Introduction to dedicated systems codesign

Lecture 01 on Dedicated systems

Teacher: Giuseppe Scollo

University of Catania
Department of Mathematics and Computer Science
Graduate Course in Computer Science, 2016-17

Table of Contents

Motivation and fundamental concepts of:

- modelling, design and optimal implementation of application-specific information processing systems
- use of hardware and software tools, such as development and cosimulation platforms, to design and to implement application-specific systems

A "traditional" definition:

> the design of cooperating hardware components and software components in a single design effort
> (Schaumont, p. 11)

A "nontraditional" definition:

> the partitioning and design of an application in terms of fixed and flexible components
> (Schaumont, p. 12)

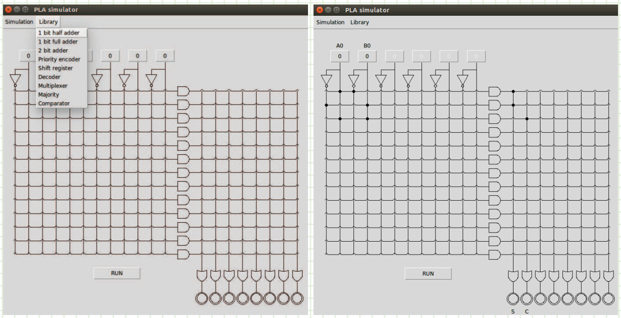
What's the difference?

- application rather than components design
- partitioning the application is a design activity
- components: hardware → fixed, software → flexible

Why the difference? See next...

Hardware flexibility, to various extent:

- prototypical case: PLA—*see the PLA simulator by Alice Plebe:*



- today's practical case: FPGA

  the "program" is a user-specified netlist of logical elements and connections between them

- soft hardware: a *soft-core* is a processor implemented in the netlist of an FPGA

The two terms are *not* synonyms:

- A *dedicated* system is designed, in all of its aspects, to implement a specific application
- An *embedded* system is a dedicated system that implements an application in the context of, and interacting with, a wider physical system

  the two together form a *cyber-physical system*

Embedded systems: wide variety, fast-growing markets:

automotive, avionics, traffic control, mobile telephony, digital cameras, television, domotics, robotics ...

Dedicated systems are also components or subsystems of *general-purpose* information processing systems:

arithmetic coprocessors, cryptographic coprocessors, videographic cards, A/V codecs, DMA I/O controllers, GPU subsystems ...

Technological factors tip the balance in favour of more hardware:

- **Computational performance**

  work done per time unit, or clock cycle: hardware parallelism as well as dedicated hardware accelerators yield increased computational performance

- **Energy efficiency**

  may vary over several orders of magnitude, for example (Schaumont, p. 14):

Energy efficiency of AES encryption implementations					
Gb/J:	10^{-6}	10^{-3}	10^{-2}	10^{0}	10^{1}
platform:	Java KVM Sparc	C Sparc	Asm Pentium-III	Virtex-II FPGA	0.18µm CMOS ASIC

- **Power density**

  computational performance improvement by clock frequency rise is limited by the directly proportional rise of power dissipation, hence by cost-effectiveness of cooling technology → parallel architectures

Best-match for HW/SW codesign: parallel computing platforms

- shared-memory multiprocessors, FPGA accelerators, GPU's, multi-core architectures ...

  an example of energy-efficient, open HW/SW platform: the Parallella board

Economical factors tip the balance in favour of more software:

- **Design cost**

  Chip design is a very expensive effort, with high NRE cost;
  reprogrammable chips, which allow reuse through reprogramming, spread the chip design cost through multiple products or product versions;
  reprogrammability may take many different forms, though

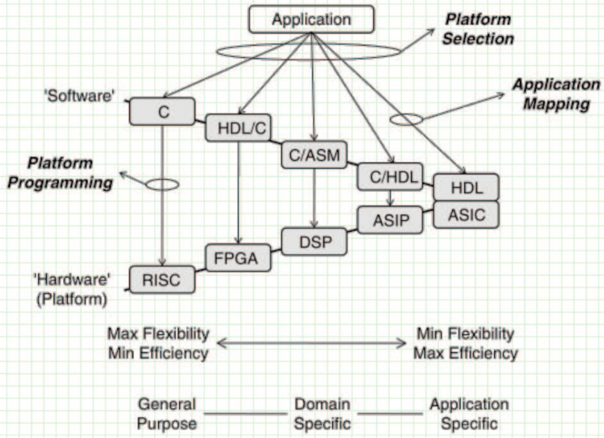
- **Development time**

  Not only design cost but also development time of a new chip is fairly high;
  on the other hand, low time to market enables timely entry into the market window;
  this yields higher revenues, which is especially significant for innovative products

- **Design complexity**

  fixed hardware means fixed design decisions;
  the flexibility of software enables designers:
  – to develop the application at a higher abstraction level, and
  – to maintain the application through the changes needed to resolve bugs or to cope with evolving requirements

