include all outputs (all time-varying inputs).

Pipelining revisited

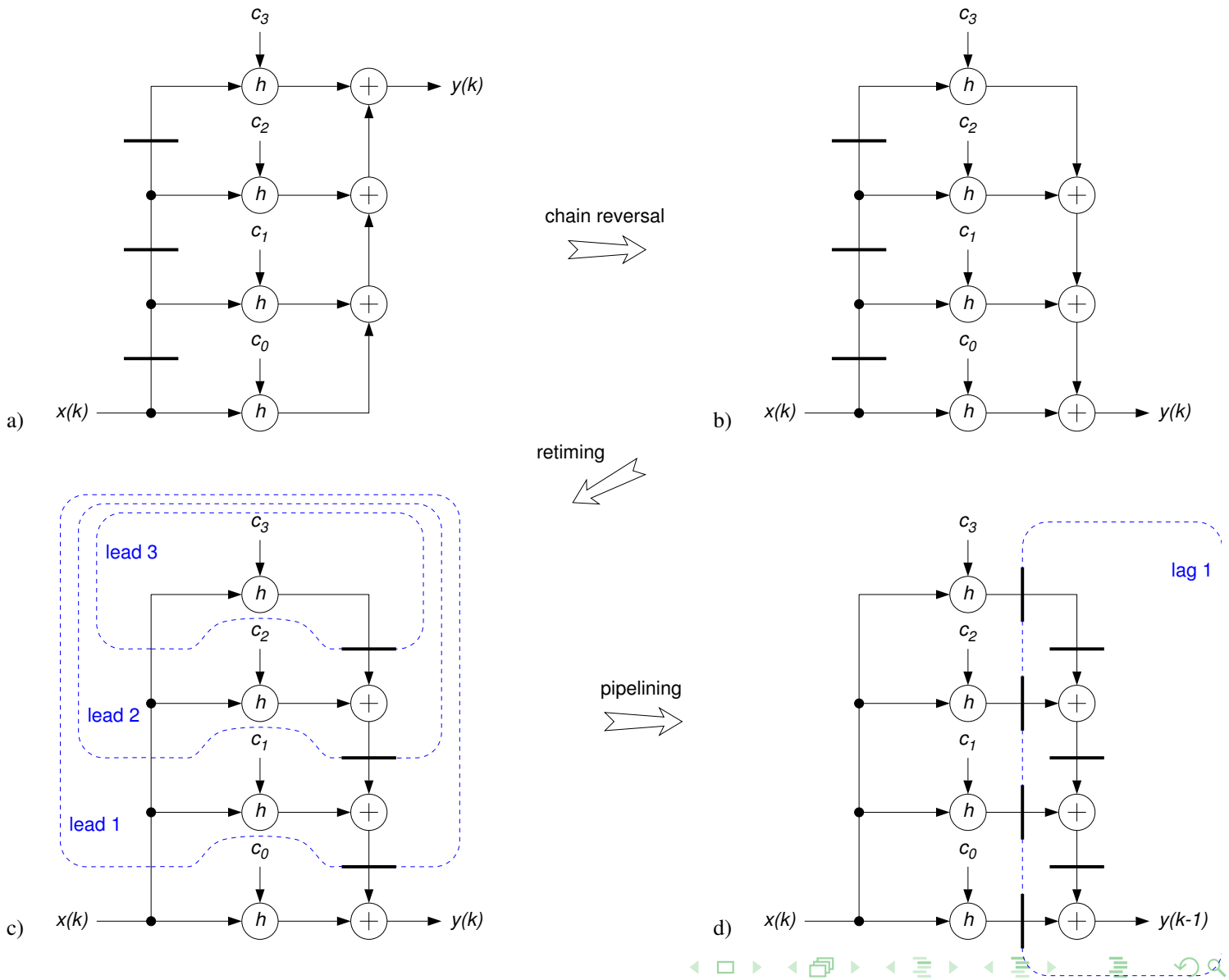
Same rules as for retiming except

1. Any supervertex to be assigned a lag (lead) must include all outputs (all time-varying inputs).

Comparison

- Both transforms aim at shortening the longest path.
- Pipelining increases latency as registers get added.
- Retiming leaves latency unchanged as registers get relocated.

Example: Nonlinear time-invariant third order correlator



Example: Nonlinear time-invariant third order correlator

The subsequent transforms change the circuit's performance as follows:

Key characteristics	Architectural variant: original (a)	reversed (b)	+ retimed (c)	+ pipelined (d)
arithmetic units	$(N+1)A_h + NA_+$	*idem*	*idem*	*idem*
functional registers	NA_{reg}	*idem*	*idem*	*idem*
nonfunctional registers	0	*idem*	*idem*	$(N+1)A_{reg}$
cycles per data item Γ	1	*idem*	*idem*	*idem*
longest path delay t_{lp}	$t_{reg} + t_h + N\,t_+$	*idem*	$t_{reg} + t_h + t_+$	$t_{reg} + \max(t_h, t_+)$
for $N = 3$ [ns]	9.5	*idem*	5.5	3.5
for $N = 30$ [ns]	63.5	*idem*	5.5	3.5
latency L	0	*idem*	*idem*	1

A DDG is termed systolic if the edge weight between any two vertices is one or more. For a given granularity, maximum speed is obtained when there is no more than one combinational operation between any two registers. This is the basic idea behind systolic computation

Example: Nonlinear time-invariant third order correlator

The subsequent transforms change the circuit's performance as follows:

Key characteristics	Architectural variant: original (a)	reversed (b)	+ retimed (c)	+ pipelined (d)
arithmetic units	$(N+1)A_h + NA_+$	*idem*	*idem*	*idem*
functional registers	NA_{reg}	*idem*	*idem*	*idem*
nonfunctional registers	0	*idem*	*idem*	$(N+1)A_{reg}$
cycles per data item Γ	1	*idem*	*idem*	*idem*
longest path delay t_{lp}	$t_{reg} + t_h + N\,t_+$	*idem*	$t_{reg} + t_h + t_+$	$t_{reg} + \max(t_h, t_+)$
for $N = 3$ [ns]	9.5	*idem*	5.5	3.5
for $N = 30$ [ns]	63.5	*idem*	5.5	3.5
latency L	0	*idem*	*idem*	1

Net benefits:

- Long path delay greatly reduced at little hardware costs.
- Maximum operating speed no longer a function of correlation order N.

Iterative decomposition and time sharing revisited

- Decomposing and time sharing sequential computations is straightforward and can significantly reduce datapath hardware.
- Functional memory requirements remain the same as in the isomorphic architecture (memory bound).
- Mixed blessing energy-wise.
  - \+ More uniform combinational depth reduces glitching activity.
  - – Extra multiplexers necessary to route, recycle, collect and/or redistribute data.
  - – Extra counter or finite state machine required to control the datapath.

Example: Third order transversal filter

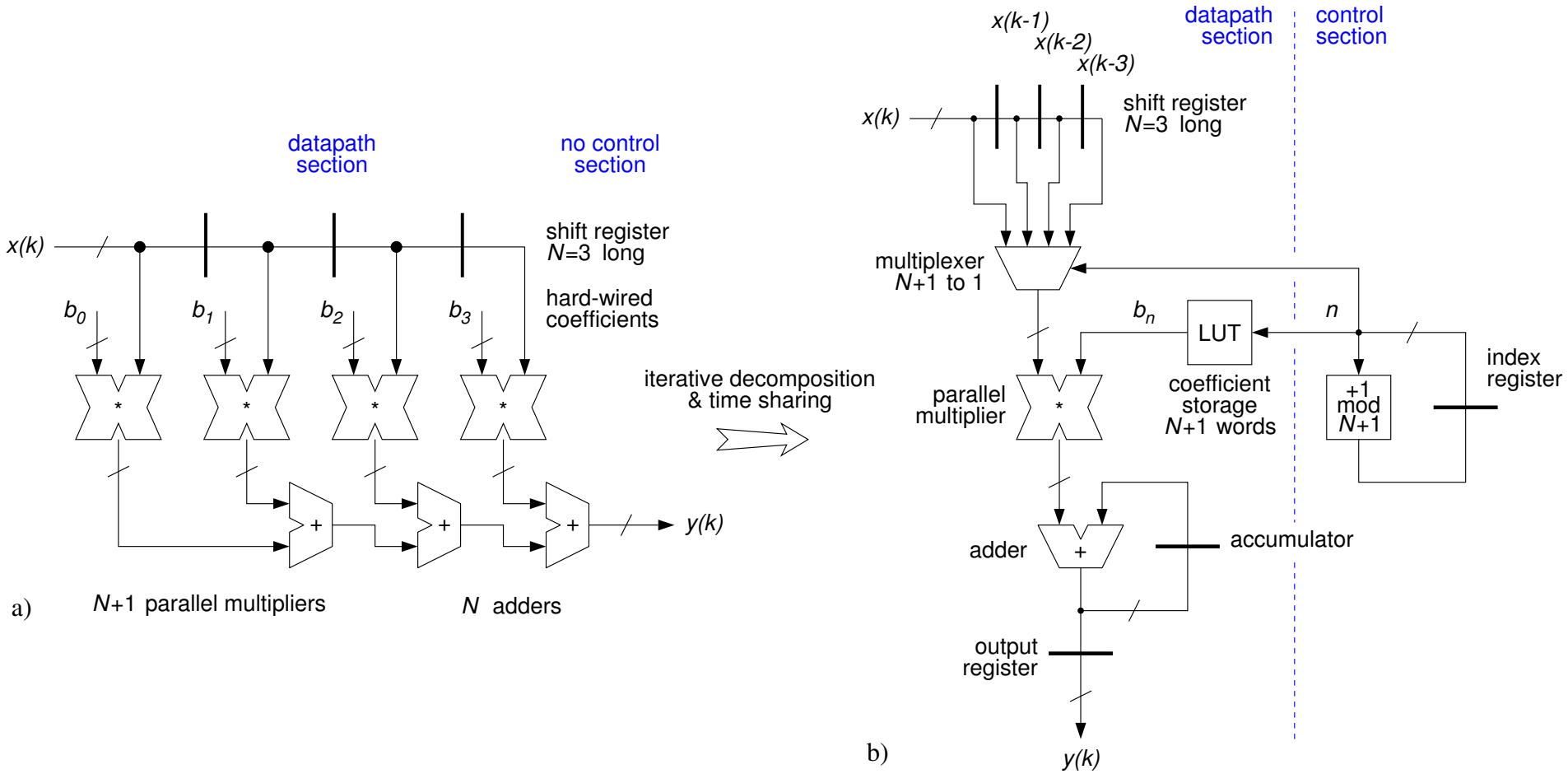


Figure: Isomorphic architecture (a) and a more economic alternative (b).

Recapitulation

Retiming

can help to optimize datapath architecture for sequential computations without affecting functionality nor latency.

- Retiming, pipelining and combinations of the two can improve throughput of arbitrary feedforward computations.
- The associative law allows one to take full advantage of the above transforms by having a DDG rearranged beforehand.
- Iterative decomposition and time sharing are the two options available for reducing circuit size.
- Highly time-multiplexed architectures dissipate energy on ancillary activities that do not directly contribute to data computation.

Subject

Transforms for recursive computations

What do we mean by recursive computation?

A computation is termed (sequential and) recursive if

- Result is dependent on earlier outcomes of the computation itself.
- Edges with weights greater than zero are present in the DDG.
- Circular paths (of non-zero weight) exist in the DDG.

Linear time-invariant first-order feedback loop I

Recursions such as

$$y(k) = ay(k-1) + x(k)$$

which in the z domain corresponds to transfer function

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - az^{-1}}$$

have many technical applications.

Examples:

- IIR filters
- Differential pulse code modulation encoders (DPCM)
- Servo loops

They impose a stiff timing constraint, however.

Linear time-invariant first-order feedback loop II

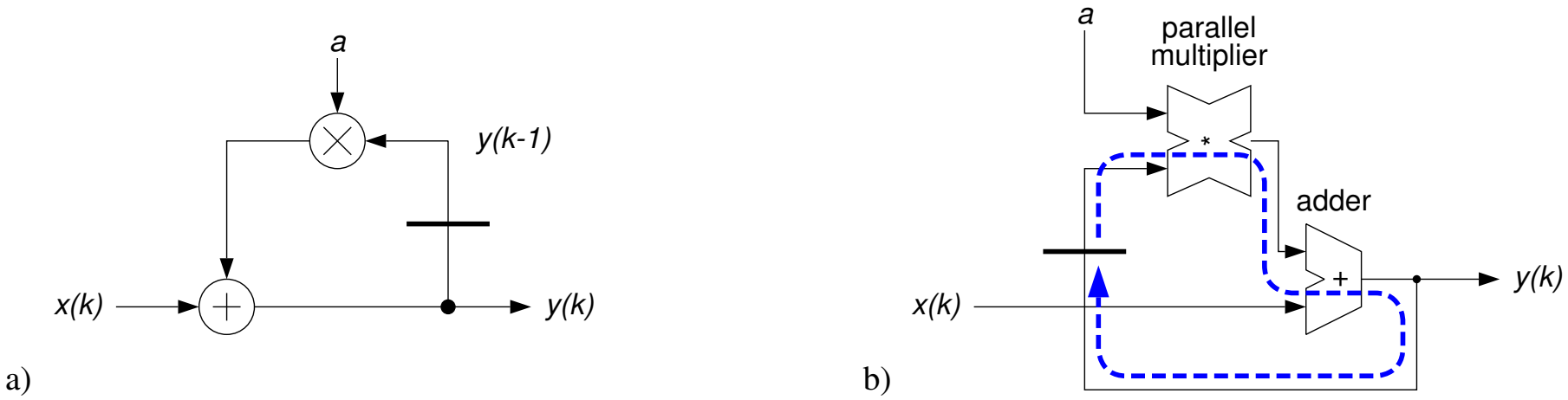


Figure: DDG (a) and isomorphic architecture (b).

Linear time-invariant first-order feedback loop II

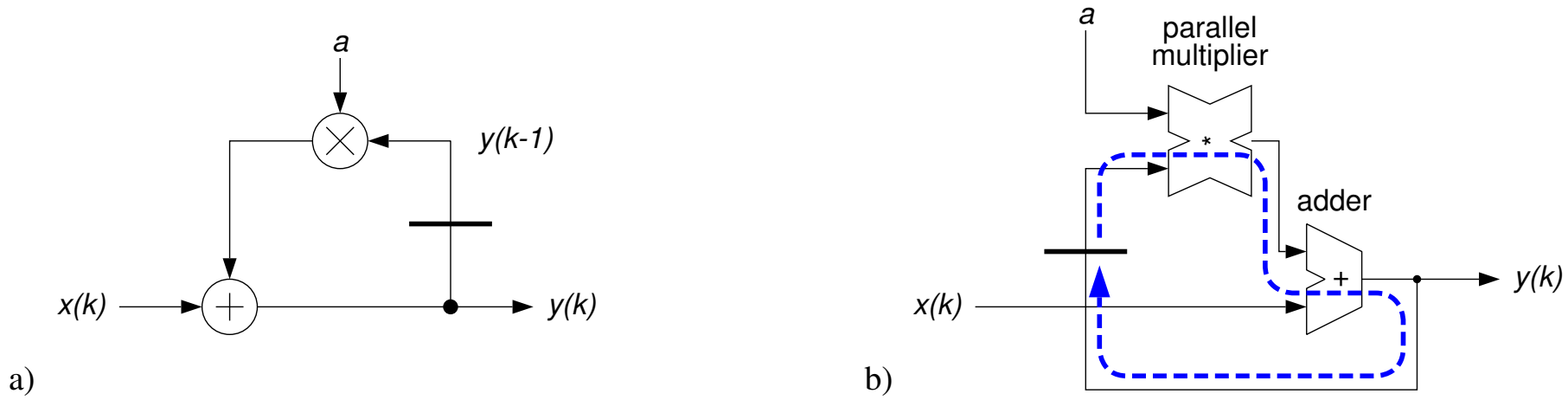


Figure: DDG (a) and isomorphic architecture (b).

