Hardware/Software Co-design and the SPLASH Effect
CEG 4392 Lecture

Rami Abielmona
SMRLab – SITE – University of Ottawa
January 18, 2006
rabielmo@site.uottawa.ca

Presentation Outline

- Brief History of Computers
- Embedded Systems Overview
- Co-specification & Co-synthesis
- Co-simulation & Co-execution
- Co-design languages and environments
- The SPLASH Effect
- Reconfigurable Computing
- Reconfigurable Architectures
- Run-Time Environment
- Run-Time Reconfigurability
- The Alternatives
- Potential Applications
- References

Generational History of Computers

- **1st generation** (1945-1955)
  - Stored program, assembly → machine language instructions (assembler)
  - Vacuum tubes used for logic
  - Magnetic core memories were invented
- **2nd generation** (1956-1965)
  - HLL used (Fortran), HLL → assembly language instructions (compiler)
- **3rd generation** (1966-1975)
  - ICs were invented and used as processor and memories
  - Microprogramming, parallelism and pipelining
  - Cache and virtual memory (VM) developed
- **4th generation** (1976-Today)
  - VLSI started being efficient (microprocessor)
  - Concurrency, pipelining, caches and VM schemes evolved
  - LANs, WANs, Internet and WWW flourished
- **5th generation** (?-?)
  - Embedded systems: calculators, pagers, watches
  - Artificial intelligence (AI): symbolic processors, cognitive computers
  - Massively parallel systems: evolvable computers (evolutionary algorithms)
  - Distributed systems: network computers (NCs), real-time control

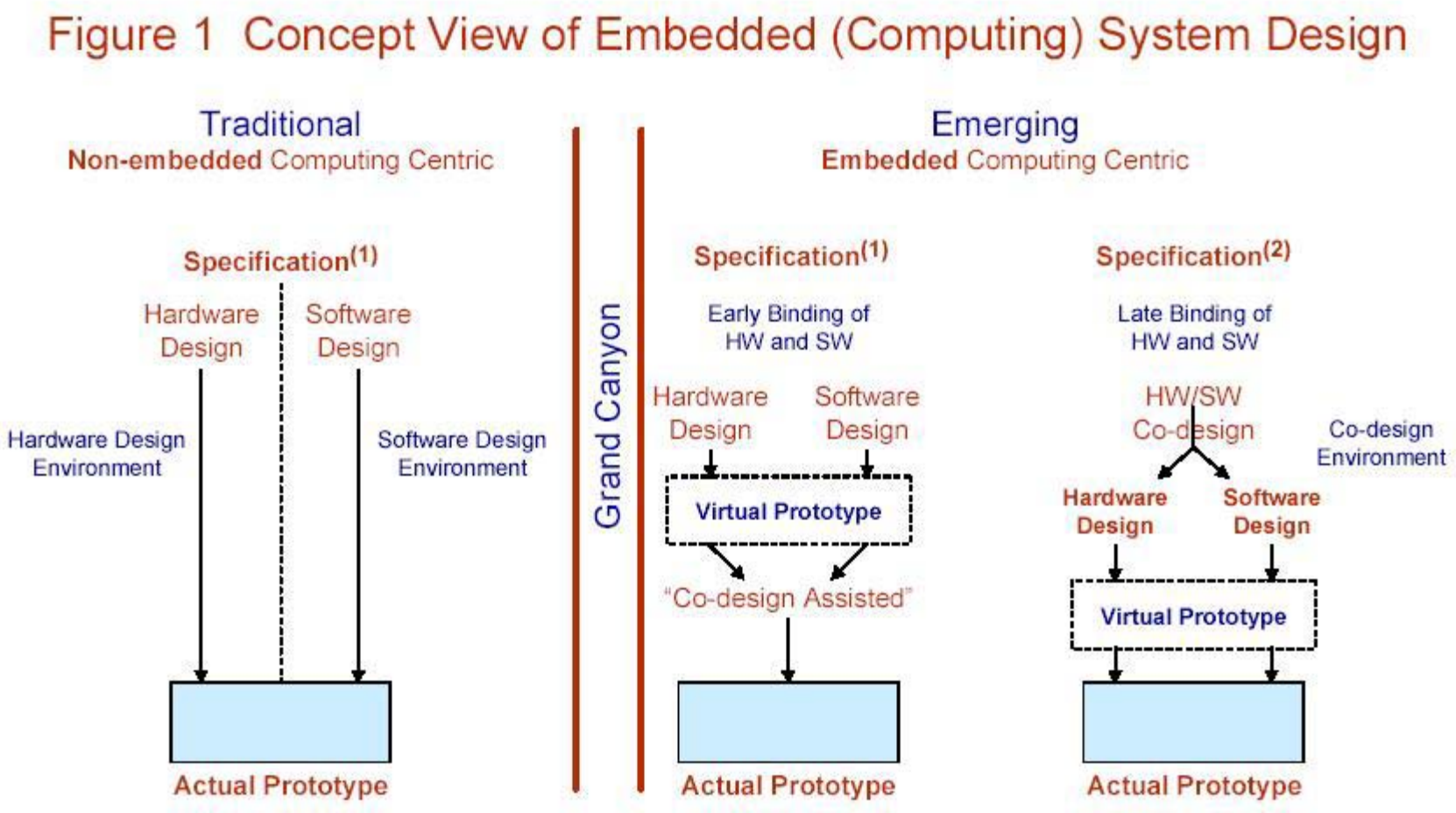
Flynn's Classifications (1972) [ES-1]

- SISD – Single Instruction stream, Single Data stream
  - Conventional sequential machines
  - Program executed is instruction stream, and data operated on is data stream
- SIMD – Single Instruction stream, Multiple Data streams
  - Vector machines (superscalar)
  - Processors execute same program, but operate on different data streams
- MIMD – Multiple Instruction streams, Multiple Data streams
  - Parallel machines
  - Independent processors execute different programs, using unique data streams
- MISD – Multiple Instruction streams, Single Data stream
  - Systolic array machines
  - Common data structure is manipulated by separate processors, executing different instruction streams (programs)

Embedded Systems Overview (1)

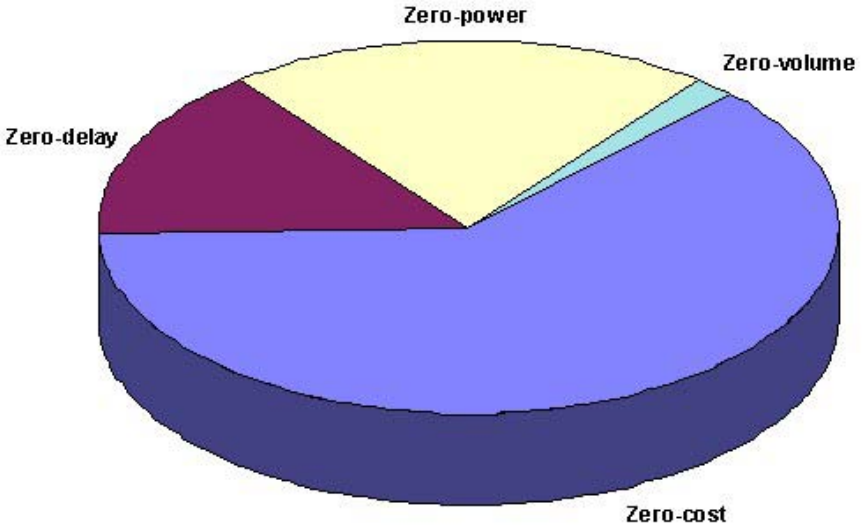
- What is an embedded system ?
  - An architecture that can execute an application-specific function, while meeting all performance, cost, size, weight and power requirements
  - Became popular in the 1980s
  - Differs from the data-processing (non-embedded) architectures, as the latter are more general-purpose and less performance- or requirement-driven
- Hardware/software co-design
  - A solution to system objectives through the concurrent design of both hardware and software components, by the exploitation of their trade-offs

Embedded Systems Overview (2) [ES-2]



(1) *Heterogeneous (use of specific languages for hardware and software components)*
(2) *Homogeneous (use of a single language for the specification of the overall system)*

Embedded Systems Overview (3) [ES-3]



Embedded systems market breakdown

Zero-delay – printers, copiers, scanners

Zero-power – cellulars, pagers, watches, cameras

Zero-cost – blenders, TVs, radios

Zero-volume – military, supercomputers

- Next-generation devices must be based on low-cost, low-power and extremely fast electronic circuits
- Cannot count on hardware design (leaves out malleability of software)
- Cannot count on software design (leaves out inherent parallelism of hardware)
- We need a better-suited computing paradigm!