The structured collection of all possible implementations of a given application



Schaumont, Fig. 1.7 - The hardware-software codesign space

Definable by the *time granularity* of elementary (atomic) actions

Starting at the lowest abstraction level:

- continuous signals

  models are systems of differential equations; useful for hybrid systems with analog components
  not used in practice to describe typical HW/SW systems

- discrete events

  signal level changes at irregularly spaced points in time—lowest abstraction for digital hardware

- clock cycles

  discrete events observed at regularly spaced time-intervals
  register-transfer level (RTL) models, useful for single-clock synchronous hardware

- machine instructions

  useful for simulation of complex software systems, where cycle-accurate simulation would be too expensive; instruction-accurate simulation may not reveal real time-performance, though

- transactions

  models expressed in terms of interactions between components of the system; useful when even instruction-accurate simulation would be too expensive, as well as in the early phases of a system design

Feature constructs for specification of (static) *structure* as well as of (dynamic) *behaviour*

The three most prominent ones, all with discrete event semantics:

- VHDL
  - *IEEE 1076 (revised) standard dates: 1987, 1993, 1999 (VHDL-AMS), 2006-2008*
  - *HW components are "entities" which comprise "processes"; these react to events at their input ports*
  - *a "synthesizable" subset of VHDL may be automatically compiled to an FPGA netlist*
- Verilog
  - *IEEE 1364 standard (version) dates: 1995, 2001, 2005, 2009 (SystemVerilog: IEEE 1800)*
  - *similar to VHDL, but built-in support for 4-valued logic, features for transistor-level description etc.*
- SystemC
  - *a C++ class library providing required functions for HW modeling*
  - *structured into: core language, data types, elementary channels, higher-level channels*

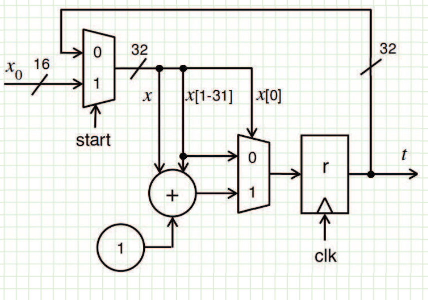
A more concise language, for RTL description of synchronous hardware:

- GEZEL
  - *cycle-based: no explicit modeling of clock events*
  - *FSMD (Finite State Machine with Datapath) models + library of processor instruction-set simulators*
  - *automated translation of "proper" FSMD models to synthesizable VHDL*

Collatz trajectories

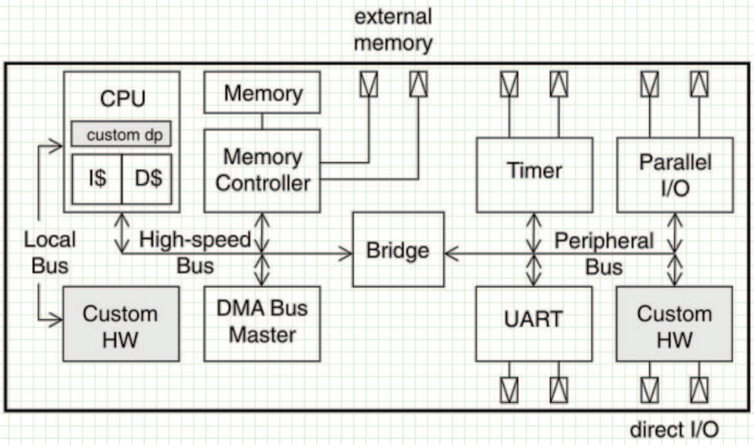
- for each positive integer x_0, the (infinite) sequence of outcomes of the iterated application, starting at x_0, of the function over the positive integers defined by: *f(x) = 3x+1 if x odd, f(x) = x/2 if x even*
- since 3x+1 is even when x is odd, consider a slightly compressed form of the trajectories, as is defined by iteration of the function: *t(x) = (3x+1)/2 if x odd, t(x) = x/2 if x even*
- Conjecture: for every positive integer x_0, the trajectory eventually falls into the small loop through 1
- here is a hardware datapath that produces the *t* trajectory (for 16-bit x_0), and its description in GEZEL

N.B. for odd x : $(3x+1)/2 = x + \lfloor x/2 \rfloor + 1$




```
dp collatz ( in start : ns(1) ; in x0 : ns(16) ;
             out t ns(32)) {
  reg r : ns(32) ;
  sig x : ns(32) ;
  always {
     t = r ;
     x = start ? x0 : r ;
     r = x[0] ? x + (x >> 1) + 1 : x >> 1 ;
  }
}
```

A generic SoC design template:



Schaumont, Fig. 8.1 - Generic template for a system-on-chip

Basic concepts:

- Synchronization

  time granularity: clock cycle, bus cycle, transaction
  data exchange: abstract, scalar, composite
  control: semaphores, handshake protocols, blocking vs nonblocking etc.

