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
Figure 1  Concept View of Embedded (Computing) System Design

Traditional  
Non-embedded  Computing Centric

specification(1)
Hardware Design  Software Design
Hardware Design Environment  Software Design Environment
Actual Prototype

Emerging  
Embedded  Computing Centric

specification(1)
Early Binding of HW and SW
Hardware Design  Software Design
Virtual Prototype
"Co-design Assisted"
Actual Prototype

specification(2)
Late Binding of HW and SW
HW/SW Co-design
Hardware Design  Software Design
Virtual Prototype
Actual Prototype

(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)

© Copyright 2005 by the Petrov Group. All rights reserved.
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!

Embedded systems market breakdown
- Zero-delay – printers, copiers, scanners
- Zero-power – cellulares, pagers, watches, cameras
- Zero-cost – blenders, TVs, radios
- Zero-volume – military, supercomputers
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 Particular Flow [CO-3]
### Co-design Environments (1)

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

Configuring Host

Bitstream

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

Execute
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 \textit{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
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
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
  - \( Performance = \frac{\text{frequency} \times \text{IPS}}{\text{number of instructions}} \)

- Basic performance equation (processor time)
  - \( T = \frac{N \times S}{R} \), 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 ↓\([N \times S]\) → **Borland, Microsoft, gcc**
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-ii processor

**SPEC rating** = \([\text{Running time on the reference computer}] / [\text{Running time on the computer under test}]\)

- 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
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 = \( \frac{\text{frequency} \times \text{IPS}}{\text{number of instructions}} \)
  
  Power = 0.5 \times \text{capacitance} \times \text{[voltage]}^2 \times \text{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
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 *download* 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)


Reconfigurable Computing (RC)

References (2)

Co-design (CO)