Celoxica

>: cutting a LONG story SHORT //

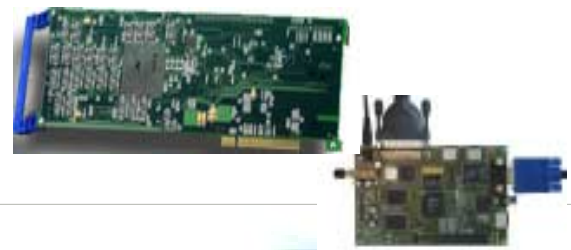
Celoxica announces DK1.1 for rapid hardware design

Celoxica Limited

- A supplier of high-level design tools that bridge the gap between the worlds of ESD (Embedded System Design) and EDA (Electronic Design Automation)
- Bringing new efficiencies to hardware design
- At the same time, we enable software engineers to accelerate software in parallel hardware to overcome CPU bottlenecks

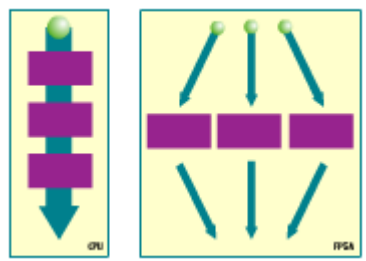
Celoxica's products and services

DK1.1 featuring Handel-C, the world's first design suite to build custom hardware from a high-level software language



RC range of development boards for rapid prototyping

Design services for proof of concept work and applications engineering



Hardware libraries (Rijndael, voice Codec, SDRAM controller)