Iteration bound: $\sum_{loop} t = t_{reg} + t_* + t_+ = t_{lp} \leq T_{cp}$

- No problem as long as long path constraint can be met with available and affordable technology.
- No obvious solution otherwise, recursiveness is a real bottleneck.

Linear time-invariant first-order feedback loop III

Have a second look!

Key idea

Relax the timing constraint by inserting additional latency registers into the feedback loop.

Linear time-invariant first-order feedback loop III

Have a second look!

Key idea

Relax the timing constraint by inserting additional latency registers into the feedback loop.

A tentative solution must look like

$$H(z) = \frac{Y(z)}{X(z)} = \frac{N(z)}{1 - a^p z^{-p}}$$

where $N(z)$ is here to compensate for the changes due to the new denominator.

Recalling the sum of geometric series we easily establish $N(z)$ as

$$N(z) = \frac{1 - a^p z^{-p}}{1 - a z^{-1}} = \sum_{n=0}^{p-1} a^n z^{-n}$$

Linear time-invariant first-order feedback loop IV

The new transfer function can then be completed to become

$$H(z) = \frac{\sum_{n=0}^{p-1} a^n z^{-n}}{1 - a^p z^{-p}}$$

and the new recursion in the time domain follows as

$$y(k) = a^p y(k-p) + \sum_{n=0}^{p-1} a^n x(k-n)$$

Linear time-invariant first-order feedback loop V

After unfolding by a factor of $p = 4$, the original recursion takes on the form

$$y(k) = a^4 y(k-4) + a^3 x(k-3) + a^2 x(k-2) + a x(k-1) + x(k)$$

which corresponds to transfer function

$$H(z) = \frac{1 + az^{-1} + a^2 z^{-2} + a^3 z^{-3}}{1 - a^4 z^{-4}} \qquad \text{in lieu of} \qquad \frac{1}{1 - az^{-1}}$$

Net result:

- Denominator has been widened to include p unit delays rather than one.
- Numerator stands for a feedforward circuit that is amenable to pipelining.

Linear time-invariant first-order feedback loop VI

Particularly elegant and efficient solutions exist when p is an integer power of 2 because of the lemma

$$\sum_{n=0}^{p-1} a^n z^{-n} = \prod_{m=0}^{\log_2 p - 1} (a^{2^m} z^{-2^m} + 1) \qquad p = 2, 4, 8, 16, \ldots$$

With $p = 4$, for instance, the numerator can be factorized into

$$H(z) = \frac{(1 + az^{-1})(1 + a^2 z^{-2})}{1 - a^4 z^{-4}} \quad \text{in lieu of} \quad \frac{1}{1 - az^{-1}}$$

Linear time-invariant first-order feedback loop VII

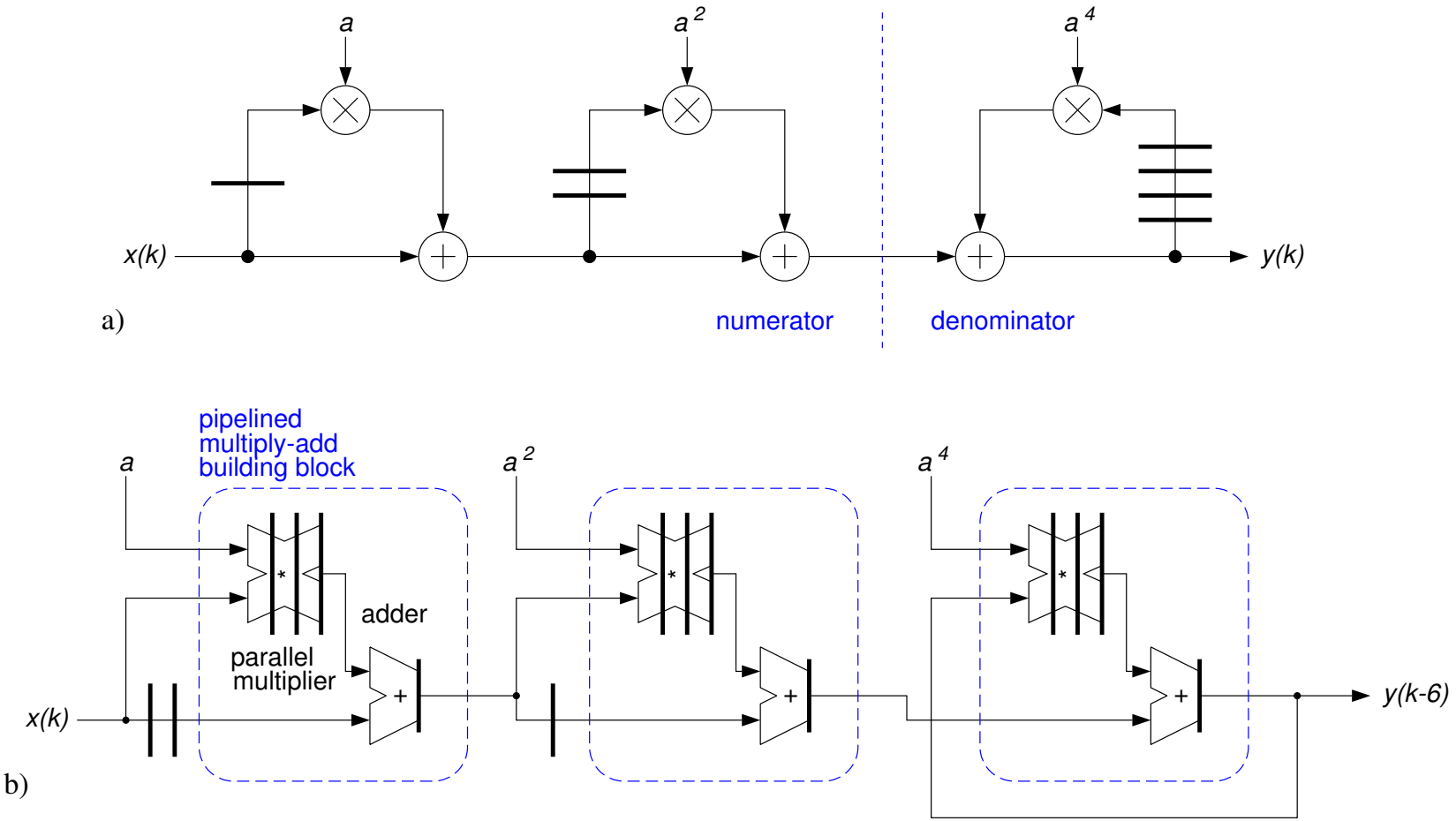


Figure: DDG unfolded by $p = 4$ (a) and high-performance architecture (b).

Higher-order loops

Guideline

Do not attempt to unfold loops of arbitrary order directly.
Make use of a common technique from digital filter design.

- Any higher-order transfer function can be factored into a product of second- and first-order terms.
- The resulting DDG takes the form of cascaded second- and first-order sections.
- As an added benefit, cascade structures are known to be less sensitive to quantization of coefficients and signals than direct forms.

Linear time-invariant second-order feedback loop I

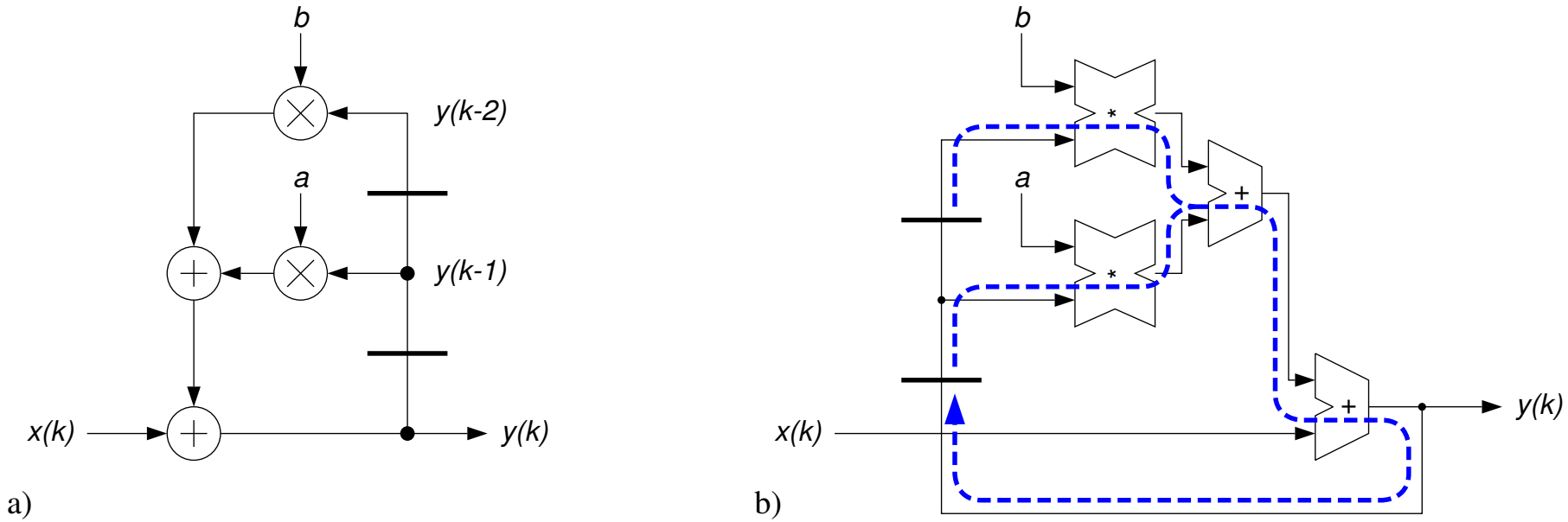


Figure: DDG (a) and isomorphic architecture (b).

Linear time-invariant second-order feedback loop II

A second-order recursive function goes

$$y(k) = ay(k-1) + by(k-2) + x(k)$$

or, in the z domain,

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 - az^{-1} - bz^{-2}}$$

Unfolding is obtained from multiplying numerator and denominator by an adequate factor. For $p = 4$, the transfer function becomes

$$H(z) = \frac{(1 + az^{-1} - bz^{-2})\,(1 + (a^2 + 2b)z^{-2} + b^2z^{-4})}{1 - ((a^2 + 2b)^2 - 2b^2)z^{-4} + b^4z^{-8}}$$

Linear time-invariant second-order feedback loop III

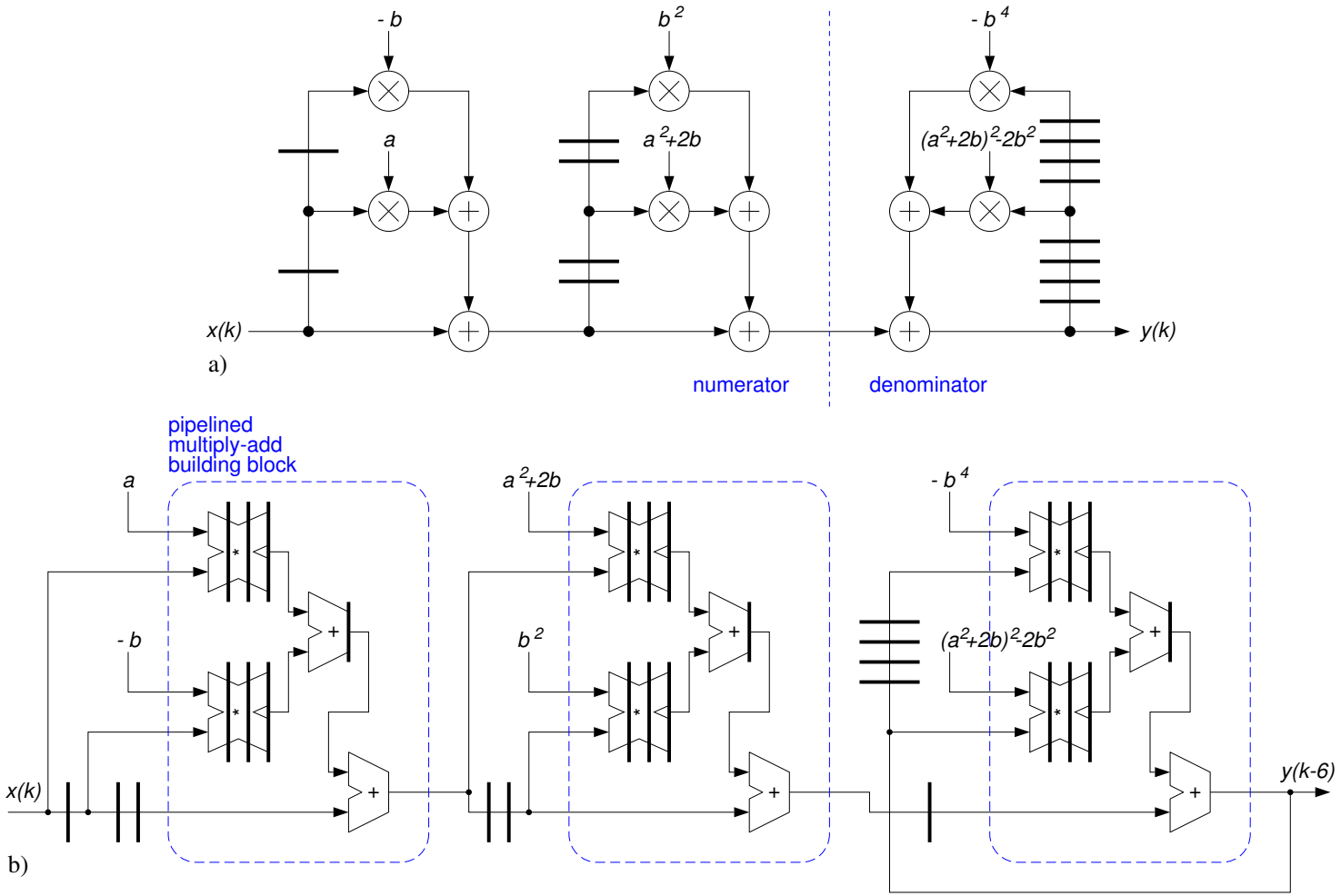


Figure: DDG unfolded by $p = 4$ (a) and high-performance architecture (b).

Example: Fourth-order ARMA filter [1]

- Two second-order sections cascaded, loops unfolded with p=4.
- Pipelined multiply-add units with carry-save and carry-ripple adders.
- Fabricated in standard 0.9 μm CMOS technology (1992).
- Sampling frequency $f_s = f_{clk} = 85$ MHz, $\Gamma = 1$.
- Computation rate $\approx$ 1.5 GOPS.
- One to two extra data bits added to maintain similar roundoff noise.
- Circuit size approximately 20 kGE.
- Supply 5 V, power dissipation 2.2 W at full speed.

[1] ARMA stands for "auto recursive moving average", i.e. for IIR filters that comprise both recursive (AR) and non-recursive computations (MA).