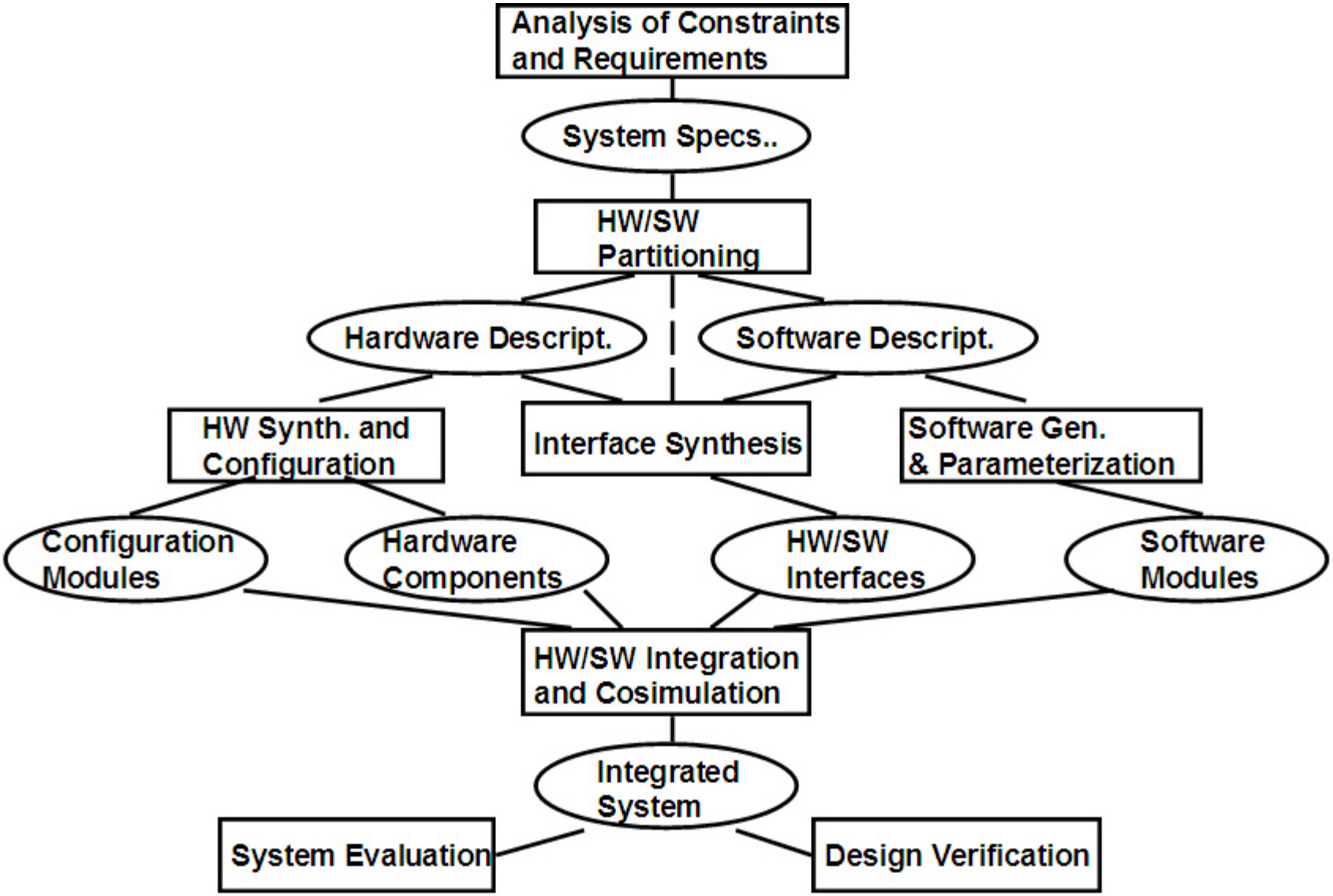
Co-specification & Co-synthesis

- Co-specification involves the creation of system specifications that describe both the HW and SW elements, and their relationships
- Co-synthesis is the (semi-)automatic design of HW and SW to meet a specification
  - Scheduling computations
  - Allocating computations to processing elements (PEs)
  - Partitioning functionalities into computational units
  - Mapping computational units to HW or SW elements