PAL and DSM for rapid design, design reuse and co-processor acceleration

Celoxica
>: cutting a LONG story SHORT //

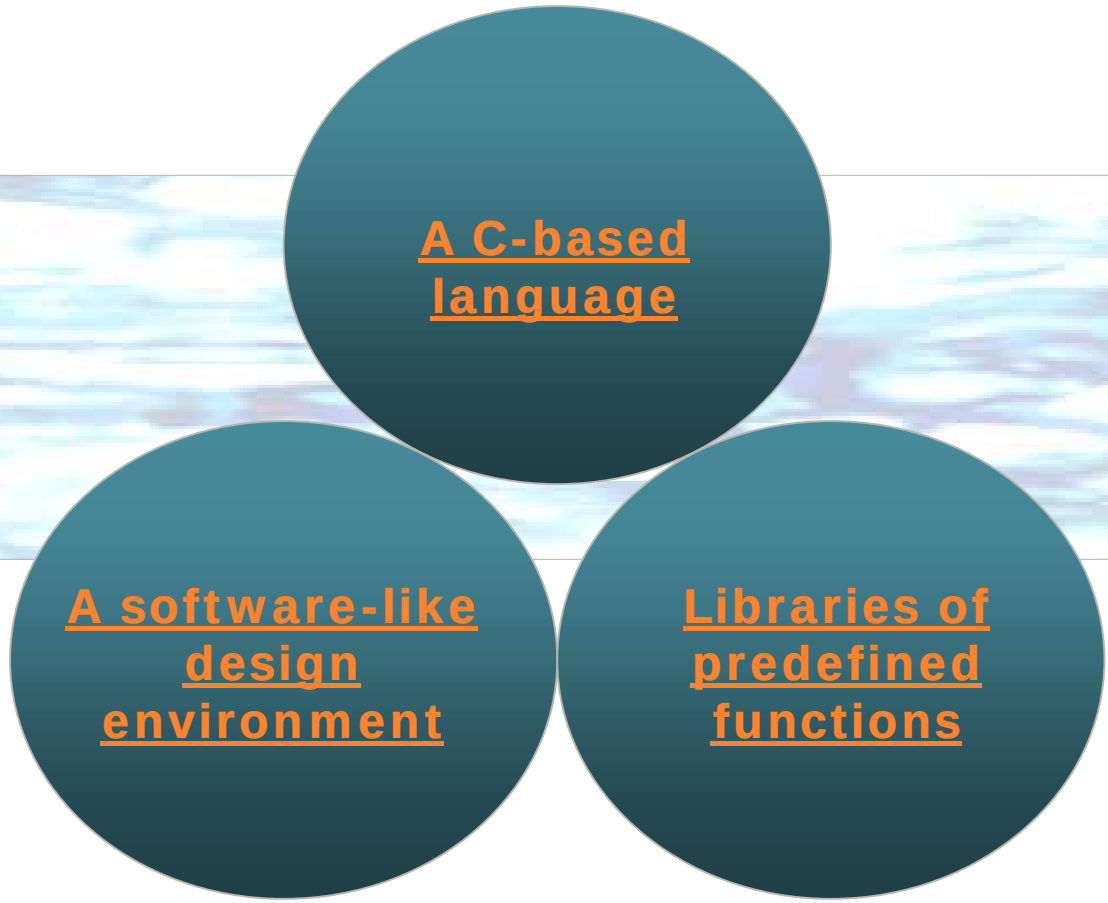
FPGA evolution

- FPGA/PLD technology has evolved rapidly
  - *Glue logic*
  - *Alternative to ASIC*
  - *System or sub-system platform*
- FPGA/PLD technologies now offer chips that integrate
  - *Soft or hard processors*
  - *Memory, multipliers and advanced I/O functionality*
  - *Millions of programmable gates*

Implications

- The hardware design objects are no longer circuits but systems or sub-systems
- Designers are not necessarily pure hardware designers but system designers, hardware and/or software designers working in cooperation
- Similar design methods for hardware and software can simplify optimal allocation of functionality to software, hardware and reconfigurable hardware
- The close integration between processors and hardware logic opens up new opportunities to accelerate software functionality

A software-like design approach



A C-based language: Handel-C

- Almost all of ANSI-C plus syntactic extensions for hardware
  - *Describes the behaviour rather than the structure*
  - *Software code compiled to hardware has same behaviour*
- Optimise code incrementally with full control
  - *Introduce parallel execution with simple **par** construct*
  - *Set key variable widths and deduce the rest automatically*
  - *Control resource sharing by in-line functions or arrays of functions*
  - *All constructs translate directly to hardware*
- Restructure code to gain more optimisation
  - *Concise readable code for better exploration of a large design space*

Key hardware adaptations

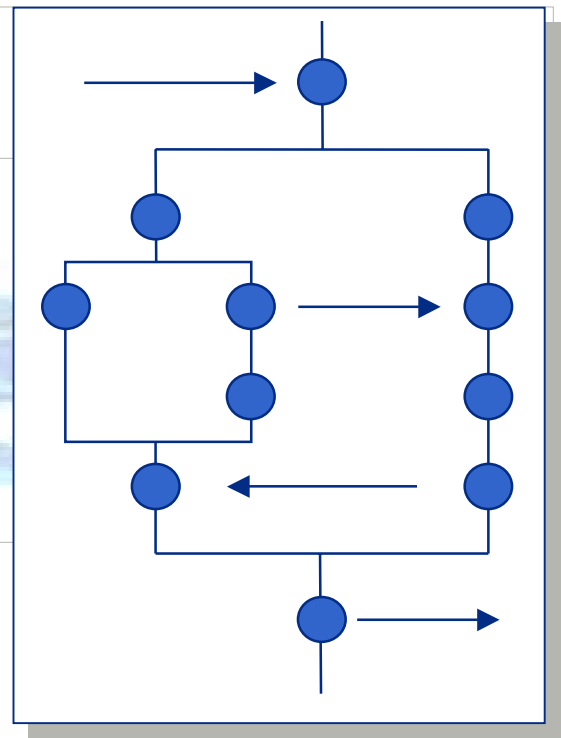
- Assignment takes one clock-cycle
- Serial execution is default – Parallel is declared
- Declare and infer width of variables

```
int a;
int b, tmp;
{
  tmp = a; // first  clock-cycle
  a = b;   // second clock-cycle
  b = tmp; // third  clock-cycle
}
```

```
int 5 a;
int b;
par {
  a = b;   // first clock-cycle
  b = a;   // same  clock-cycle
}
```

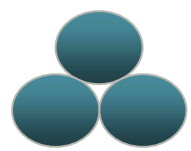
Pars and channels

- With nested ***par*** and ***seq*** blocks any serial/parallel execution graphs can be built
- No finite state machines are needed to control the flow
- Parallel treads can communicate via ***channels***
  - *synchronised transfer of values*
    between threads
    between clock-domains
  - *I/O*
    files (in simulation)



Benefits

- A **programming language** for hardware rather than a **hardware description language**
- A high level of abstraction to express algorithmic functionality while allowing full control over sequencing and resource use
- Simple timing model and **par** construct enables mixed sequential and parallel execution flow without FSMs
- Compact readable code enables designers to investigate a larger design space to find more optimal solutions.