Example: Fourth-order ARMA filter [1]

- Two second-order sections cascaded, loops unfolded with p=4.
- Pipelined multiply-add units with carry-save and carry-ripple adders.
- Fabricated in standard 0.9 μm CMOS technology (1992).
- Sampling frequency $f_s = f_{clk} = 85$ MHz, $\Gamma = 1$.
- Computation rate $\approx$ 1.5 GOPS.
- One to two extra data bits added to maintain similar roundoff noise.
- Circuit size approximately 20 kGE.
- Supply 5 V, power dissipation 2.2 W at full speed.

$\rightsquigarrow$ Loop unfolding allows to push out the need for fast but costly fabrication technologies such as GaAs, then and now.

[1] ARMA stands for "auto recursive moving average", i.e. for IIR filters that comprise both recursive (AR) and non-recursive computations (MA).

Linear time-variant first-order feedback loop

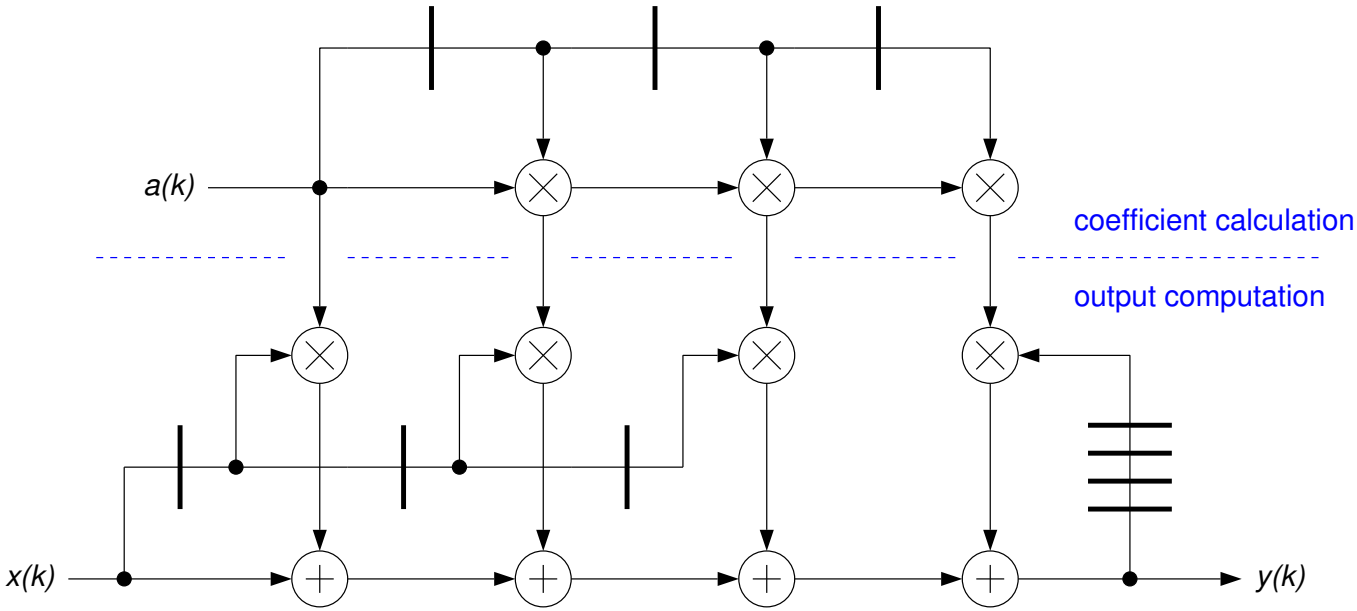


Figure: DDG after unfolding by a factor of $p = 4$.

- Coefficient terms must be calculated on-line requiring extra hardware.

Nonlinear or general loops I

The most general case of a first-order recursion goes

$$y(k) = f(y(k-1), x(k))$$

and can be unfolded an arbitrary number of times,
e.g. with $p = 2$ to become

$$y(k) = f(f(y(k-2), x(k-1)), x(k))$$

Nonlinear or general loops II

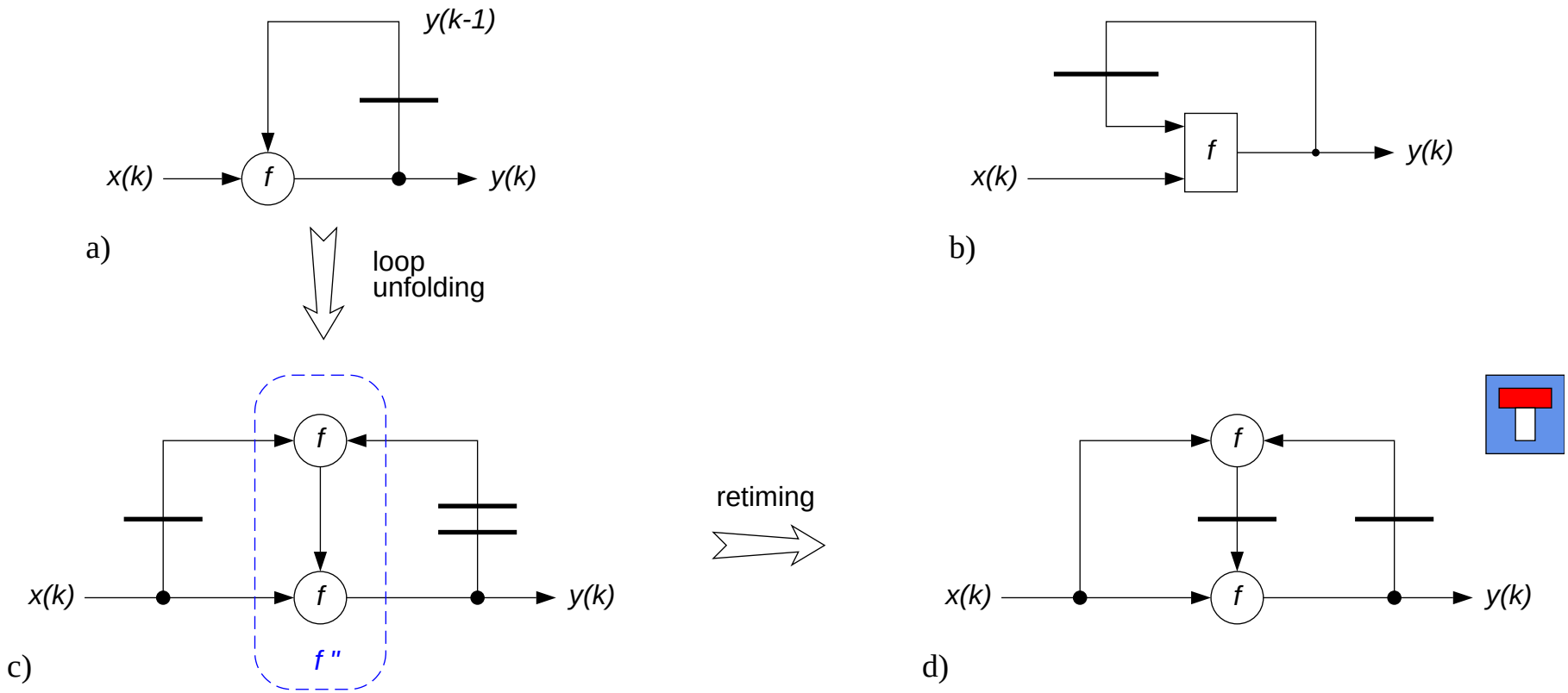


Figure: Original DDG (a) and isomorphic architecture (b), DDG after unfolding by a factor of $p = 2$ (c), same DDG with retiming added on top (d).

Nonlinear or general loops III

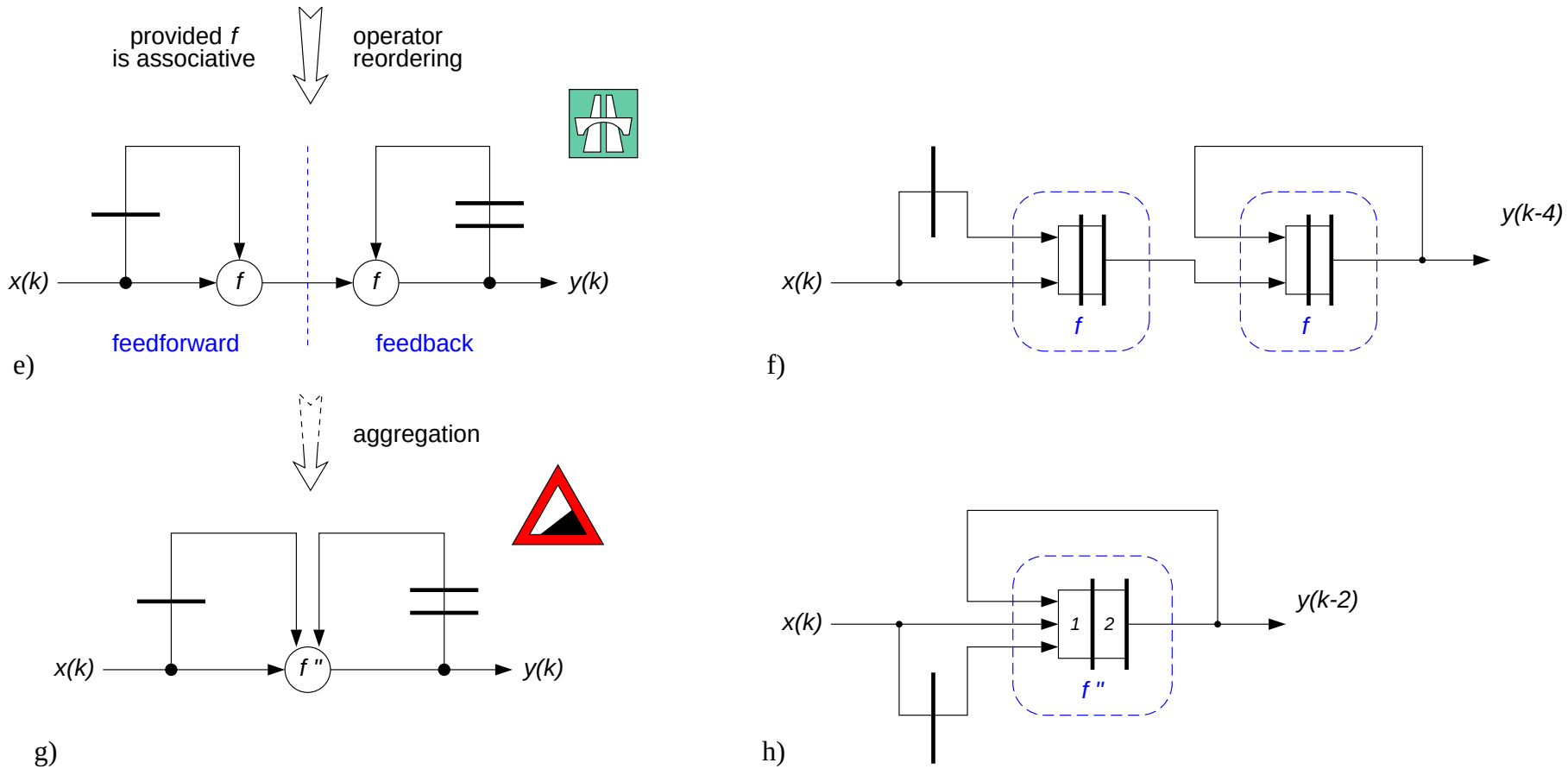


Figure: DDG with the two functional blocks for f combined into f'' (g), pertaining architecture after pipelining and retiming (h).

Limits to loop unfolding

Observation

- All successful architectural transforms for recursive computations take advantage of algorithmic properties such as linearity, fixed coefficients, associativity, limited word width or of a very limited set of register states.
- When the state size is large and the recurrence is not a closed-form function of specific classes, our methods for generating a high degree of concurrency cannot be applied.

Example: Ciphering I

In electronic codebook mode, a block of ciphertext $y(k)$ gets computed from the present block of plaintext $x(k)$ and from key $u(k)$ using some complex and non-analytical cipher function c.

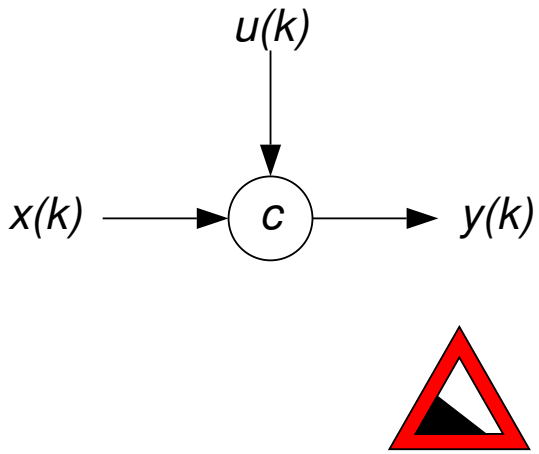


Figure: Block cipher in electronic codebook (ECB) mode.

- In search of throughput, the door is wide open for pipelining.

Example: Ciphering II

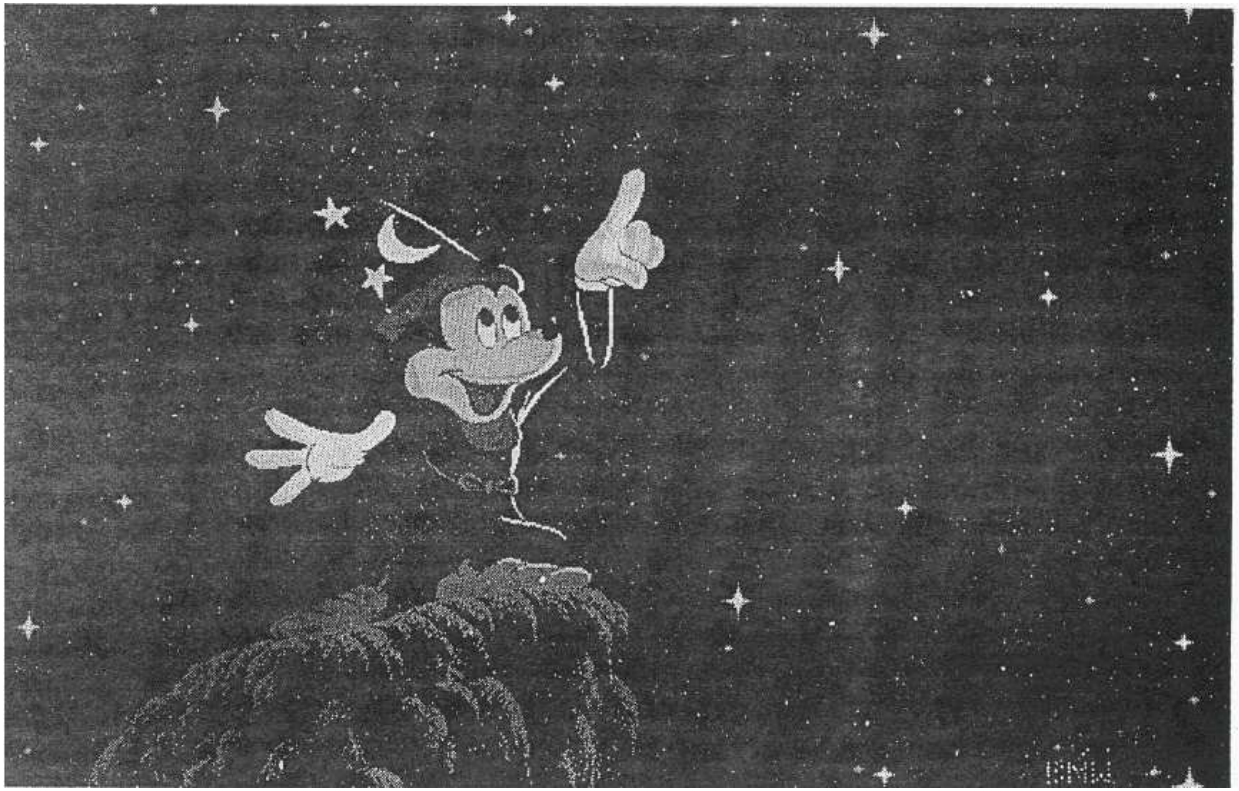


Figure: A computer graphics image in clear text.