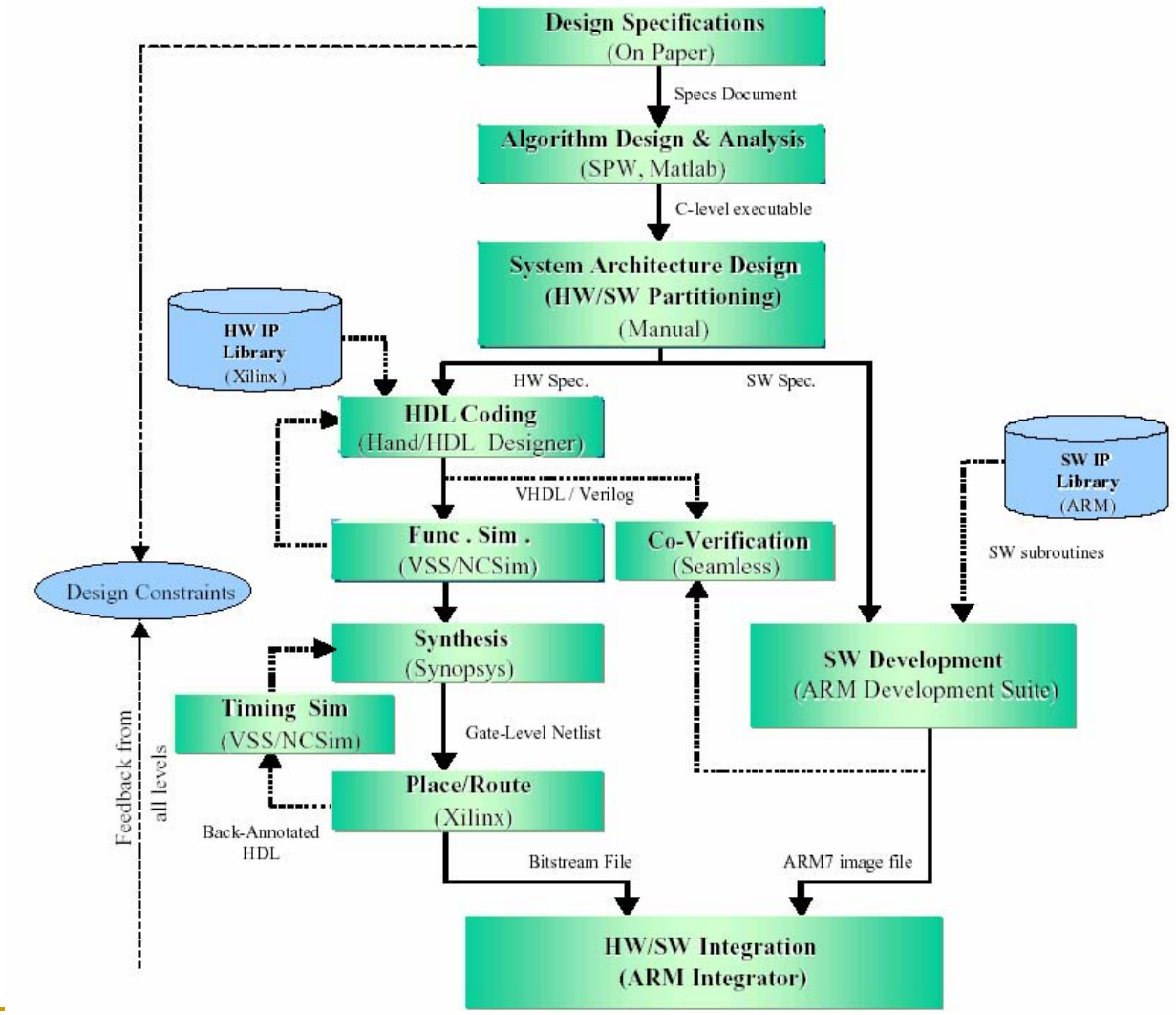
Co-simulation & Co-execution

- Co-simulation involves the concurrent simulation of HW and SW elements, at different abstraction levels
- Co-execution is the simultaneous execution of both HW and SW components, on the various CPUs, MCUs, DSPs, ASICs, and FPGAs in the system
  - Watch out for co-verification!
  - It is when we verify our SW on a model of our target HW (e.g. Mentor Graphics Seamless tool [CO-1])

Co-design Generic Flow [CO-2]



Co-design Particular Flow [CO-3]



Co-design Environments (1)

Language/Environment	Description
Handel-C [CO-4]	Programming language designed for compiling programs into hardware images of FPGAs. Subset of C, extended with a few constructs for configuring and generating the HW
Single-assignment C (SA-C) [CO-5]	Variant of C that can be directly and intuitively mapped onto circuits, including FPGAs
SystemC [CO-6]	C++ class library that provides necessary constructs to model system architectures, including HW timing, concurrency and reactive behaviors
POLIS [CO-7]	Based on CFSMs and includes translating the application from formal languages, simulating the app. behavior, partitioning the system, and obtaining a physical prototype
ImpulseC (Streams-C) [CO-8]	Allows for the description of computational processes, and their connections, and their parallel realization on FPGAs and µPs/DSPs
HardwareC [CO-9]	C-like syntax extended with concurrent processes, message passing, timing and resource constraints and template models
psC [CO-10]	Parallel-C language that provides a high-level abstraction for RTL parallel-synchronous execution

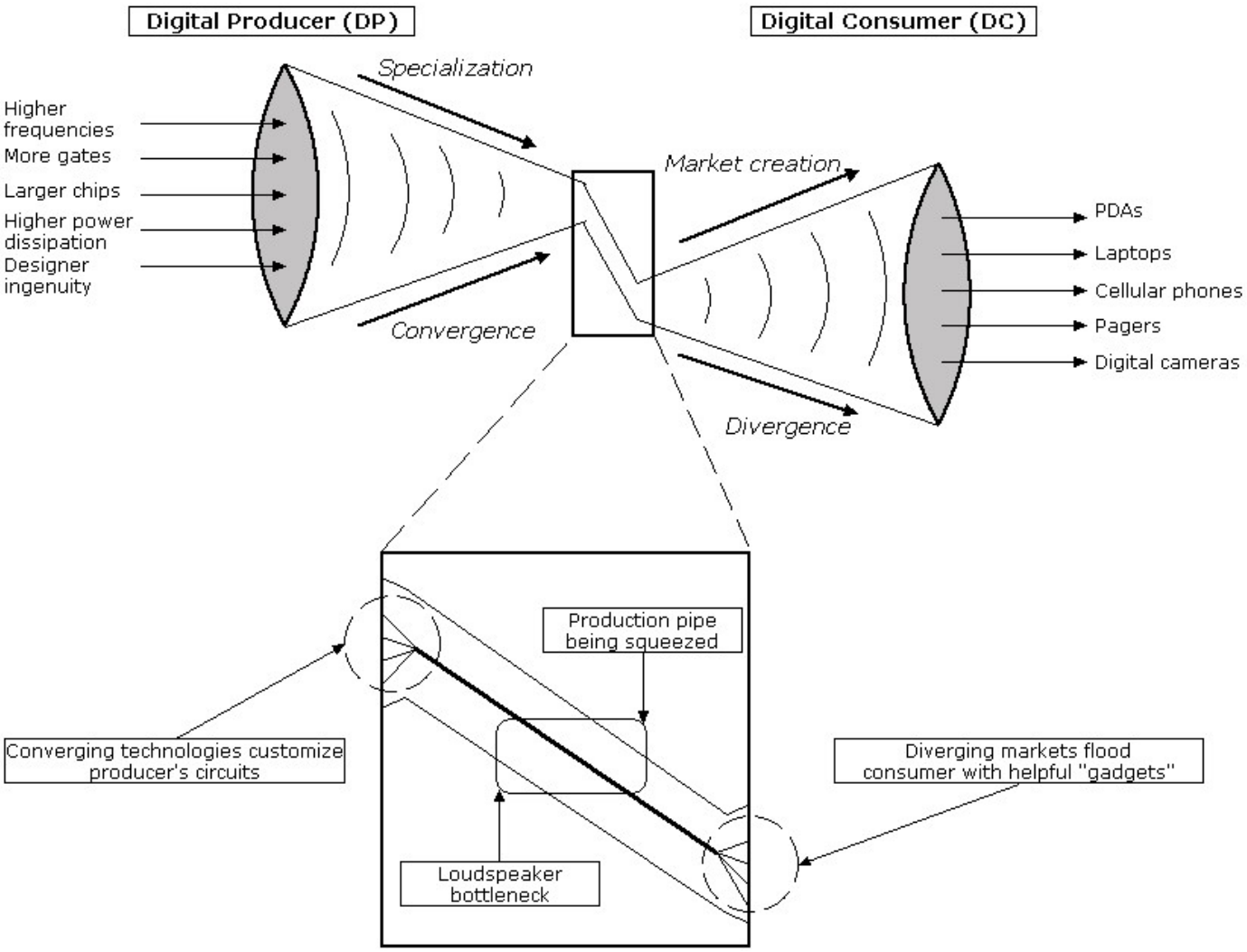
Co-design Environments (2)

- Other true HW/SW co-design environments
  - Cosyma, Vulcan, Lycos, Castle, SpecSyn, Cosmos
- Other co-design languages
  - SpecC, PureC/C++
- Timed vs. untimed C
  - If the programmer is able to explicitly specify the clock boundaries of a C expression → timed C
  - If the programmer cannot map the execution of a C expression to a specific clock event → untimed C
  - Timed C involves more work, but can precisely control the hardware
  - Untimed C is easier to work with, however, if hardware control is required, HDL programming may be involved