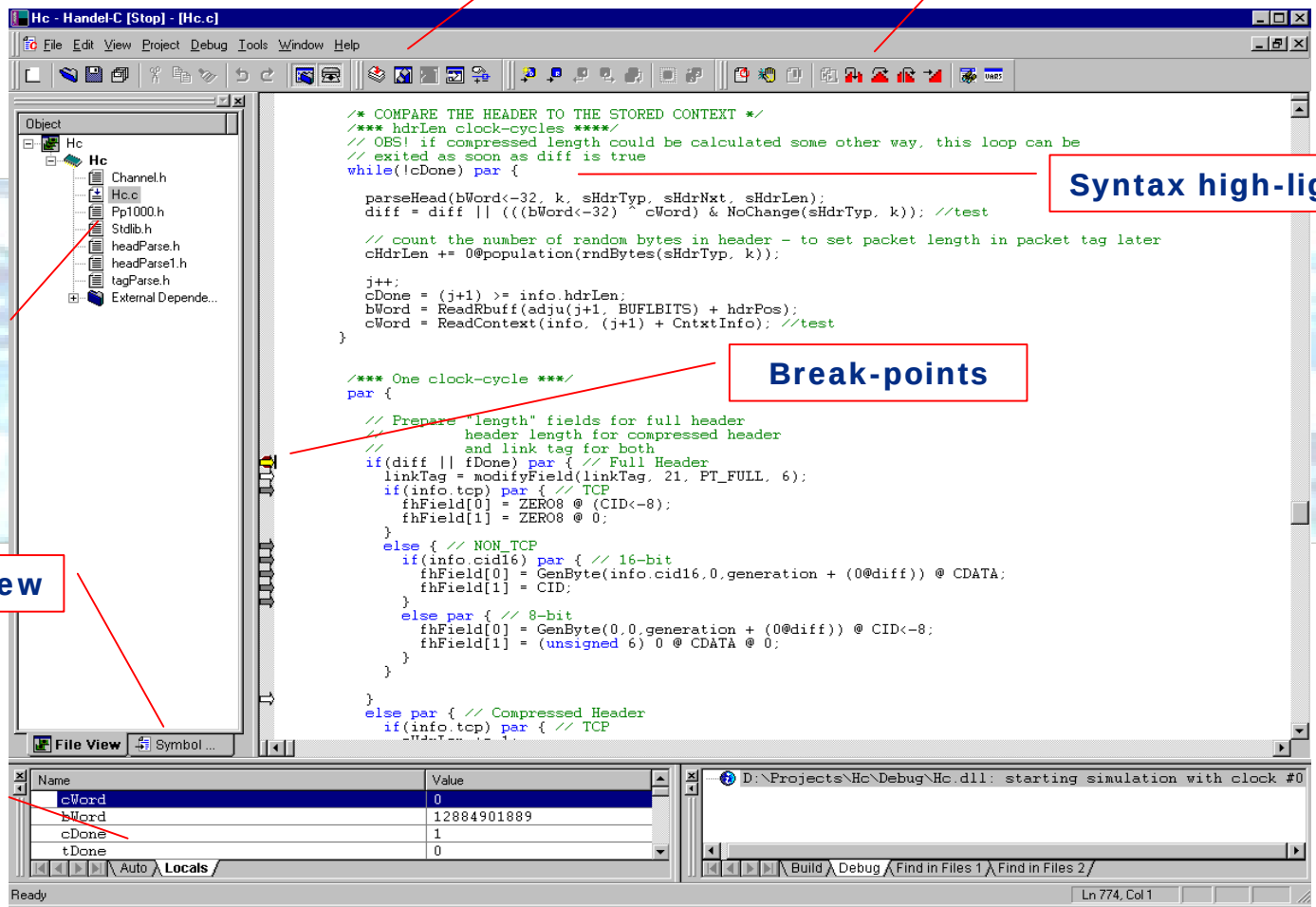


A software-like design approach?