- Computational performance

  bottleneck analysis, say:
  — channel at *v* bits/transfer, *B* cycles/transfer
  — coprocessor at *w* bits/execution, *H* cycles/execution
  communication-constrained: $v/B < w/H$
  computation-constrained: $v/B > w/H$

- HW/SW coupling

  tight coupling: frequent interaction, fine granularity
  loose coupling: infrequent interaction, coarse granularity

Collections of HW and SW tools for codesign development and testing

FPGA development boards are the basic hardware tools to this purpose

they come equipped with sophisticated software systems for high-level codesign and cosimulation

for example, the DE1-SoC development board by Altera (see picture), which hosts a Cyclone V FPGA chip, with an ARM Cortex-A9 processor on the same chip, may include two NIOS II softcore processors on the FPGA, and is supported by the Quartus Prime Lite software, freely available

Altera DE1-SoC development board with Cyclone V FPGA
source: Altera University Program

Open-hardware platforms include: Parallella, Arduino, Cosino ...

see website references

Cosimulation may also be carried out on a software platform, with no FPGA involved

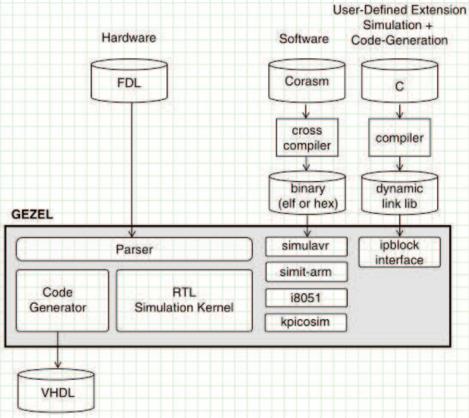
such a platform typically includes:

- cross-compilers and cross-assemblers for a given set of programming languages and microprocessor families, for the SW part of a codesign model
- HDL simulators, for the custom hardware part of the model
- cycle-accurate microprocessor instruction-set simulators
- software models of microprocessor hardware interfaces
- and possibly more ...

cycle-accurate cosimulation allows designers to estimate the performance of codesign solutions well before their actual implementation

A collection of Debian packages for Ubuntu installation (updated for every new LTS)

see website references



Schaumont, Fig. A.1 - Overview of the GEZEL tools

Reference textbooks

P.R. Schaumont: *A Practical Introduction to Hardware/Software Codesign*
2nd Edition, Springer (2012)

P. Wilson *Design Recipes for FPGAs: Using Verilog and VHDL*
2nd Edition. Newnes, Elsevier (2015)

Textbooks

P. Marwedel: *Embedded System Design: Embedded Systems Foundations of Cyber-Physical Systems*
2nd Edition. Springer (2011)

F. Vahid & T. Givargis: *Embedded System Design: A Unified Hardware/Software Introduction*
Wiley (2002)

C. Brandolese, W. Fornaciari: *Sistemi embedded: sviluppo hardware e software per sistemi dedicati*
Pearson, Milano (2007)

E.A. Lee & S.A. Seshia: *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*
2nd Edition, Version 2.0 (2015)

F. Vahid, T. Givargis & B. Miller: *Programming Embedded Systems: An Introduction to Time-Oriented Programming*
Version 4.0. Uniworld (2012)

M. Wolf: *Computers as components: Principles of embedded computing system design*
3rd Edition, Morgan Kaufmann (2012)

D. Ibrahim *PIC Microcontroller Projects in C*
2nd Edition. Newnes, Elsevier (2014)

Codesign and embedded systems courses

Hardware/Software Codesign, Patrick Schaumont, VirginiaTech
www.faculty.ece.vt.edu/schaum/teaching/4530

Hardware/Software Codesign with FPGAs, Jim Plusquellic, U. of New Mexico
ece-research.unm.edu/jimp/codesign

Cyber-physical system fundamentals, P. Marwedel, TU Dortmund
ls12-www.cs.tu-dortmund.de/daes/en/lehre/courses/sommersemester-2015/ss15-cyber-physical-system-fundamentals/slides-cpsf-ss-2015.html

Introduction to Embedded Systems, Edward A. Lee and Sanjit A. Seshia, U. of Berkeley
chess.eecs.berkeley.edu/eecs149

Free online course on Embedded Systems, EE Herald, Bangalore
eeherald.com/section/design-guide/esmod.html

Codesign platforms and tools

GEZEL: rijndael.ece.vt.edu/gezel2

Altera (University Program): www.altera.com/support/training/university/overview.html

Xilinx: www.xilinx.com

CUDA: developer.nvidia.com/cuda-zone

Parallella: www.parallella.org

Arduino: www.arduino.cc

Cosino: www.cosino.io