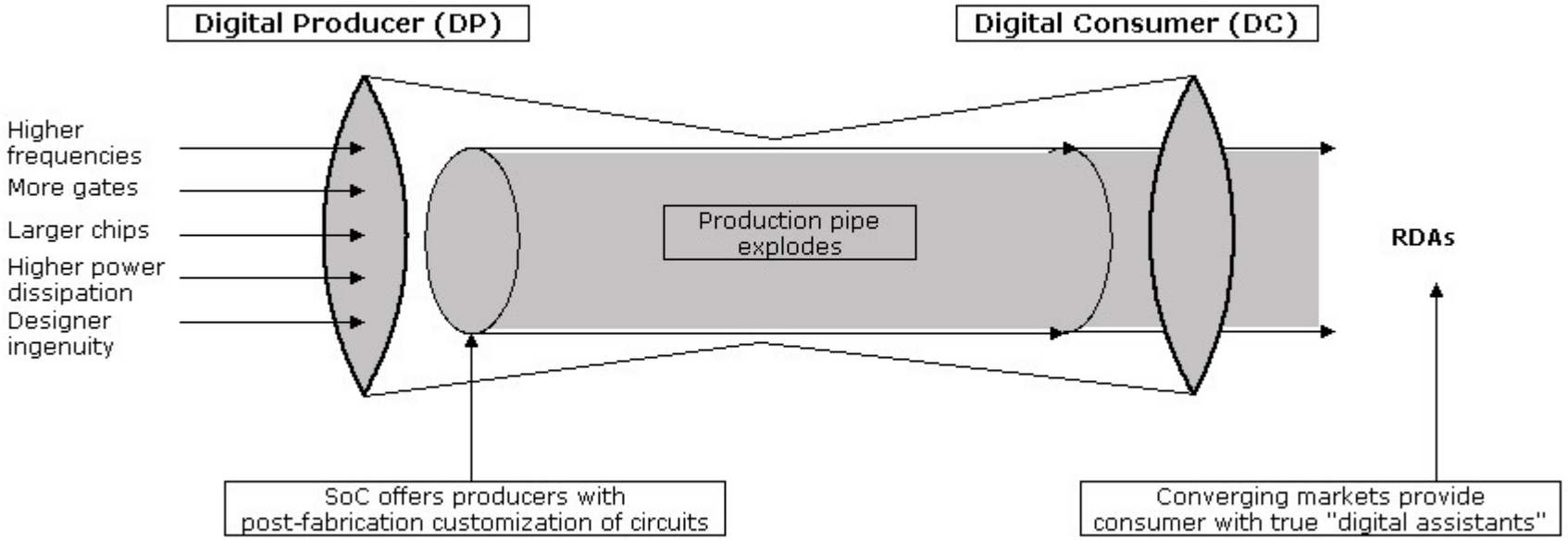
The SPLASH Effect (1) [ES-4]

- A problem has occurred on our way to deep sub-micron levels:
  - Schism between the traditional ASIC tools and their required outcomes
    - Designers are becoming too specialized in one set of CAD tools
  - Plethora of gates and not enough designers to use them
    - Hardware design is now to complex and customized
  - Lack of a technology to which Moore's Law can be extrapolated to
    - Self-prophesizing law "The length of eternity is 18 months, the length of a product cycle" [ES-5]
  - Augmentation of the cost of fabrication plants
    - US $ 14 million in 1966
    - US $ 3 billion in 1998
    - US $ 10 billion in 2005
  - Separation between the digital producer and the digital consumer
    - Demonstrated by the loudspeaker bottleneck (next slide)
  - Hindrance of the optical lithographical process caused by physical limitations
    - Physics could signal the end of Moore's Law (but not yet!)

The SPLASH Effect (2) [ES-4]



The SPLASH Effect (3) [ES-4]

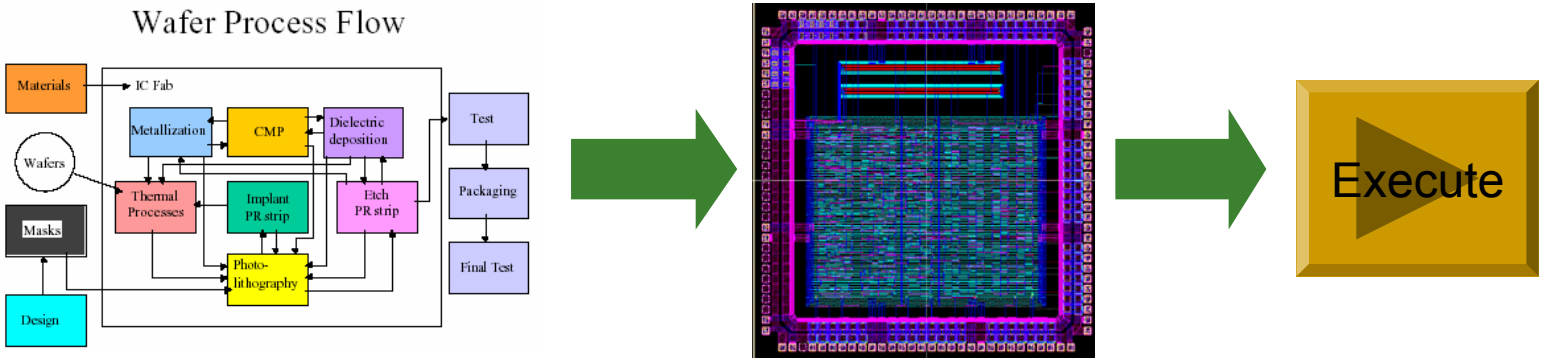


- An RDA is a reconfigurable digital assistant
  - It can be reconfigured to tailor to an intended application
  - It can build the required resources, on demand and in real-time
  - No more customization and no more market utilization gap!

Reconfigurable Computing Overview

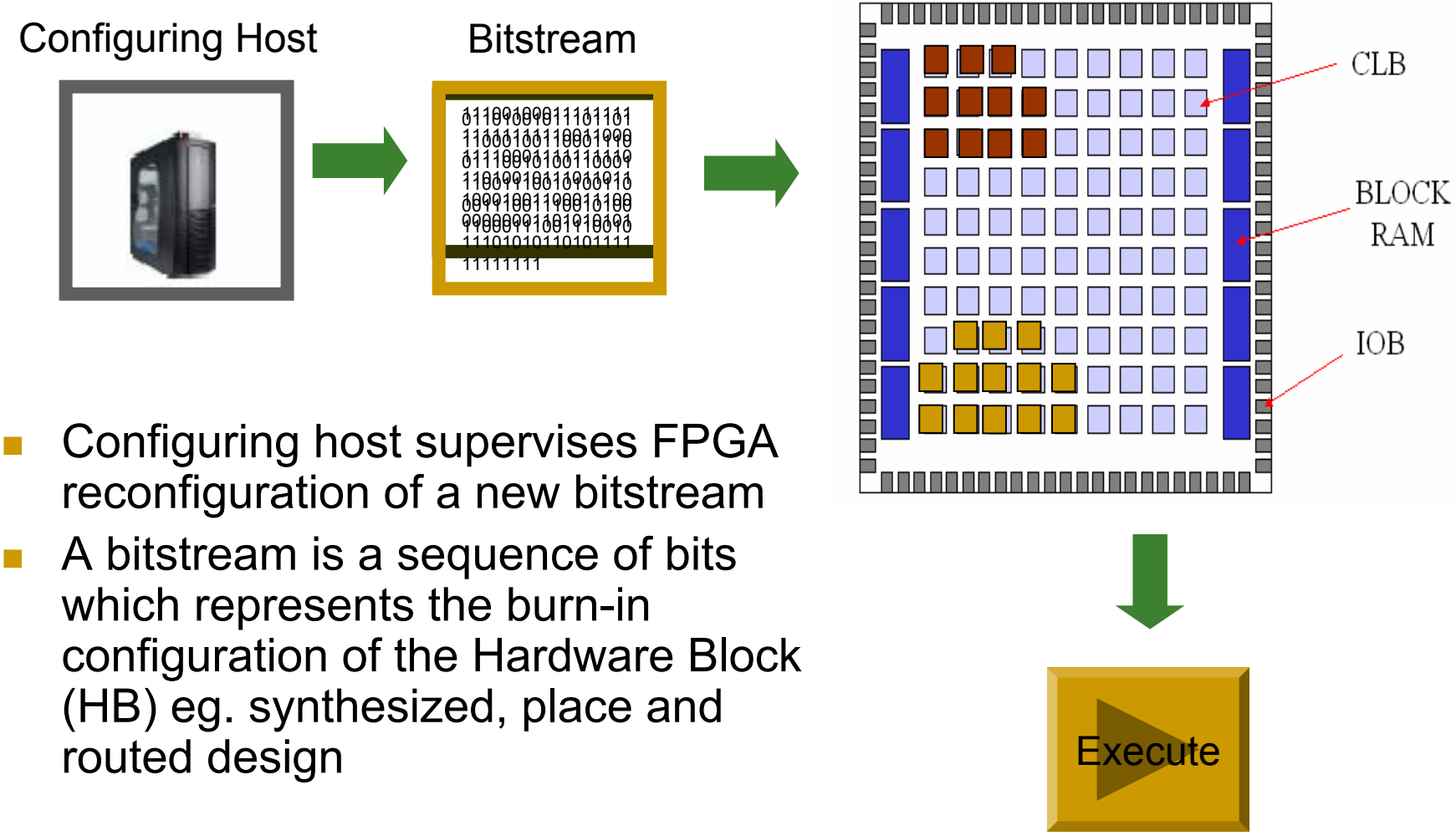
- Research began in the late 1980s but didn't take off until the FPGA became viable
- RC fills the gap between hardware and software
  - It performs much higher than software
  - It is much more flexible than hardware
- Let us begin with a simple classification
  - Non-configurable computing
  - Configurable computing
  - Reconfigurable computing
- Each has its own set of advantages, disadvantages and applications