A software-like design environment

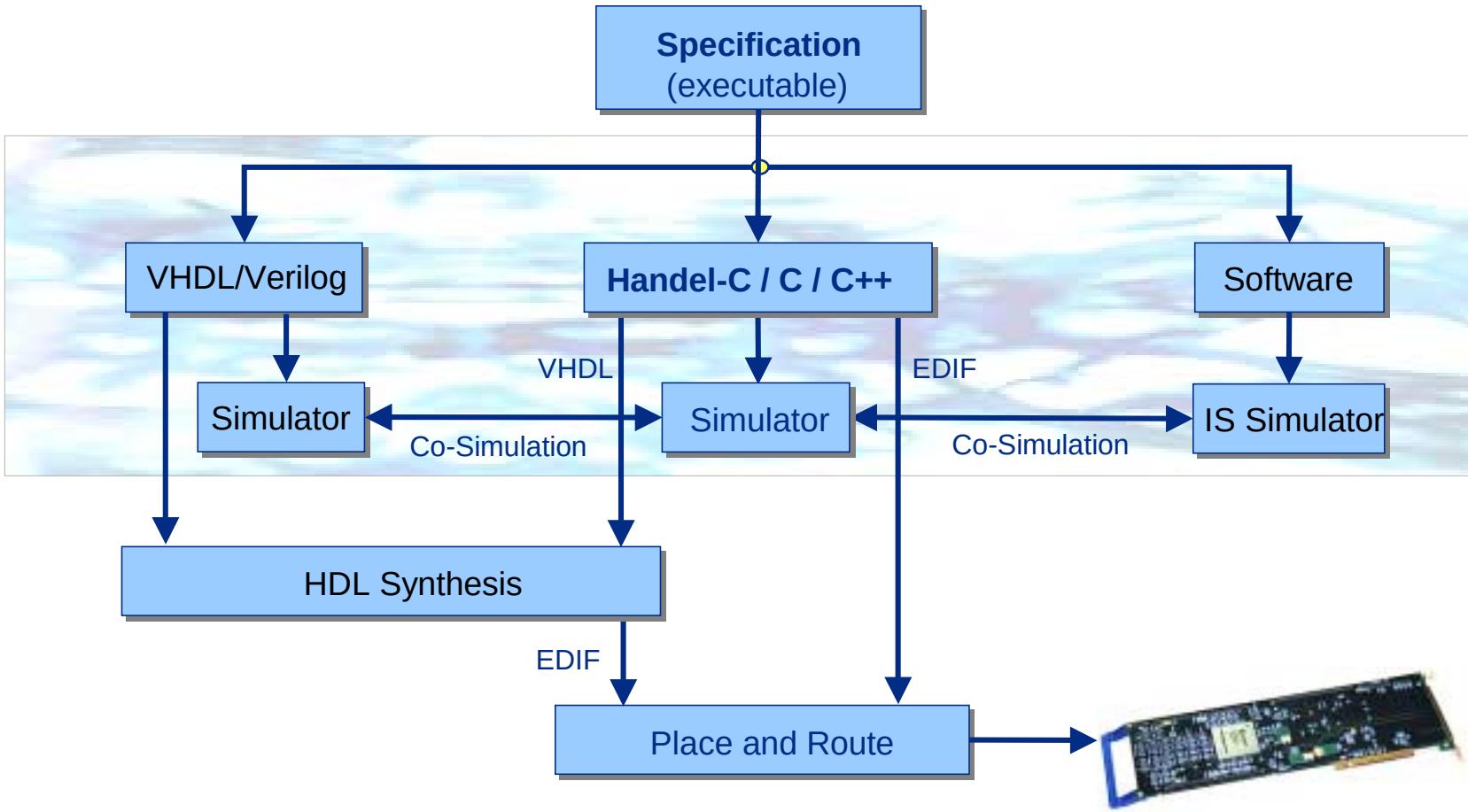
- An integrated design environment with the look and feel of a software design system
  - *Manage multiple source, header and object files*
  - *Symbol view with direct jump to definition or use*
  - *Syntax high-lighting*
- Verify and debug using a Symbolic Debugger
  - *Execute stepwise or to break-point or cursor*
  - *Watch selected and local variable values*
- Co-simulation with software and HDL-cores
- Fast compilation targeting FPGA/PLD (Xilinx, Altera, Actel)
- Immediate execution on plug-in prototyping boards

DK1.1 user interface



Celoxica
>: cutting a LONG story SHORT //

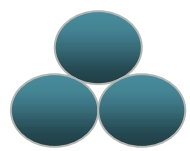
DK1.1 design flow



Celoxica
>: cutting a LONG story SHORT //

Benefits

- An Integrated Design Environment that includes source browsing, editing, symbolic debugging with the ability to follow parallel threads of execution.
- Fast simulation and compilation/synthesis as expected in a software design environment
- Co-simulation and verification facilities, that enable hardware/software partitioning decisions to be made at any stage in the design process.



A software-like design approach?