Example: Ciphering III

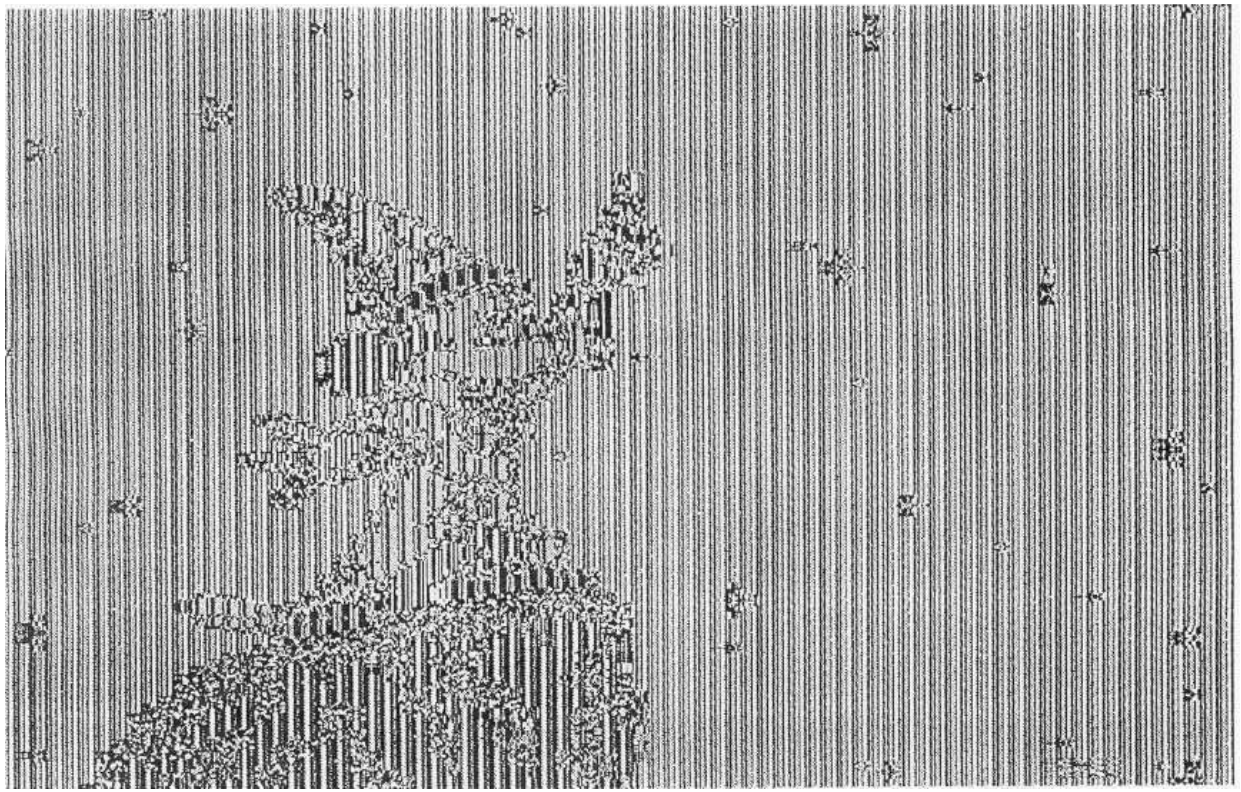


Figure: Same image ciphered in electronic codebook mode (ECB).

Example: Ciphering IV

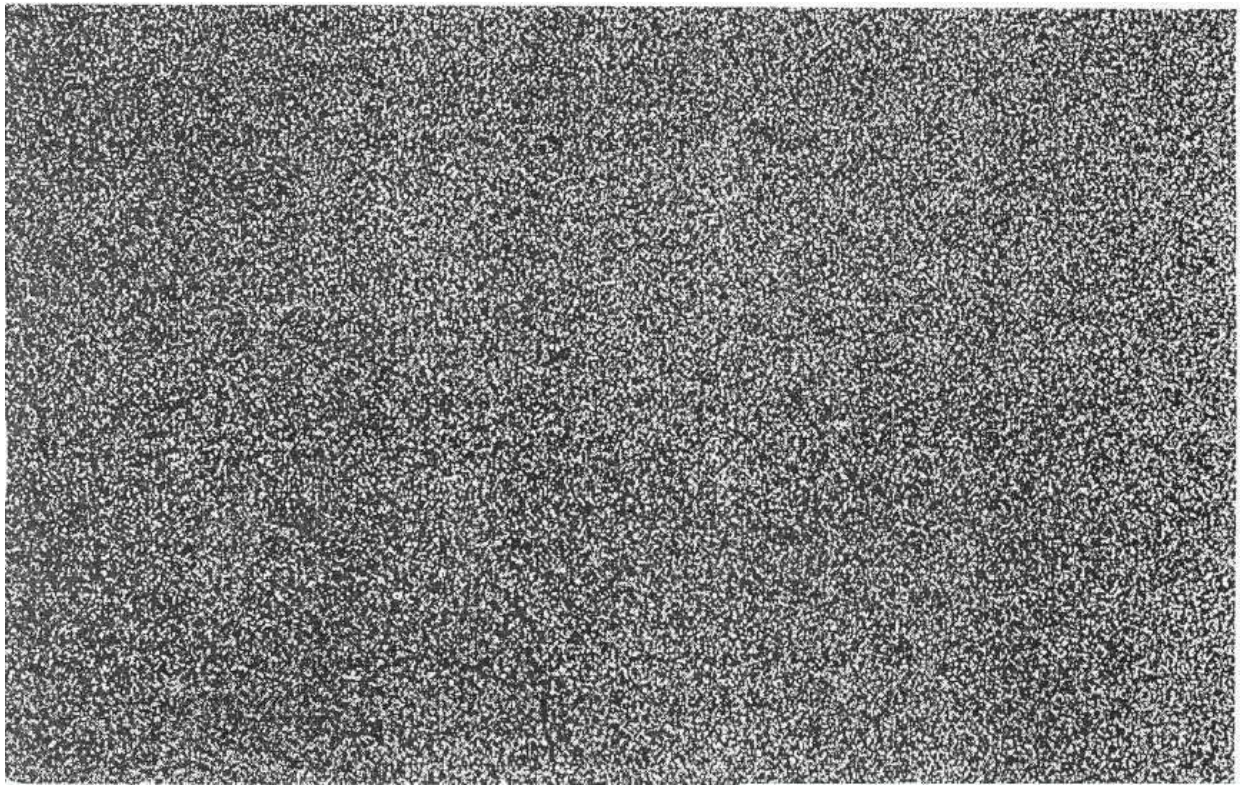


Figure: Same image ciphered in cipher back chaining mode (CBC).

Example: Ciphering V

Remedy: Cipher block chaining (CBC).

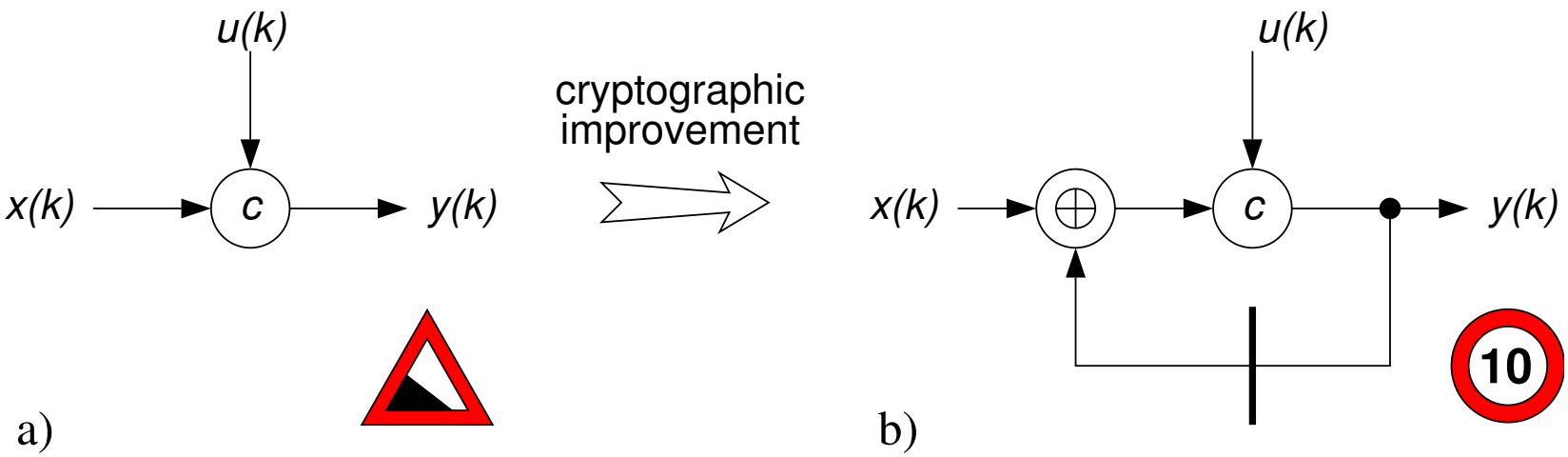


Figure: Combinational operation in ECB mode (a) vs. recursion in CBC mode (b).

- The nonlinear feedback introduced to improve cryptographic security vetoes pipelining.

Pipeline interleaving I

In search of higher throughput for a cipher in CBC mode, [2]
none of our architectural transforms applies.

Think the unthinkable!

- "What is the effect of inserting an extra register into a first-order recursive loop with the idea of pipelining the datapath?"

[2] Operating a cipher in counter mode (CTR) manages without feedback and still avoids the leakage of plaintext into ciphertext that plagues ECB. This asks for a modification at the algorithmic level, though.

Pipeline interleaving II

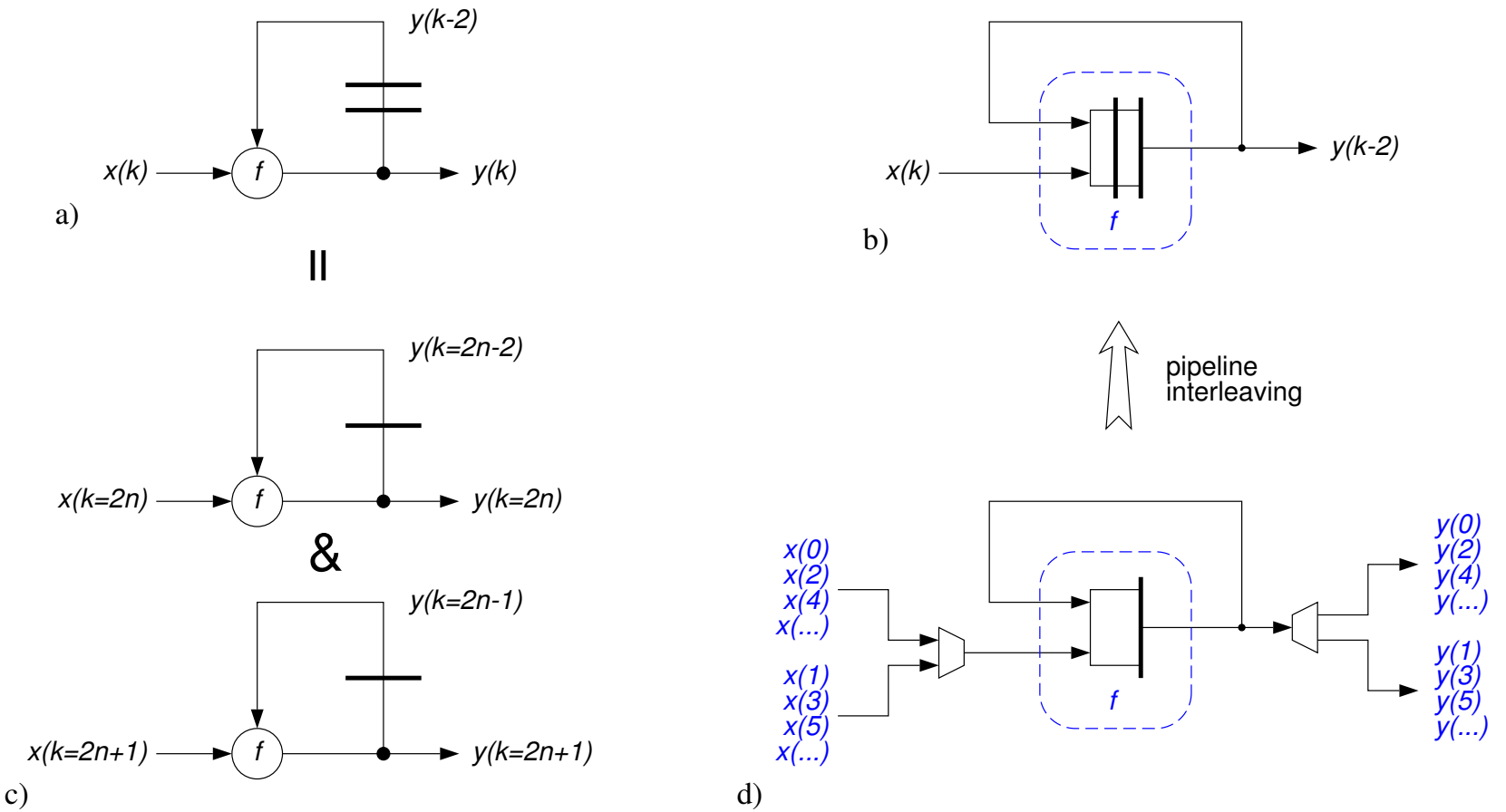


Figure: Nonlinear time-variant first-order feedback loop with one extra register inserted (a,b). Interpretation as two interleaved data streams (c,d).