Non-configurable Computing



- Uses fixed hardware such as ASICs or custom VLSI circuits (e.g. microprocessors like x86, Sparc, DEC, PowerPC, etc…)
- Long product turnaround time, usually around 3-6 months
- Optimized performance
- Can be quite costly
- Hardwired, thus, no room for error, re-work or improvement

Configurable Computing (1)



- Configuring host supervises FPGA reconfiguration of a new bitstream
- A bitstream is a sequence of bits which represents the burn-in configuration of the Hardware Block (HB) eg. synthesized, place and routed design

Configurable Computing (2)

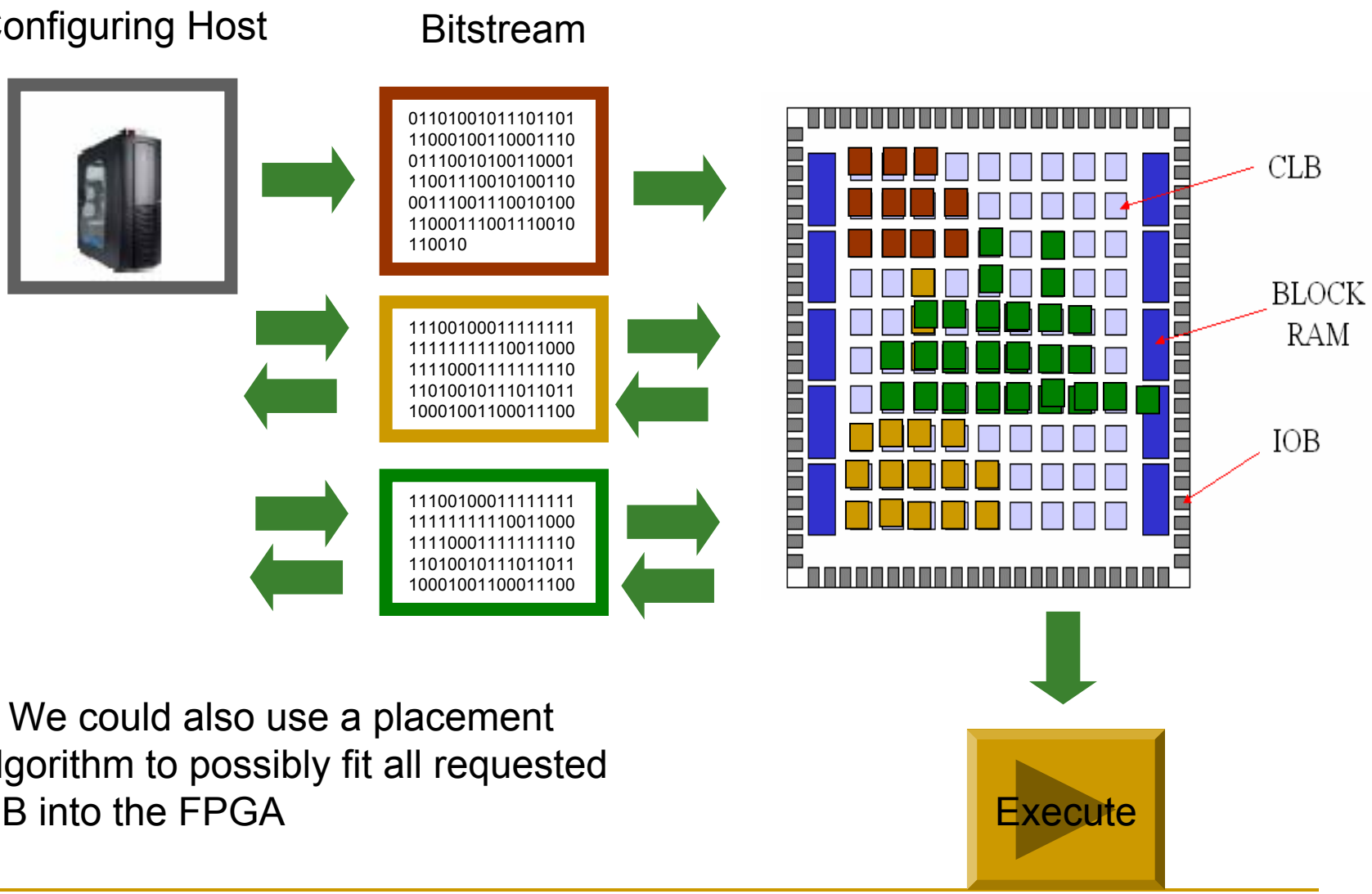
Advantages:

- Uses configurable hardware such as FPGAs or CPLDs
- PLDs are soft wired for reuse of static hardware resources
- Cost effective
- Quick turnaround time
- Flexible and ease in design process

Disadvantages:

- Inefficient use of hardware resources, cannot use unused idle FPGA area during run-time
- Slow reconfiguration time, because of reconfiguring the entire FPGA for a single Hardware Block (HB)
- Thus, must stop execution while reconfiguring a new Hardware Block

Reconfigurable Computing (1)



- We could also use a placement algorithm to possibly fit all requested HB into the FPGA

Reconfigurable Computing (2)

Advantages:

- Same as Configurable Computing
- No need to completely stop the execution while reconfiguring the FPGA with a new HB
- Efficient use of static hardware resources; can swap out or move HBs around to fit new HBs on the FPGA, no need for a larger FPGA or a second one
- Fast reconfiguration times, (with a **Xilinx Virtex** FPGA, reconfiguration times can be **less then 1 ms** for reconfiguring the entire FPGA)
- Run-time reconfiguration on the fly
- Less power consumption, as we can swap out HBs

Disadvantages:

- Routing HBs can be a heavy overhead for the configuring host especially if HBs are too large or when defragmentation is necessary