Software-like library access

- Access to predefined functionality via Header files and linkage with precompiled functions
- Standard libraries with language extensions
- Platform environment and access to peripherals and processors
- Project libraries
- Application libraries of third party IP-cores

```
#include "stdlib.h“
#include “Platformlib.h“
#include “Projectlib.h“
#include “Encryptionlib.h“
. . .

void main()
{
  . . .
}
```

Example: API for peripheral access

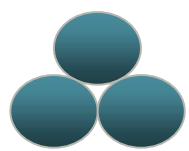
- Inverting colours in a video signal

```
#include “Platformlib.h”
VideoPixel Pixel;
VideoCoord X, Y;
.
.
par {
  VideoInInit(VideoIn[0], …);
  VideoOutInit(VideoOut[0], …);
}

while(1) par {
  VideoInGet (VideoIn[0], &X, &Y, &Pixel);
  VideoOutPut(VideoOut[0], X,  Y, ~Pixel);
}
```

Benefits

- Simplify development by providing an environment of predefined definitions, macros and functions
- Enable multiple team co-development
- Enable portability of application designs between different platforms
- Enable reuse of own and third party IP-cores



A software-like design approach?

Conclusions

- New technology that integrates processors and programmable logic on large chips is likely to trigger a paradigm shift in embedded system design
- Integrated technology that blurs the distinction between software, hardware and reconfigurable hardware
- Integrated design teams can work across traditional competence barriers
- Using similar languages, methods and tools for software and hardware design is a critical factor in utilizing this technology to its full potential

DK1.1 overview

- Designed for system architects, hardware and software developers
- Introducing software design methods for hardware design
  - *Increasing the productivity of hardware engineers*
  - *Enabling software engineers to target hardware*
- Language is based on ANSI-C
  - *A programming language for hardware*
  - *Not another HDL with C-language syntax*
- Suite includes functions typical of software development environment

Celoxica
cutting a LONG story SHORT

DK1.1 new features overview

- System level design
  - *Mixed Handel-C/C/C++ simulation*
  - *Co-simulation support for ARM and PowerPC embedded processors*
  - *APIs for peripheral access (PAL)*
- Compatibility
  - *Verilog output added to VHDL - improved structure and readability*
  - *Improved EDIF preserving names and added debug info*
- Optimization
  - *Technology mapping of gates to LUTs*
  - *Timing and area analysis tool with direct reference to source*
- New simulator improves speed by 100x
- New target technologies - Altera Excalibur and Actel parts
- Operating systems – Windows 95/98, NT, 2000, XP

Multiple language descriptions in DK1.1

- Design projects can include descriptions in Handel-C, C and C++
  - *C/C++ functions can be called from Handel-C and vice versa*
- This enables:
  - *High level modelling of system functionality in C/C++*
  - *Functional level software/hardware co-design*
  - *Incremental conversion of software to hardware*
  - *Test-benches in C/C++*
  - *Access to C/C++ for advanced analysis*

Co-simulation support

- High level functional co-simulation
- Support for plug-ins to ModelSim and cycle-accurate Instruction Set Simulators
- Back plane model enables powerful co-simulation
  - *Multiple simulators connected via a single node*
  - *Handel-C, C/C++, ModelSim and ARM, PowerPC SWIFT models*

Compatibility

- DK1.1 produces RTL-level Verilog output as well a VHDL
- The code is improved for readability and cross-referencing
  - *The hierarchy from the Handel-C code is preserved*
  - *The code is structured for readability*
  - *Names of functions and variables are preserved for cross-referencing*
  - *Code is targeted to simulators and synthesis systems (ModelSim, Synplify, Leonardo/Spectrum, FPGA Express)*
- EDIF output improved
  - *Names are preserved*
  - *Debug info for cross-referencing*

Fast simulation

- A new simulator is added to the DK1.1 Suite
  - *Cycle accurate*
  - *Compiles directly to native code*
- Executes in the order of 100 times faster than the current net-list based simulator
- Enables analysis of large designs

Optimisation

- Technology mapper
  - *Maps gates into LUTs of the target technology*
  - *Based on CutMap, an area efficient, depth optimal algorithm*
- Timing and area analysis tool
  - *Estimates area resource use*
  - *Estimates timing for critical paths*
  - *Both directly related to the source code*
- Gate and LUT based analysis

Target technologies

- New target technologies added:
- Xilinx
  - *Virtex-II Pro (PPC405, 3 Gbit serial I/O)*
- Altera
  - *EPXA10 – Excalibur (Apex2ke + ARM 922T)*
- Actel
  - *EX*
  - *54SX, 54SX-A, RT54SX, RT54SX-S*
  - *ProASIC, ProASIC+*

Operating Systems

- Windows
  - *98, NT, 2000, XP*
  - *Available now*