Example: Ciphering revisited

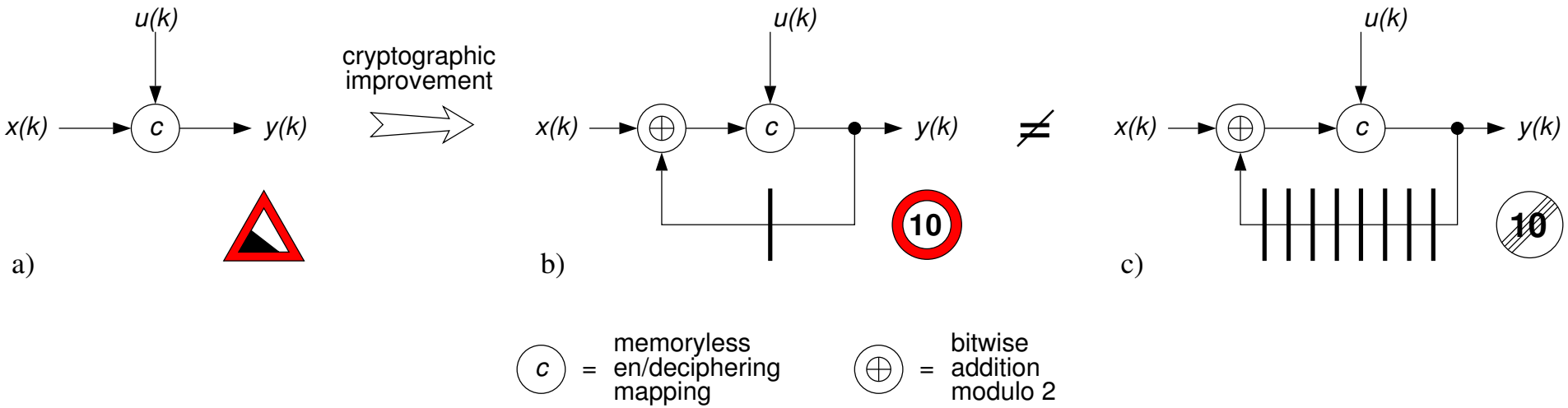


Figure: ECB mode (a), CBC mode with feedback (b), and CBC-8 operation (c).

Observation

Pipeline interleaving removes the bottleneck but alters functionality.

- Acceptable where data can be viewed as separate time-multiplexed streams that are to be processed independently from each other.

Example: Sphere decoding in a MIMO OFDM receiver I [3]

- Sphere decoding is a key subfunction in a MIMO OFDM receiver and essentially a sophisticated tree-traversal algorithm of low average search complexity.

Observation

- OFDM operates on many subcarriers at a time (typically 48 to 108).
- Each subcarrier poses an independent tree-search problem.

[3] MIMO = Multi Input Multi Output, OFDM = Orthogonal Freq. Division Multiplex

Example: Sphere decoding in a MIMO OFDM receiver II

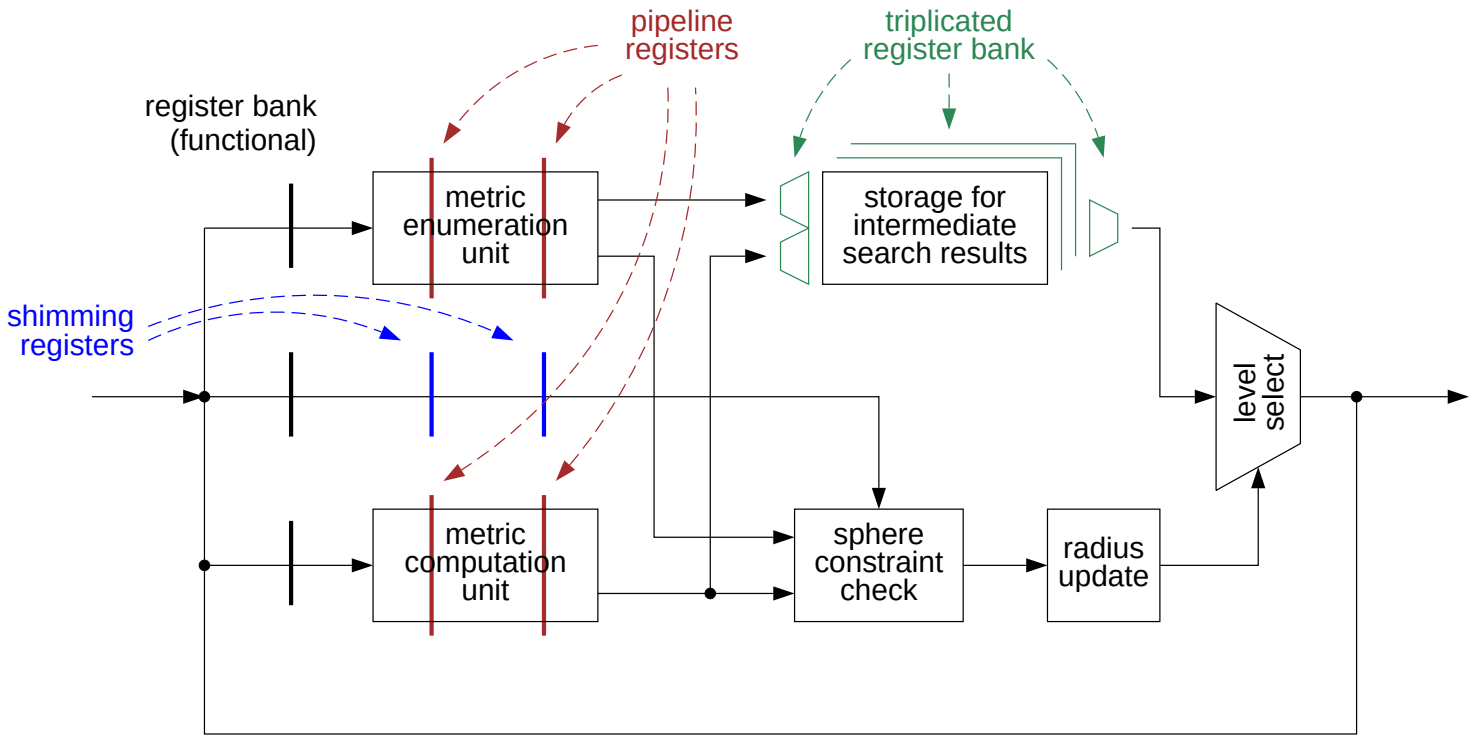


Figure: Sphere decoder; black ↦ original architecture; color items ↦ extra circuitry required to handle three individual subcarriers in an interleaved fashion.

Example: Sphere decoding in a MIMO OFDM receiver III

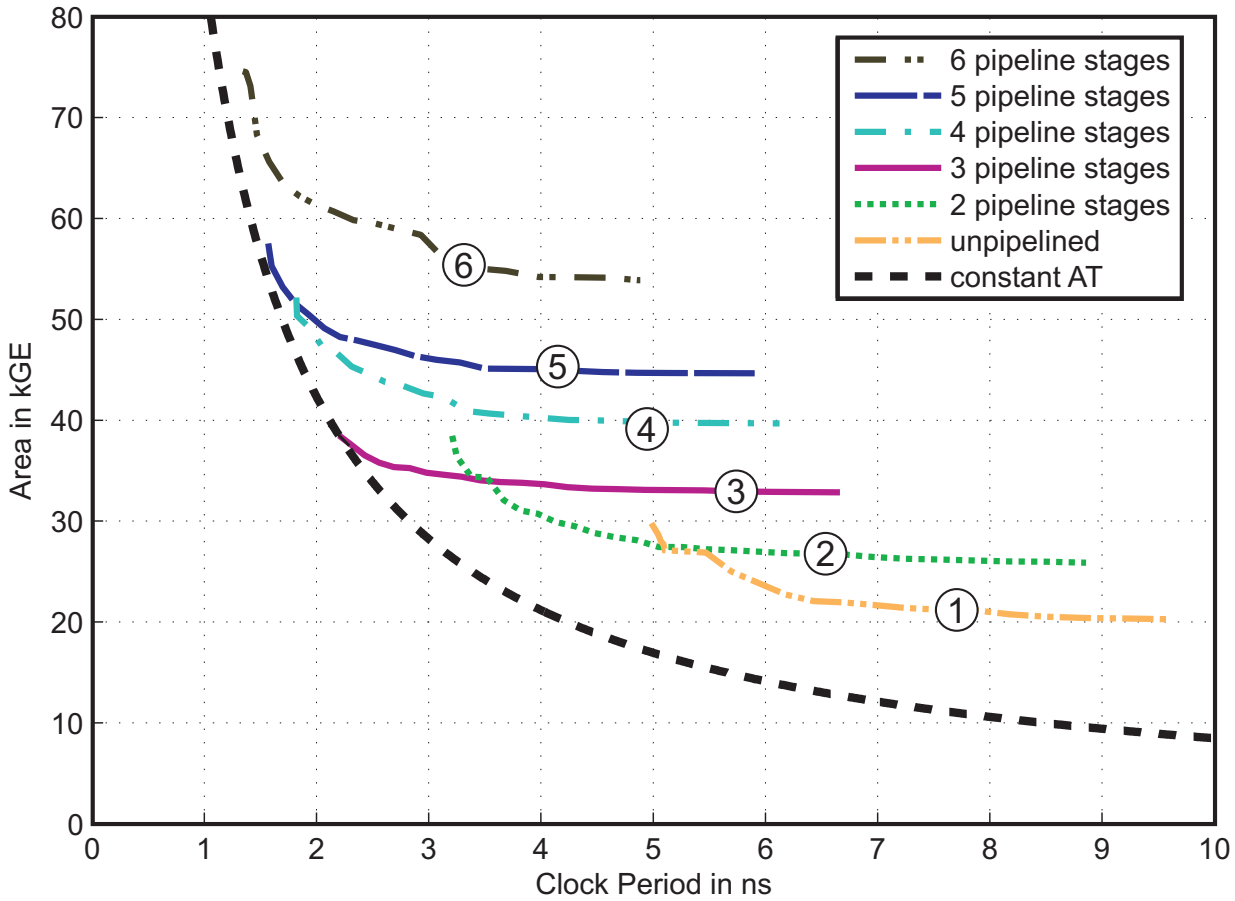


Figure: The beneficial impact of pipeline interleaving on area and throughput of a sphere decoder circuit (diagram courtesy of Dr. Markus Wenk).

Recapitulation

Loop unfolding

can significantly improve the throughput of linear time-invariant feedback calculations.

- The rapid growth of overall circuit size tends to limit economically practical unfolding degrees to fairly low values, say $p = 2...8$.
- Nonlinear feedback loops are, in general, not amenable to throughput multiplication by applying unfolding techniques.
  A notable exception exists when the loop function is associative.
- Pipeline interleaving is not an equivalence transform but nevertheless helpful where multiple data streams undergo the same processing independently from each other.

Subject

Generalizations of the transform approach

Generalization to other levels of detail

Level of abstraction	Granularity	Relevant items	
		Operations	Data
Architecture	◯	subtasks, processes	time series, pictures
Word	○	arithmetic/logic ops	words, samples, pixels
Bit	·	gate-level ops	individual bits

What if we try to apply equivalence transforms at levels of abstraction other than the word level?

Generalization to other levels of detail

Level of abstraction	Granularity	Relevant items	
		Operations	Data
Architecture	◯	subtasks, processes	time series, pictures
Word	○	arithmetic/logic ops	words, samples, pixels
Bit	·	gate-level ops	individual bits

What if we try to apply equivalence transforms at levels of abstraction other than the word level?

- Recall: DDGs are not concerned with the granularity of operations and data.

Lucky finding

Everything we have learned is applicable at multiple levels of abstraction.

Examples of transforms at the architecture level

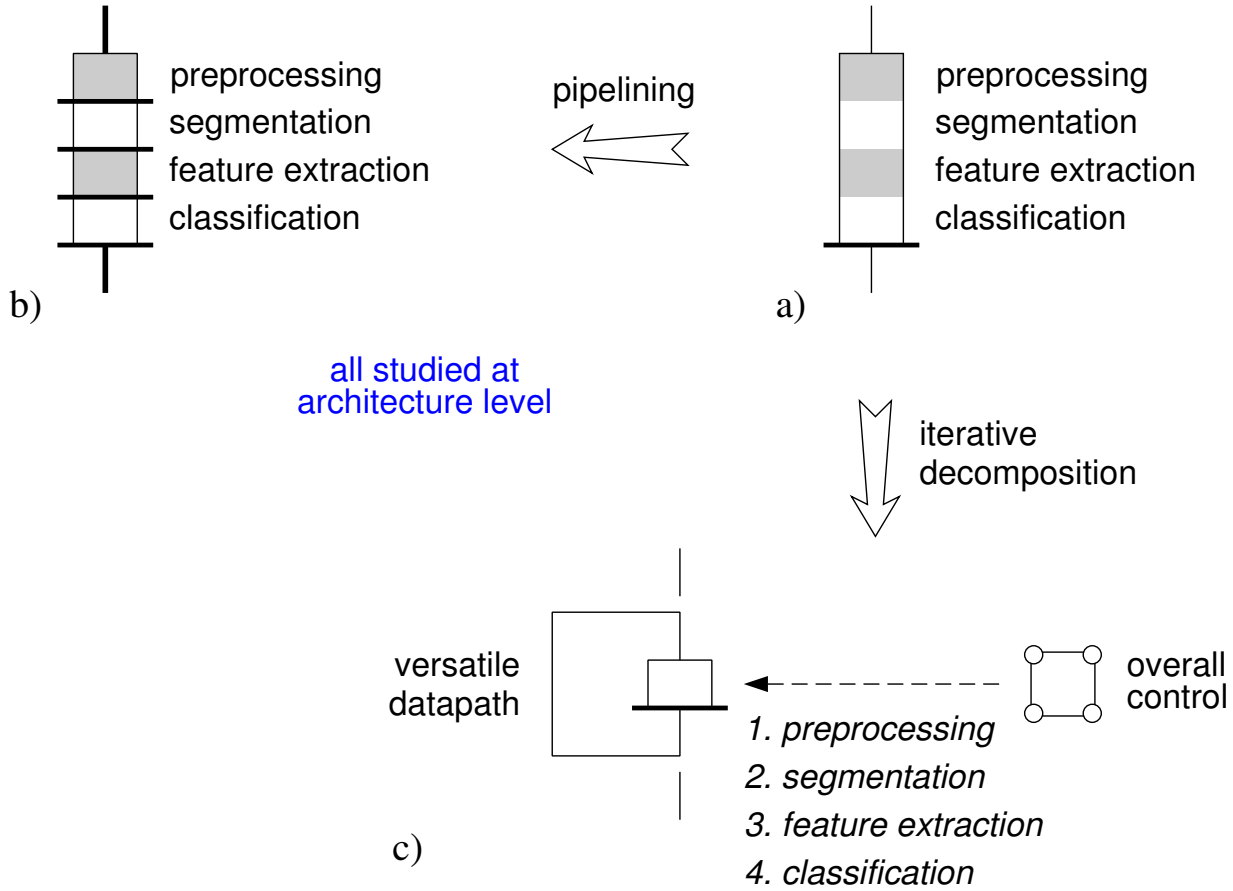


Figure: Architectural alternatives for a typical pattern recognition system.