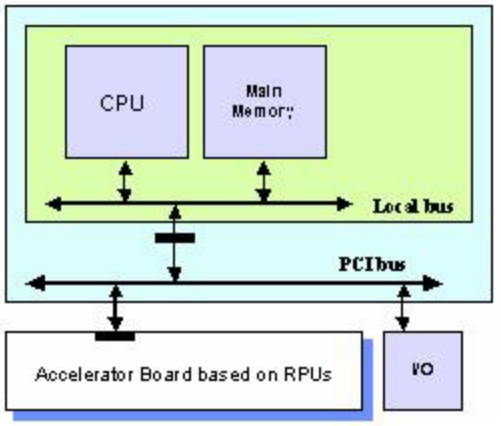
Reconfigurable Computing (3)

- A RC system must contain the following **features**:
  - A reconfigurable architecture (RA)
  - A run-time environment (RTE)
  - Run-time reconfigurability (RTR)
- A RC system must adhere to the following **requirements**:
  1. Dynamic reprogrammability of the device
     - No down time
     - Functionality gained must offset reconfigurability times
  2. Partial reconfiguration of the device
     - Necessary blocks are swapped online and in real-time
  3. Accessible and visible internal state
     - Eases task switching, scheduling and allocation
  4. Embedded processor presence
     - Required to handle context-switching, hardware routing, pre-emptive scheduling and so on.

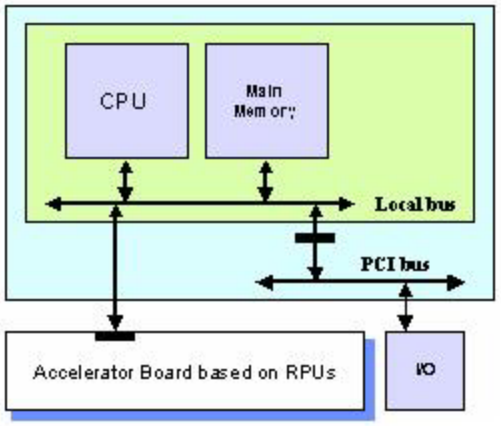
Reconfigurable Architectures (1)

- A RA is a PLD-based design system, coupled with a microprocessor used to combine the strengths of both hardware and software
- RAs can be classified in four different architectures, based on coupling strategies:
  a) Working as an *external processor* coupled through the *I/O bus*;
  b) Working as an *attached processor* coupled through the *local bus*;
  c) Working as a *coprocessor* directly coupled to the *main processor*; and
  d) Working as a *functional unit* coupled through the *datapath* of the main processor

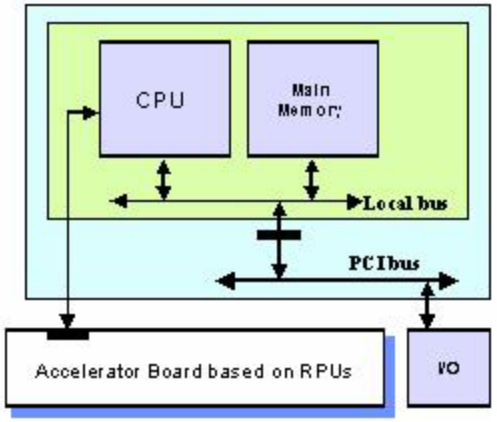
Reconfigurable Architectures (2) [RC-1]



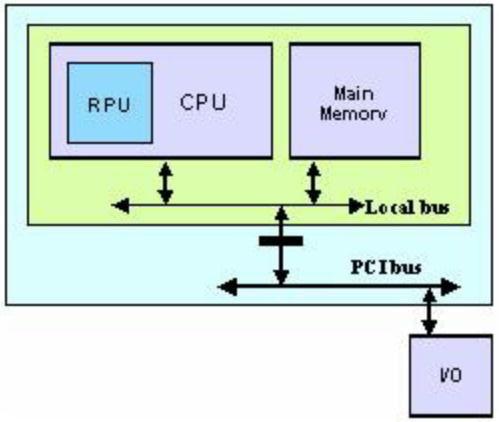
a) RPU coupled to the I/O system bus



b) RPU coupled to the local bus



c) RPU coupled to the CPU



d) RPU integrated in the process chip

Reconfigurable Architectures (3)

- Can also be classified as coarse- or fine-grained
  - **Coarse-grained architectures**: the minimal path width is at least greater than one
  - **Fine-grained architectures**: usually 1-bit path widths
- There are numerous examples of RAs (much more at [RC-2])
  - CHESS Array (1999)
    - Floorplan is chessboard-like
    - Interleaved ALUs and switchboxes (logical architecture)
    - 16 buses in each row and column (interconnect architecture)
    - 4-bit, multi-granular (granularity)
    - JHDL compilation (mapping)
    - Mesh based (structure)
  - XD1 (2004)
    - AMD Opteron 64-bit processors along with 6 Xilinx V2P FPGAs
    - RapidArray provides high-speed, low-latency paths (interconnect)
    - 1-bit, multi-granular (granularity)
    - Verilog, VHDL, Handel-C, Matlab/Simulink, Impulse-C (mapping)
    - Mesh-based (structure)

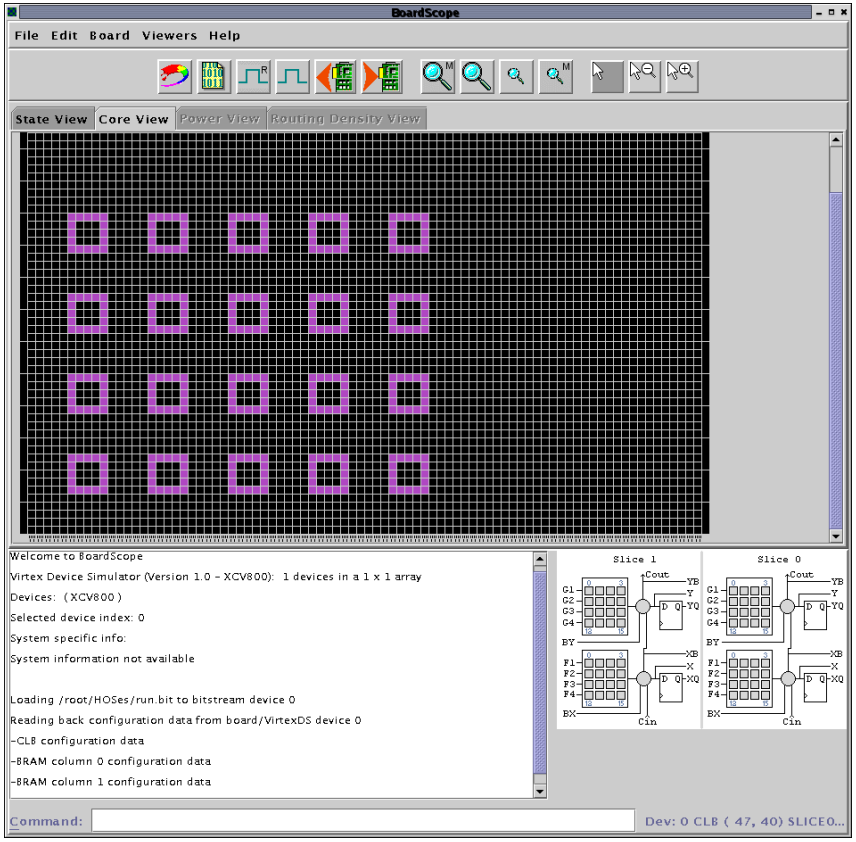
Run-Time Environment

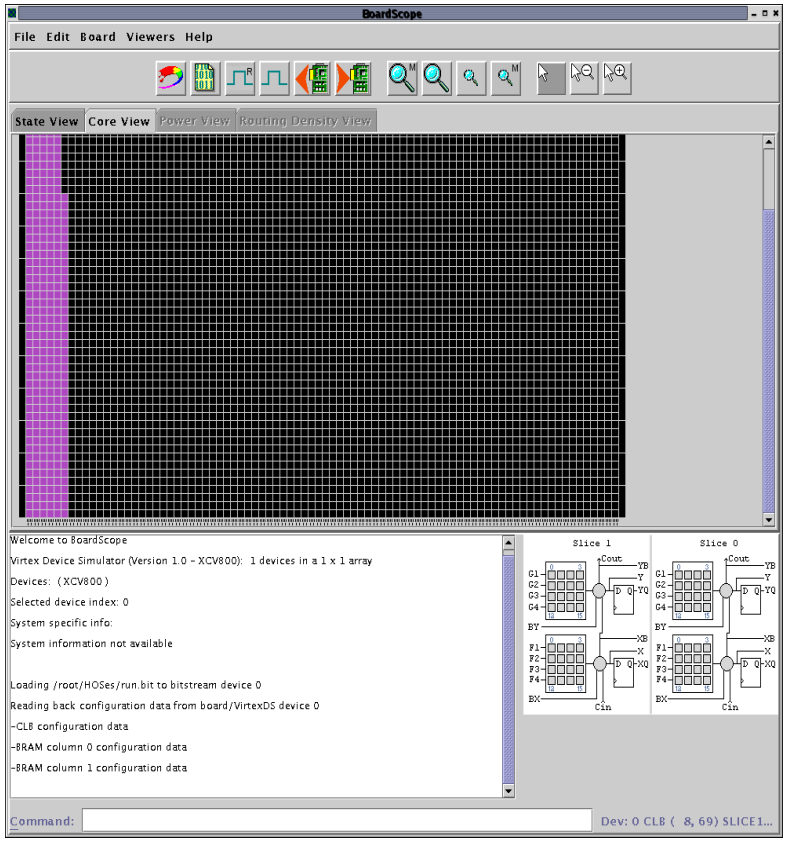
- A RTE is required to manage resources in an abstract manner
  - Models must be created to *virtualize the resources*
  - The more concrete the model is, the **more** a designer knows about the architecture, however
  - The more abstract a complex architecture is to a designer, the **easier** it is for the designer to create applications
- Needed to promote RC design to a much wider pool of designers
- Will aid in the transition to application designers
- Best if residing within the OS (see hardware operating systems in [RC-3])

Run-Time Reconfiguration (1)

- RTR involves the direct manipulation of the available hardware resources at run-time, in order to respond to the surrounding requirements placed on the system
- Time-sharing of different tasks (temporal partitioning) allows for:
  - Minimizes the required silicon area
  - Introduces the virtual hardware concept
  - Cycle-by-cycle context switching
  - Post-fabrication adaptation to new standards/features
  - Acceleration of the application through H/W hot-spot cores
  - True multitasking of applications and algorithms

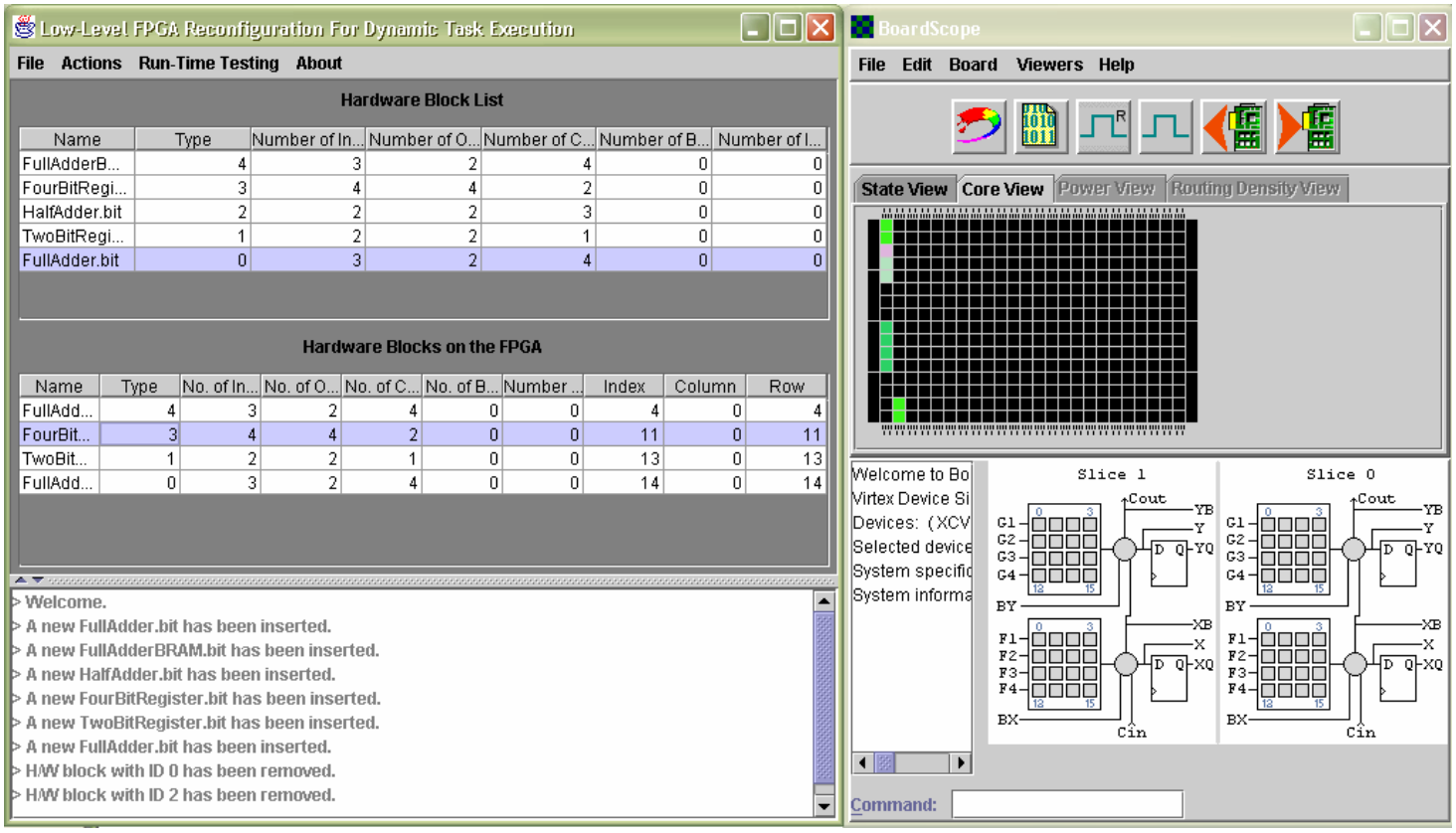
Run-Time Reconfiguration (2)





- Decided on reconfiguring all HBs into columnar-blocks
- The Virtex FPGA's atomic unit of reconfiguration is the column

Run-Time Reconfiguration (3) [RC-3]



- Inserting and removing HBs at run-time
- Implemented FPGA defragmentation to “fill the holes” created by the application flow

The SPLASH Effect (revisited)

- RC resolves the SPLASH effect by
  - Schism…
    - A new and innovative set of tools and compilers have been developed
    - Allows for application designers, rather than software or hardware designers
  - Plethora of gates…
    - A larger designer pool is targeted
    - The computer did not truly flourish world-wide until software and the Internet allowed the free exchange of ideas and applications amongst its consumers
  - Lack of a technology…
    - Nano-technology, DNA computing, chaos-based computing, quantum computing, Xputers
  - Augmentation of the cost…
    - RC dramatically lowers fabrication costs!
  - Separation between the DP and DC…
    - There is no need for post-fabrication customization in order to bridge the DP-DC gap
  - Hindrance of the optical process…
    - RC opens up various fields where the hardware is shared in both space and time!