Examples of transforms at the bit level

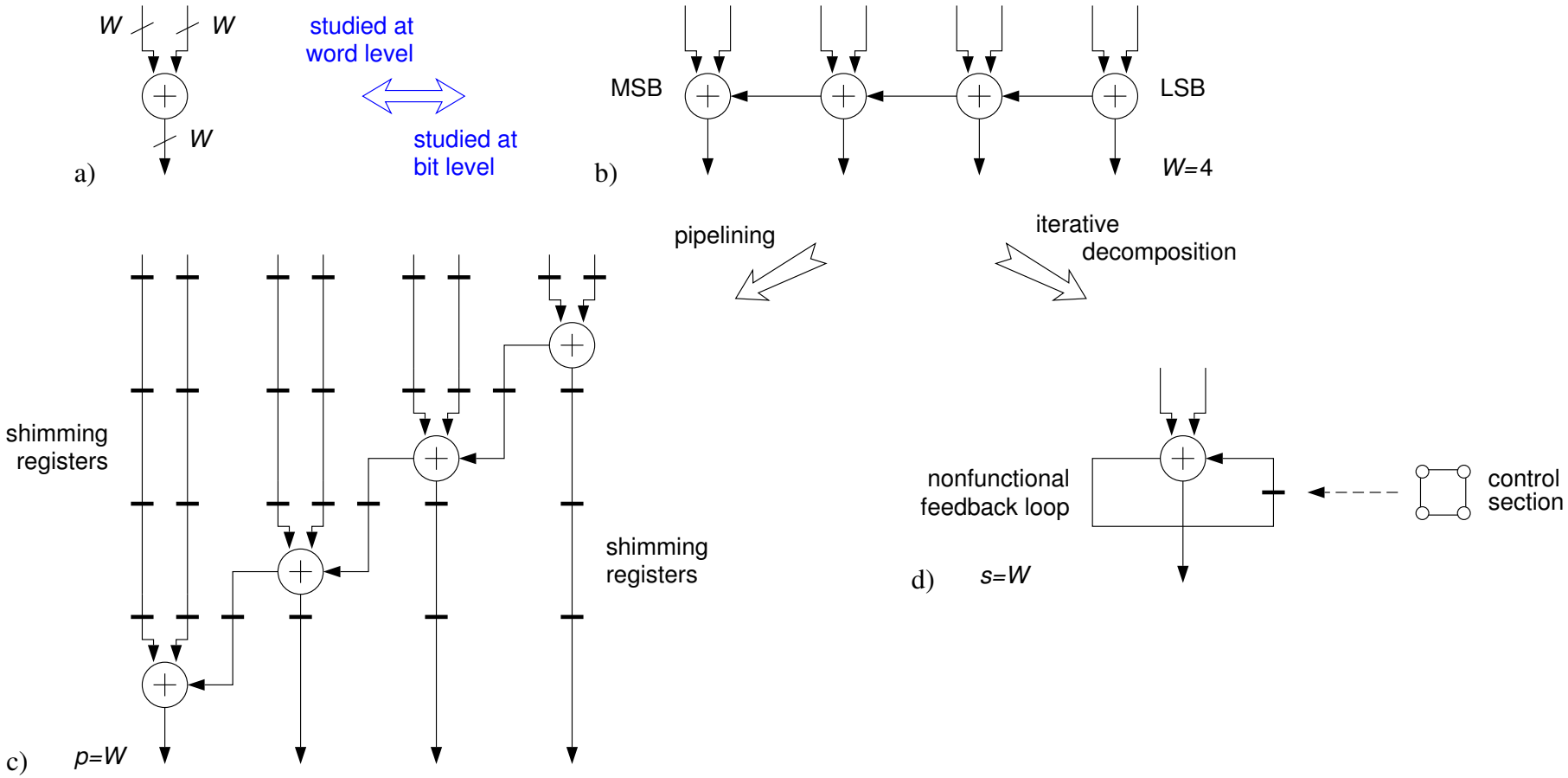
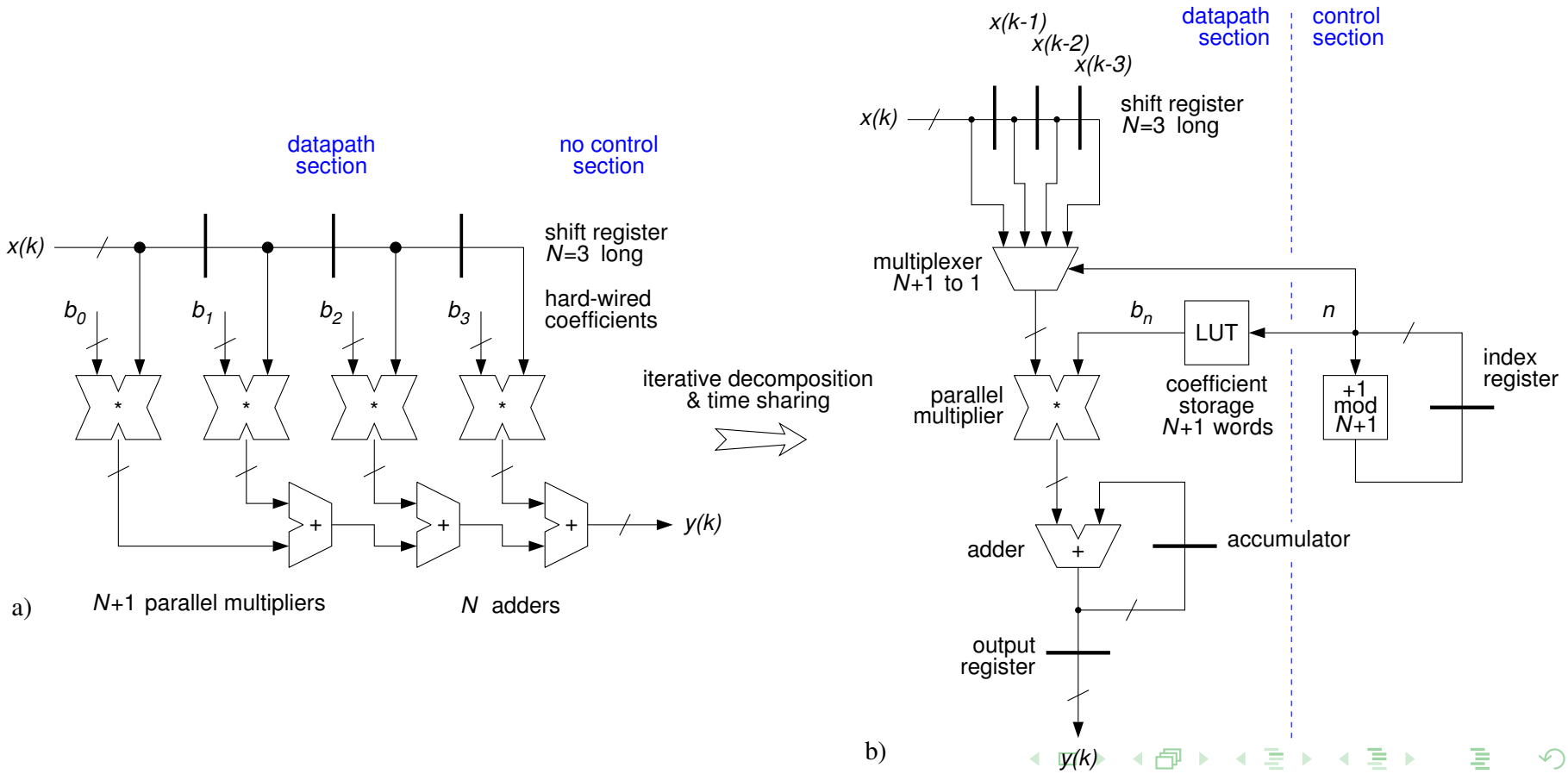


Figure: 4-bit addition (a) broken up into a ripple-carry adder (b) before being subject to pipelining (c) and iterative decomposition (d).

What we have seen so far

"Standard" datapaths. Word-level operations executed one after the other with all bits being processed simultaneously.



What we will see next

Uncommon architectural concepts where one bit from each data word is being operated upon at a time until all bits have been processed.

Bit-serial architectures.

1. Word-level operations broken up into bit-level operations.
2. Iterative decomposition.

Distributed arithmetic.

1. Word-level operations broken up into bit-level operations.
2. Algebraic transforms to get rid of multiplication.

Example of a bit-serial architecture

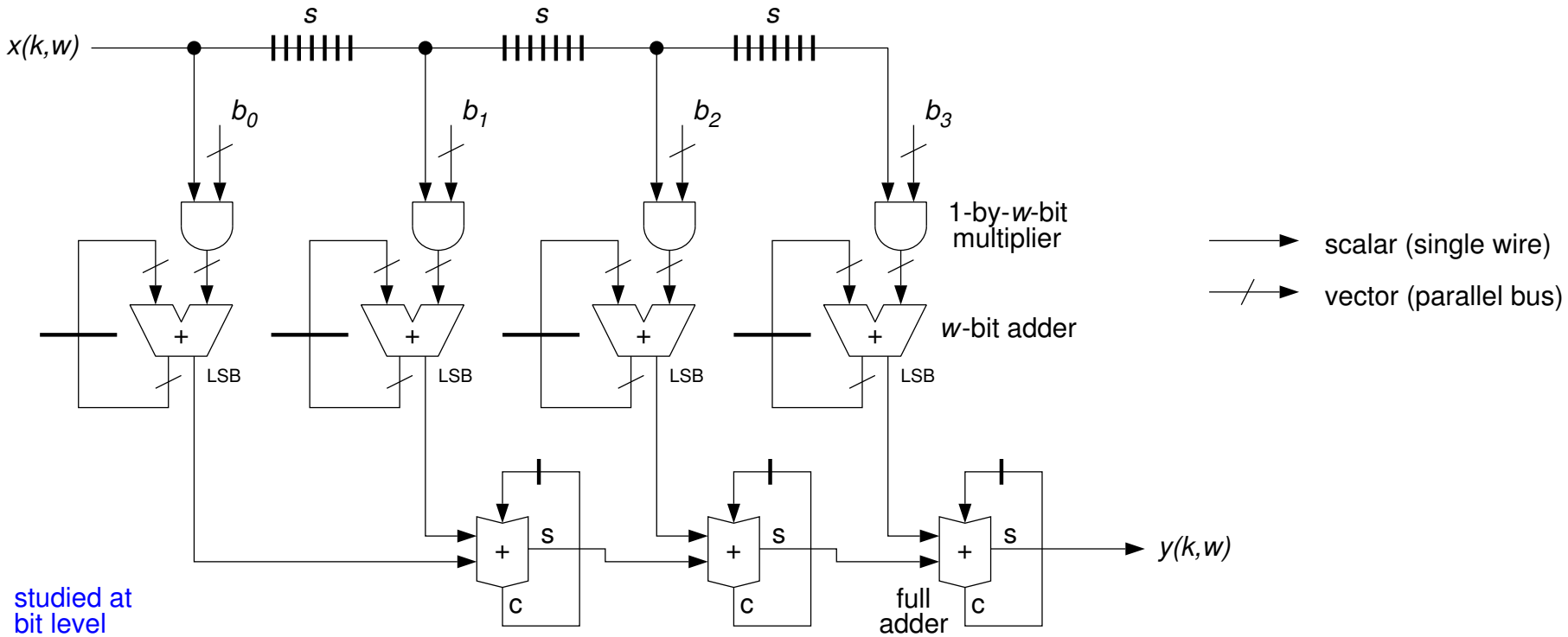


Figure: Third order transversal filter

Pros and cons of bit-serial architectures

- $\sim$ Overall hardware structure remains isomorphic with the DDG.
- $+$ Small control overhead.
- $-$ Inflexible because DDG is hardwired into the datapath with no explicit controller.
- $+$ High computation rates keep computational units busy.
- $+$ All non-local data communication is via serial links.
- $+$ Much of the data circulation is local.
- $-$ Division, data-dependent decisions, etc. ill-suited for bitwise iterative decomposition and pipelining.
- $-$ Incompatible with word-oriented RAMs and ROMs (bit-parallel), successive approximation and max./min. picking (MSB first).

Pros and cons of bit-serial architectures

- $\sim$ Overall hardware structure remains isomorphic with the DDG.
- $+$ Small control overhead.
- $-$ Inflexible because DDG is hardwired into the datapath with no explicit controller.
- $+$ High computation rates keep computational units busy.
- $+$ All non-local data communication is via serial links.
- $+$ Much of the data circulation is local.
- $-$ Division, data-dependent decisions, etc. ill-suited for bitwise iterative decomposition and pipelining.
- $-$ Incompatible with word-oriented RAMs and ROMs (bit-parallel), successive approximation and max./min. picking (MSB first).

Rule of thumb

Bit-serial architectures are at their best for unvaried real-time computations that involve operations such as addition and multiplication by a constant.

Distributed arithmetic I

Consider the calculation of the following inner product

$$y = \sum_{k=0}^{K-1} c_k x_k$$

where each c_k is a fixed coefficient. Input data x_k are scaled such that $|x_k| < 1$ and coded with a total of W bits in 2's-complement format.

$$x_k = -x_{k,0} + \sum_{w=1}^{W-1} x_{k,w}\, 2^{-w}$$

The desired output y can be expressed as

$$y = \sum_{k=0}^{K-1} c_k \left(-x_{k,0} + \sum_{w=1}^{W-1} x_{k,w}\, 2^{-w}\right)$$

Distributed arithmetic II

With distributive law, commutative law, and reversed order of summation

$$y = \sum_{k=0}^{K-1} c_k(-x_{k,0}) + \sum_{w=1}^{W-1}\left(\sum_{k=0}^{K-1} c_k x_{k,w}\right) 2^{-w}$$

The pivotal observation refers to the term in parentheses

$$\sum_{k=0}^{K-1} c_k x_{k,w} = p(w)$$

For any given bit position w, calculating the sum of products takes one bit from each of the K data words x_k, so $p(w)$ can take on no more than 2^K distinct values. With the coefficients c_k constant, all those values can be kept in a lookup table (LUT). The computation then simply becomes

$$y = -p(0) + \sum_{w=1}^{W-1} p(w)\, 2^{-w}$$

Example of distributed arithmetic

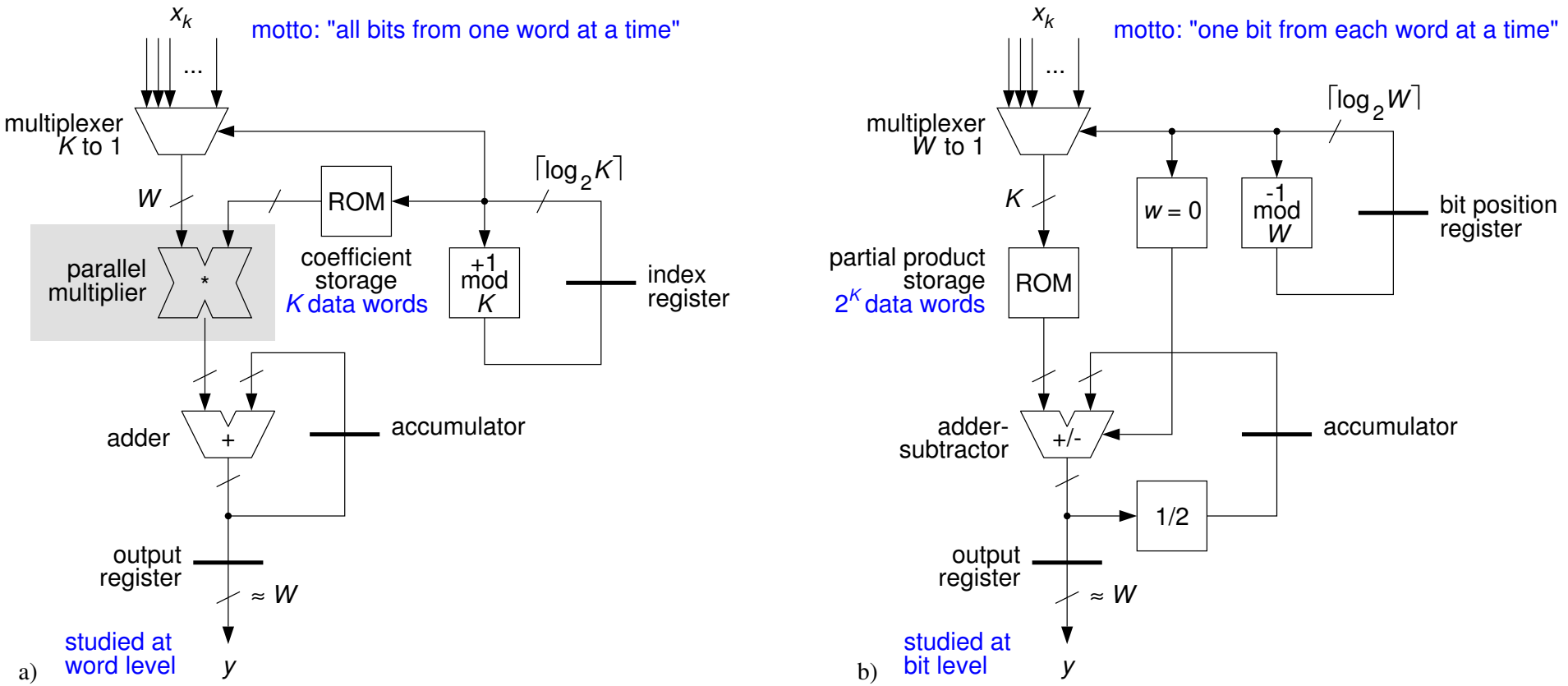


Figure: Computing a sum of products by way of repeated multiply-accumulate operations (a) and with distributed arithmetic (b).