The Alternatives

- A few alternatives to RC design exist, including:
  - General-purpose microprocessors (μPs)
  - Digital signal processors (DSPs)
  - Application-specific integrated circuits (ASICs)
  - System-on-chip (SoC) designs
- We will next explore these alternatives focusing on their performance measures

Performance Measures (1)

- Generic performance equation
  - *Performance = [frequency * **IPS**] / number of instructions*
- Basic performance equation (processor time)
  - *T = [N * S] / R*, where
    - T is the processor time
    - N is the number of instructions in the program
    - S is the average number of basic steps needed to execute one machine instruction **(CPI**)
    - R is the clock rate (processor speed)
- The goal is to decrease processor time and to increase performance. How is that attained ?
  - Increase clock rate (or frequency) → **Controlled by IC processes**
  - Modify the instruction set → **CISC vs. RISC**
    - Interesting point!
      - CISC → ↓N but ↑S
      - RISC → ↑N but ↓S
  - Increase the number of instructions per second (IPS) → **VLIW**
  - Increase the efficiency of the compiler ↓[N * S] → **Borland, Microsoft, gcc**

Performance Measures (2)

- SPEC
  - System Performance Evaluation Corporation
  - Publishes suites of programs for each application to be tested by the computer
  - Results are referenced to a well-known computer
    - For SPEC1995, it was the SUN SPARCstation 10/40
    - For SPEC2000, it was the Ultra-SPARC-10 workstation with a 300-MHz UltraSPARC-Iii processor
- SPEC rating = [Running time on the reference computer] / [Running time on the computer under test]
  - E.g. SPEC rating of 50 means CUT is 50 times as fast as the reference computer
  - Watch out!
    - SPEC ratings measure the combined effect of all factors affecting performance, including the compiler, the OS, the processor and the memory

µP/DSP Co-design

- The microprocessor introduced a new computing paradigm
  - Instead of designers having to map a fixed problem onto fixed resources (e.g. TTL design), they were mapping variable problems onto fixed resources
- This created a big boom in computing
- Two major bottlenecks exist today
  - Instruction execution
    - Each instruction is fetched, decoded and executed
    - Complexity is always increasing
  - Computational efficiency
    - *Performance = [frequency * **IPS**] / number of instructions*
    - *Power = 0.5 * capacitance * [voltage]^2 * frequency*
    - Thus, increasing the performance increases the power!
    - However, embedded systems are high-computation, low-power devices!

ASIC Co-design

- Powerful customized chips operating at very high speeds, and consuming less power than typical µPs/DSPs
- However, they are neither reconfigurable nor flexible
- They are very expensive to design and produce
- They require very specialized designers
- They have a very long time-to-market
  - Small changes in a design might cost the product cycle months at a time! (for a satirical view of the “man-month” refer to [RC-4])
- Changes in the applications environment and the ability to support dynamic standards rule out ASICs over the long run

SoC Co-design

- As soon as the process technologies entered the deep sub-micron levels
  - Grouping various cores together became viable
  - Controlled the size and complexity
  - Offered off-the-shelf functionalities
- The mixing and matching of such cores is what is termed a System-on-Chip design
- Many challenges face this infant
  - Interfacing the cores together
  - Verifying the cores' individual and combined functionalities
  - Building large IP databases
  - Managing all the licensing and legal issues involved
- SoCs have a future, but it is not their time just yet!

Potential Applications

- **Adaptive embedded systems** would be capable of arithmetic, DSP, multimedia, and other computationally intensive functions
  - Low-power requirements met by swapping out idle HBs and clocking only operational ones
  - Adaptation requirements met by updating HBs on-the-fly and allowing for the *downloa*d of Internet HBs!
- Current boom in **mobile robotics** will result in the adoption of RPUs for the management of real-time and low-power tasks
- The **computer of the future** will immensely benefit from the addition of a RPU, to complement the extremely fast and efficient, yet inflexible, contemporary processor

References (1)

Embedded Systems (ES)

1. M.J. Flynn, “Very High-Speed Computing Systems,” *Proceedings of the IEEE*, vol. 54, p.p. 1901-1909, December 1966
2. Petrov Group, “Trends in HW/SW Co-Design – User and Vendor Strategies,” Strategic Report, April 2005
3. Nick Tredennick, “Get ready for reconfigurable computing,” *Computer Design*, April 1998
4. Groza V., Abielmona R., Petriu E., “Reconfigurable Computing: Exploring Emerging Technologies,” *INES 2002, IEEE International Conference on Intelligent Engineering Systems*, ISBN 953-6071-17-7, pp. 539-544, Opatija, Croatia, 25-28 May, 2002
5. Robert Schaller, “Moore’s law: past, present and future,” *IEEE Spectrum, pp. 53-59*, June 1997

Reconfigurable Computing (RC)

1. Joao Cardoso and Mario Vestias, “Architectures and compilers to support reconfigurable computing,” *http://www.acm.org/crossroads/xrds5-3/rcconcept.html*, last viewed on 08/25/2006
2. Rami Abielmona, “Alphabetical List of Reconfigurable Computing Architectures,” *http://www.site.uottawa.ca/~rabielmo/personal/rc.html*, last viewed on 01/16/2006
3. R. Abielmona, V. Groza, N. Sakr, "Low-Level Run-Time Reconfiguration of FPGAs for Dynamic Environments," *IEEE Canadian Conference on Electrical and Computer Engineering, CCECE 2004*, pp. 2135 – 2138, Vol.4, May 3-5, 2004 Niagara Falls, Ontario, Canada
4. Frederick Brooks, “The Mythical Man-Month,” Addison-Wesley, 1995

References (2)

Co-design (CO)

1. Mentor Graphics Corp., “HW/SW Co-Verification and Performance Analysis,” Product datasheet, 2005
2. Rozenblit, J. and K. Buchenrieder (editors). Codesign Computer -Aided Software/Hardware Engineering, IEEE Press, Piscataway, NJ, 1994; © IEEE 1994
3. Canadian Microelectronics Corporation (CMC), “Digital Design Flow,” http://www.cmc.ca, last viewed on Jan. 2002
4. T. Stocklein and J. Basig, “Handel-C: an effective method for designing FPGAs (and ASICs),” Fachbereich Nachrichten-und Feinwerktechnik.
5. Sven-Bodo Scholz, “Single Assignment C: Functional Programming Using Imperative Style,” *Proceedings of the 6th International Workshop on Implementation of Functional Languages (IFL'94),* Norwich, England, UK, pp.21.1-21.13, University of East Anglia, 1994
6. Bhasker, J. “A SystemC Primer”, Star Galaxy Publishing, Allentown, PA: June 2002
7. F. Balarin, et al., “Hardware-Software Co-Design of Embedded Systems: The Polis Approach,” Kluwer Academic Press, 1997
8. David Pellerin and Scott Thibault, “Practical FPGA Programming in C,” Prentice Hall, 2005.
9. David Ku and Giovanni DeMicheli, “HardwareC – A Language for Hardware Design,” Technical Report CSL-TR-90-419, Stanford University, 1990
10. Novakod Technologies, “psC Technology,” company white paper, http://www.novakod.com/docs/Novakod_psC_Technology_Brochure.pdf, 2005