Pros and cons of distributed arithmetic

+ No need for costly multipliers
  as these get merged with coefficient tables.
− Memory size grows exponentially
  with the order of the inner product to be computed.
∼ Mitigation techniques exist
  but depend heavily on coefficient values.

Pros and cons of distributed arithmetic

\+ No need for costly multipliers
as these get merged with coefficient tables.

– Memory size grows exponentially
with the order of the inner product to be computed.

∼ Mitigation techniques exist
but depend heavily on coefficient values.

Rule of thumb

Distributed arithmetic should be considered when

- coefficients are fixed,
- number of distinct coefficient values is small,
- hardware multipliers are expensive compared to lookup tables.

Example: DSP applications with table-based FPGAs.

Generalization to other algebraic structures I

What we have seen so far:

"Standard" computations. Filters, correlators and the like where arithmetic operations were taken from the field of reals $(\mathbb{R}, +, \cdot)$.

What we will see next:

More fields.
- with infinitely many elements, and
- with some finite number of elements.

Semirings. More general algebraic structures.

You may want to present slide set "A Brief Glossary of Algebraic Structures" at this point!

Generalization to other algebraic structures II

- All algebraic fields share a common set of axioms, so any algebraic transform that is valid in one field must necessarily hold for any other field. Universal transforms remain valid anyway.

Observation

Everything we have learned is applicable to any algebraic field.

Infinite fields. $(\mathbb{R}, +, \cdot)$ and $(\mathbb{C}, +, \cdot)$ are commonplace in digital signal processing.

Finite fields. GF(2), GF(p), GF(p^n) have numerous applications in

- data compression (source coding),
- error correction (channel coding), and
- information security (ciphering).

Example: The Viterbi algorithm I

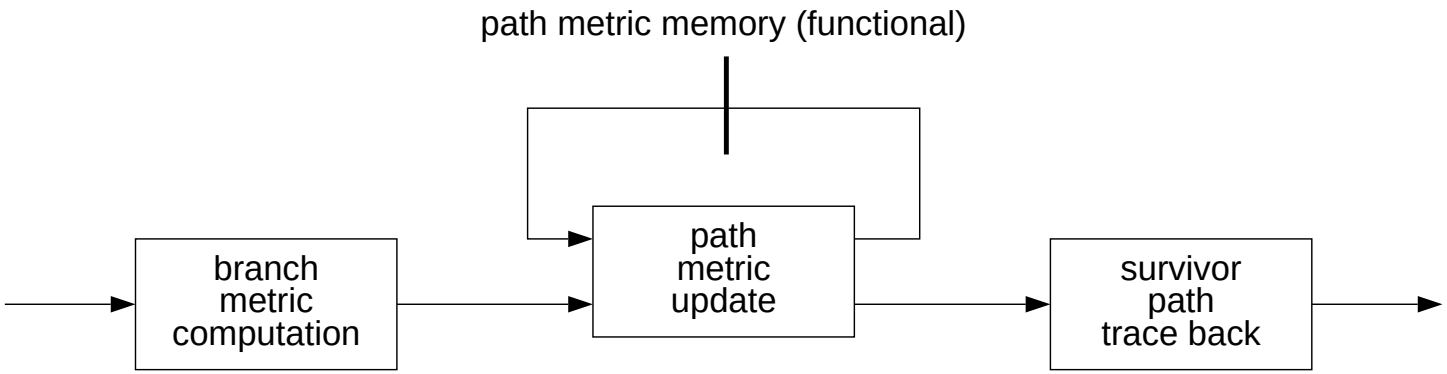


Figure: The three major steps of the Viterbi algorithm.

- Convolutional decoding is a multi-stage decision problem where Richard Bellman's principle of optimality applies: "The globally optimum solution includes no suboptimal local decision."
- Bellman has developed a technique called "Dynamic Programming", the Viterbi algorithm is a particular case thereof.

Refer to slide set "A Gentle Introduction to Dynamic Programming and the Viterbi Algorithm"!

Example: The Viterbi algorithm II

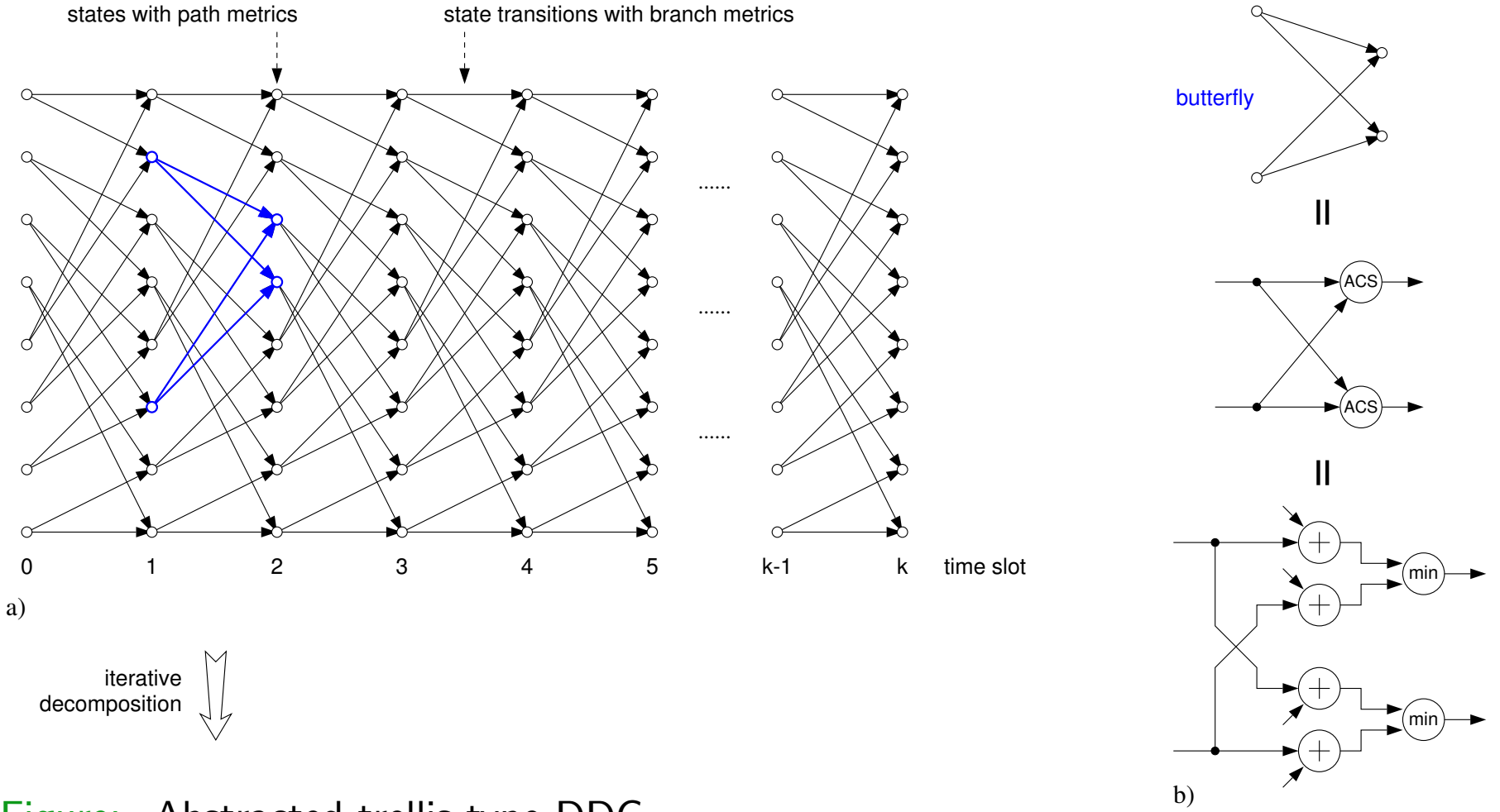


Figure: Abstracted trellis-type DDG
for path metric computation (a) with details for one butterfly (b).

Example: Architectural choices for a Viterbi decoder I

Natural choice: A datapath that computes one set of path metrics from the previous set in a single clock cycle $\mapsto$ architecture $\mathrm{d})$.

Goals and options:

Smaller circuit. Combine iterative decomposition and time sharing, ultimately leads to a processor-type datapath built around an ALU.

Reduced clock. If the longest path in architecture $\mathrm{d})$ turns out to be too fast to match that in the remainder of the circuit, a lesser degree of decomposition may prove more adequate. $\mathrm{c})$ yields roughly the same throughput with half the clock. Combinational logic gets approximately doubled, though.

Example: Architectural choices for a Viterbi decoder II

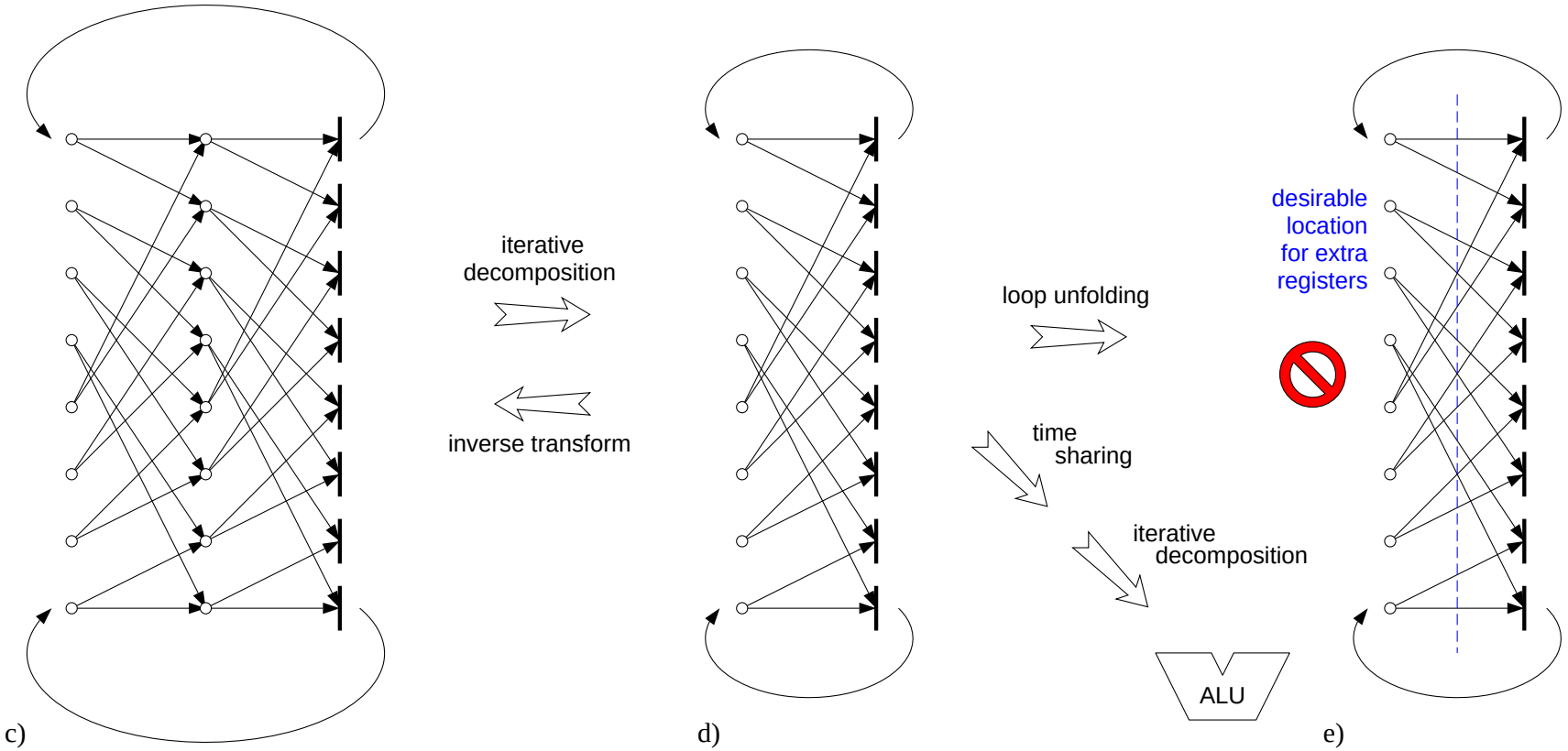


Figure: Datapath architectures obtained from different degrees of iterative decomposition (c,d). Doomed attempt to boost throughput by inserting extra latency registers into the nonlinear first-order feedback loop (e).

Example: Architectural choices for a Viterbi decoder III

Goals and options (continued):

Still higher throughput. Longest path needs to be trimmed down.

The computation in a butterfly goes

$$y_1(k) = \min(a_{11}(k) + y_1(k-1), a_{12}(k) + y_2(k-1))$$
$$y_2(k) = \min(a_{21}(k) + y_1(k-1), a_{22}(k) + y_2(k-1))$$

This is a nonlinear first-order recursion

$\rightsquigarrow$ none of our architectural transforms applies.

A more sophisticated approach is needed!

Loop unfolding revisited

Rederive substituting the generic symbols $\boxplus$ for $+$ and $\boxdot$ for $\cdot$

$$y(k) = a(k) \boxdot y(k-1) \boxplus x(k)$$

to obtain for arbitrary integer values of $p \geq 2$

$$y(k) = (\prod_{n=0}^{p-1} a(k-n)) \boxdot y(k-p) \boxplus \sum_{n=1}^{p-1} (\prod_{m=0}^{n-1} a(k-m)) \boxdot x(k-n) \boxplus x(k)$$

where $\sum$ and $\prod$ refer to operators $\boxplus$ and $\boxdot$ respectively.

- The algebraic axioms necessary for that derivation are
  - closure under both operators,
  - associativity of both operators, and
  - distributive law of $\boxdot$ over $\boxplus$.
- The algebraic structure defined by these axioms is the semiring.

Example: Boosting throughput of a Viterbi decoder I

Now consider a semiring where

- Set of elements: $S = \mathbb{R} \cup \{\infty\}$,
- Algebraic addition: $\boxplus = \min$, and
- Algebraic multiplication: $\boxdot = +$.

Example: Boosting throughput of a Viterbi decoder I

Now consider a semiring where

- Set of elements: $S = \mathbb{R} \cup \{\infty\}$,
- Algebraic addition: $\boxplus = \min$, and
- Algebraic multiplication: $\boxdot = +$.

The reformulated ACS operation now goes

$$y_1(k) = a_{11}(k) \boxdot y_1(k-1) \boxplus a_{12}(k) \boxdot y_2(k-1)$$
$$y_2(k) = a_{21}(k) \boxdot y_1(k-1) \boxplus a_{22}(k) \boxdot y_2(k-1)$$

which, making use of vector and matrix notation, can be rewritten as

$$\vec{y}(k) = A(k) \boxdot \vec{y}(k-1)$$

Example: Boosting throughput of a Viterbi decoder I

Now consider a semiring where

- Set of elements: $S = \mathbb{R} \cup \{\infty\}$,
- Algebraic addition: $\boxplus = \min$, and
- Algebraic multiplication: $\boxdot = +$.

The reformulated ACS operation now goes

$$y_1(k) = a_{11}(k) \boxdot y_1(k-1) \boxplus a_{12}(k) \boxdot y_2(k-1)$$
$$y_2(k) = a_{21}(k) \boxdot y_1(k-1) \boxplus a_{22}(k) \boxdot y_2(k-1)$$

which, making use of vector and matrix notation, can be rewritten as

$$\vec{y}(k) = A(k) \boxdot \vec{y}(k-1)$$

- Note, this is a linear first-order recursion!

Example: Boosting throughput of a Viterbi decoder II

By replacing $\vec{y}(k-1)$ one gets the unfolded recursion for $p = 2$

$$\vec{y}(k) = A(k) \boxdot A(k-1) \boxdot \vec{y}(k-2)$$

To take advantage of this unfolded form,
the product $B(k) = A(k) \boxdot A(k-1)$ must be computed outside the loop.

Resubstituting the original operators and variables we obtain the recursion

$$y_1(k) = \min(b_{11}(k) + y_1(k-2), b_{12}(k) + y_2(k-2))$$
$$y_2(k) = \min(b_{21}(k) + y_1(k-2), b_{22}(k) + y_2(k-2))$$

which includes the same number and types of operations as the original formulation but allows for twice as much time.

Example: Boosting throughput of a Viterbi decoder III

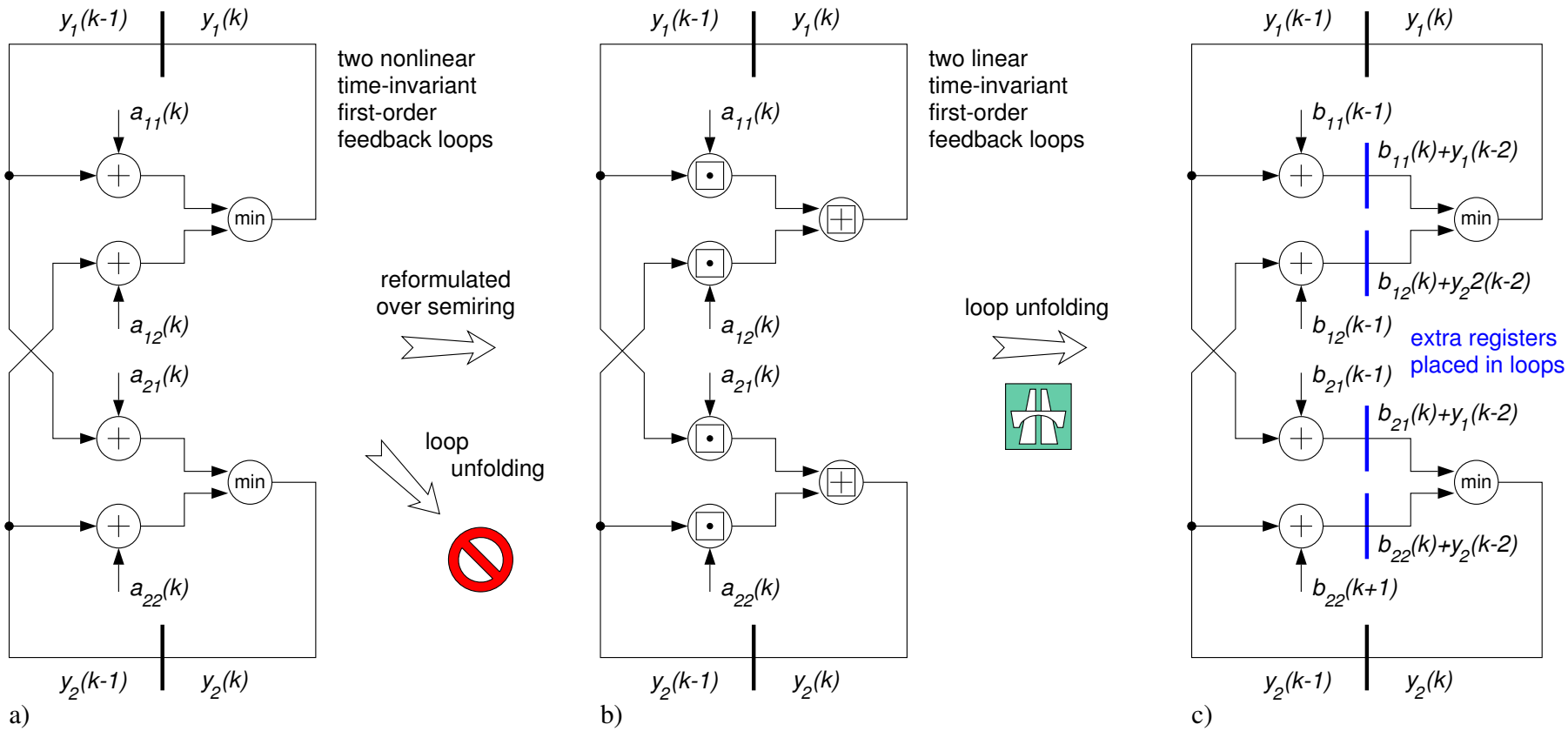


Figure: The first-order recursion of the Viterbi algorithm before (a) and after being reformulated over a semiring (b), with loop unfolding added on top (c).

Example: Boosting throughput of a Viterbi decoder IV

The price to pay is the extra hardware required to perform the non-recursive computations outside the loop

$$
\begin{aligned}
b_{11}(k) &= \min(a_{11}(k) + a_{11}(k-1), a_{12}(k) + a_{21}(k-1)) \\
b_{12}(k) &= \min(a_{11}(k) + a_{12}(k-1), a_{12}(k) + a_{22}(k-1)) \\
b_{21}(k) &= \min(a_{21}(k) + a_{11}(k-1), a_{22}(k) + a_{21}(k-1)) \\
b_{22}(k) &= \min(a_{21}(k) + a_{12}(k-1), a_{22}(k) + a_{22}(k-1))
\end{aligned}
$$

in a heavily pipelined way.

Insight gained

Compare the two formulations of the same problem:

- Nonlinear recursion over field, not amenable to loop unfolding.
- Linear recursion over semiring, amenable to loop unfolding.

Conclusion

Taking advantage of specific properties of an algorithm and of algebraic transforms has more potential to offer than universal transforms alone.

- Some computations can be accelerated by creating concurrencies that did not exist in the original formulation.

$\rightsquigarrow$ Opens a door to solutions that would otherwise remain off-limits.

Subject

Summary and conclusions

Options available for reorganizing datapath architectures

		Type of computation		
		combinational (memoryless)	sequential (memorizing) non-recursive	recursive
	Data flow	feedforward	feedforward	feedback
	Memory	no	yes	yes
	Data dependency graph	DAG with all edge weights zero	DAG with some or all edge weights non-zero	Directed cyclic graph with no circular path of weight zero
	Response length	$M = 1$	$1 < M < \infty$	$M = \infty$
Nature of system	linear time-invariant	D,P,Q,S,a	D,P,q,S,a,R	D,S,a,R,i,U
	linear time-variant	D,P,Q,S,a	D,P,S,a,R	D,S,a,R,i,U
	nonlinear	D,P,Q,S,a	D,P,S,a,R	D,S,a,R,i,u

D : Iterative decomposition
P : Pipelining
Q : Replication
S : Time sharing
a : Associativity transform provided operations are identical and associative
R : Retiming
i : Pipeline interleaving
U : Loop unfolding
u : Loop unfolding provided computation is linear over a semiring

Important architectural transforms and their characteristics

Architectural transform	Decomposition	Pipelining	Replication	Time sharing	Associativity	Retiming	Loop unfolding
Kind	universal	universal	universal	universal	algebraic	universal	algebraic
Applicable to	combinational computations					sequential computations	
Impact on						nonrecurs.	recursive
A	$-\ldots =$	$= \ldots +$	$+$	$-\ldots =$	$=$	$=$	$+$
Γ	$+$	$=$	$-$	$+$	$=$	$=$	$=$
t_{lp}	$-$	$-$	$=$, mux $-$	$=$	$-\ldots +$	$-$	$-$
$T = \Gamma \cdot t_{lp}$	$=$	$-$	$-$	$+$	$-\ldots +$	$-$	$-$
AT	$-\ldots =$	$-\ldots =$	$=$	$= \ldots +$	$-\ldots +$	$-$	$+$
L	$+$	$+$	$=$, mux $+$	$+$	$=$	$=$	$+$
E	$-\ldots +$	$-\ldots +$	$=$	$= \ldots +$	$-\ldots +$	$=$	$+$
Extra hardware overhead	recy. and cntl.	none	distrib., recoll., and cntl.	collect., redist., and cntl.	none	none	extra word width
Helpful for indirect energy saving	no	coarse grain yes	possibly yes	no	yes	yes	possibly yes
Compatible storage type	any	register	register	any	any	register	register

Power and energy considerations

What is meant by "Helpful for indirect energy saving"?

- In CMOS, the most effective way to cut the energy spent per operation is to lower the supply voltage.
- The long paths through a circuit are likely to become unacceptably slow and need to be trimmed to recover clock rate and throughput.
- Architectural transforms that help do so with no circuit overhead:
  - Retiming
  - Chain/tree conversion (and other algebraic transforms)
  - Coarse grain pipelining (small overhead only)

  Benefits must be examined in detail on a per case basis!

Power and energy considerations

What is meant by "Helpful for indirect energy saving"?

- In CMOS, the most effective way to cut the energy spent per operation is to lower the supply voltage.
- The long paths through a circuit are likely to become unacceptably slow and need to be trimmed to recover clock rate and throughput.
- Architectural transforms that help do so with no circuit overhead:
  - Retiming
  - Chain/tree conversion (and other algebraic transforms)
  - Coarse grain pipelining (small overhead only)

  Benefits must be examined in detail on a per case basis!

Simple fact

Over the first decade of the 21th century,
energy efficiency has become even more important than die size.

The grand alternatives from an energy point of view I

- Processor-type architectures rely on
  - general-purpose multi-operation ALUs
  - generic register files of generous capacity
  - multi-driver busses, bus switches, multiplexers, and the like
  - uniform and often oversized datapath width
  - program and data memories along with address generation
  - controllers, program sequencers, and iteration counters
  - instruction fetching and decoding
  - stack operations and interrupt handling
  - dynamic reordering of operations
  - branch prediction and speculative execution
  - data shuffling between main memory and multiple levels of cache

The grand alternatives from an energy point of view I

- Processor-type architectures rely on
  - general-purpose multi-operation ALUs
  - generic register files of generous capacity
  - multi-driver busses, bus switches, multiplexers, and the like
  - uniform and often oversized datapath width
  - program and data memories along with address generation
  - controllers, program sequencers, and iteration counters
  - instruction fetching and decoding
  - stack operations and interrupt handling
  - dynamic reordering of operations
  - branch prediction and speculative execution
  - data shuffling between main memory and multiple levels of cache

Observation

All of this is a tremendous waste of energy
as none of the above contributes to payload data processing!

Aside

Does the total absence of unproductive computations imply the isomorphic architecture is the most energy-efficient option then?

Aside

Does the total absence of unproductive computations imply the isomorphic architecture is the most energy-efficient option then?

Not necessarily.

Reasons:

- Glitching (redundant switching during transients) $\mapsto$ most intense when data recombine in combinational logic after having travelled along propagation paths of disparate lengths.
- Leakage (static transistor currents) $\mapsto$ everything else being equal, a smaller circuit tends to have fewer leakage paths.

The grand alternatives from an energy point of view II

- The impressive throughputs of modern processors have been bought by operating CMOS circuits under conditions that are far from optimal
  - extremely fast clock,
  - large overdrive factors,
  - comparatively high supply voltage,
  - low MOSFET threshold voltages and, hence,
  - significant leakage.

Consequence

A program-controlled processor may dissipate 100 to 1000 times as much energy for the same calculation as an application-specific circuit.

The grand alternatives from an energy point of view III

"To achieve long battery life when playing video, mobile devices must decode the video in hardware (on the GPU); decoding it in software (on the CPU) uses too much power. ... The difference is striking: on an iPhone (4), for example, H.264 videos play for up to 10 h, while videos decoded in software play for less than 5 h before the battery is fully drained." (Steve Jobs, 2010)

Imperative

Increasing performance in applications with a limited power budget (all today), requires that the amount of energy spent per payload operation be lowered.

because $P = \Theta \cdot E$

In depth discussion to follow in chapter 9 "Energy Efficiency and Heat Removal".

The grand alternatives from an energy point of view IV

- The challenge of power-constrained architecture design is to
  - minimize redundant switching activities,
  - provide as just as much flexibility as required,
  - keep the effort for design and verification within reasonable bounds,

  all at a time.

The grand alternatives from an energy point of view IV

- The challenge of power-constrained architecture design is to
  - minimize redundant switching activities,
  - provide as just as much flexibility as required,
  - keep the effort for design and verification within reasonable bounds,

  all at a time.

$\rightsquigarrow$ Finding clever combinations between hardwired units and program-controlled processors asks for creativity and methodical work.

A guide to evaluating architectural alternatives I

1. Begin by analyzing the algorithm. Give quantitative indications for
   - the data rates between all major building blocks,
   - the word widths,
   - the memory bounds and access schemes for all building blocks, and
   - the computation rates for all major arithmetic operations.
2. Look for simplifications and optimizations in the algorithmic domain.
3. Examine the control flow.
   Find out where to go for a hard-wired dedicated architecture, where for a program-controlled processor, and where to look for a compromise.
4. Let your intuition come up with preliminary architectural concepts.
   Establish a rough block diagram for each of them.
   Have boundaries between major subfunctions coincide with registers.

A guide to evaluating architectural alternatives II

5. Prepare a spreadsheet that opposes all architectures considered.
6. Estimate
   - overall circuit size,
   - computation period,
   - latency, and
   - dissipated energy.

   Synthesize, place and route time-critical portions
   as propagation delays often depend on lower-level details.
7. Identify bottlenecks and inacceptably burdensome subfunctions.
   Improve with the aid of architecture transforms.
8. Compare. Then narrow down your choice.

A guide to evaluating architectural alternatives II

5. Prepare a spreadsheet that opposes all architectures considered.
6. Estimate
   - overall circuit size,
   - computation period,
   - latency, and
   - dissipated energy.

   Synthesize, place and route time-critical portions
   as propagation delays often depend on lower-level details.
7. Identify bottlenecks and inacceptably burdensome subfunctions.
   Improve with the aid of architecture transforms.
8. Compare. Then narrow down your choice.

Concluding remark

Architecture design is more art than science.