Hardware/Software Co-design
A System-Level Approach

Robert Self
rpself@essex.ac.uk
M. Fleury and A.C. Downton
fleum@essex.ac.uk

Overview

- The need for integrated hardware and software design strategies
- Using the ‘C/C++’ language for hardware - software codesign
- Two examples, SystemC and Handel-C
- System-level design

Design Process

- Product life cycle less than product development time - often < 6 months for consumer products
- Rapid increase in system complexity - designs often comprise, multi-rate hardware and software components
- Design productivity gap - growth in silicon capacity outstrips gain in engineer productivity

Technology Diversity

- μP with embedded FPGA coprocessor
  - Chameleon Systems CS2112
- FPGAs with hardwired embedded μP cores
  - Altera ARM and MIPS RISC processors
- ASICs with embedded FPGAs
  - FPGA cores from LSI Logic
- Runtime reconfigurable FPGAs
  - Xilinx Virtex (today)

Soft Architectures

- Instead of software running on fixed hardware architecture - ‘Soft Architectures’
- Paradigm shift, since hardware architectures are (runtime) configurable
- More choice - extends the design space further as both hardware and software are equally accessible as design elements
- Need single development model encompass-ing both hardware and software design

Single Language Approach

- Executable specification to verify functionality and baseline performance
- Design at higher level of abstraction
  - Focus on application specifics, instead of implementation detail
  - Productivity gain - rely on backend tools to perform implementation
- One approach, use C/C++
  - Why use C/C++ ?
  - Ok for software, but suitable for hardware ?

‘C/C++’ For Hardware Design

- Established as the mainstream language for engineering development
- Initial system models and algorithms are often developed using ‘C/C++’
- Large base of legacy code
- Reuse existing tools and development frameworks
- More engineers for hardware development

Problems Using ‘C/C++’

- Sequential language, need to support:
  - Concurrency
  - Inter-process communication (IPC)
- Hardware clocking
- Data types required to represent tri-state ports and busses
- Language subset such as pointers, casting, recursion, and dynamic memory not supported in hardware

Language Deficiencies

- Two main approaches - class libraries and language extension
- Class libraries -> SystemC
  - Cost efficient, since able to reuse existing development tools. Also, easily extended through the provision of additional class libraries
- Language extension -> Handel-C
  - Optimal linkage between language and implementation results in efficient solution

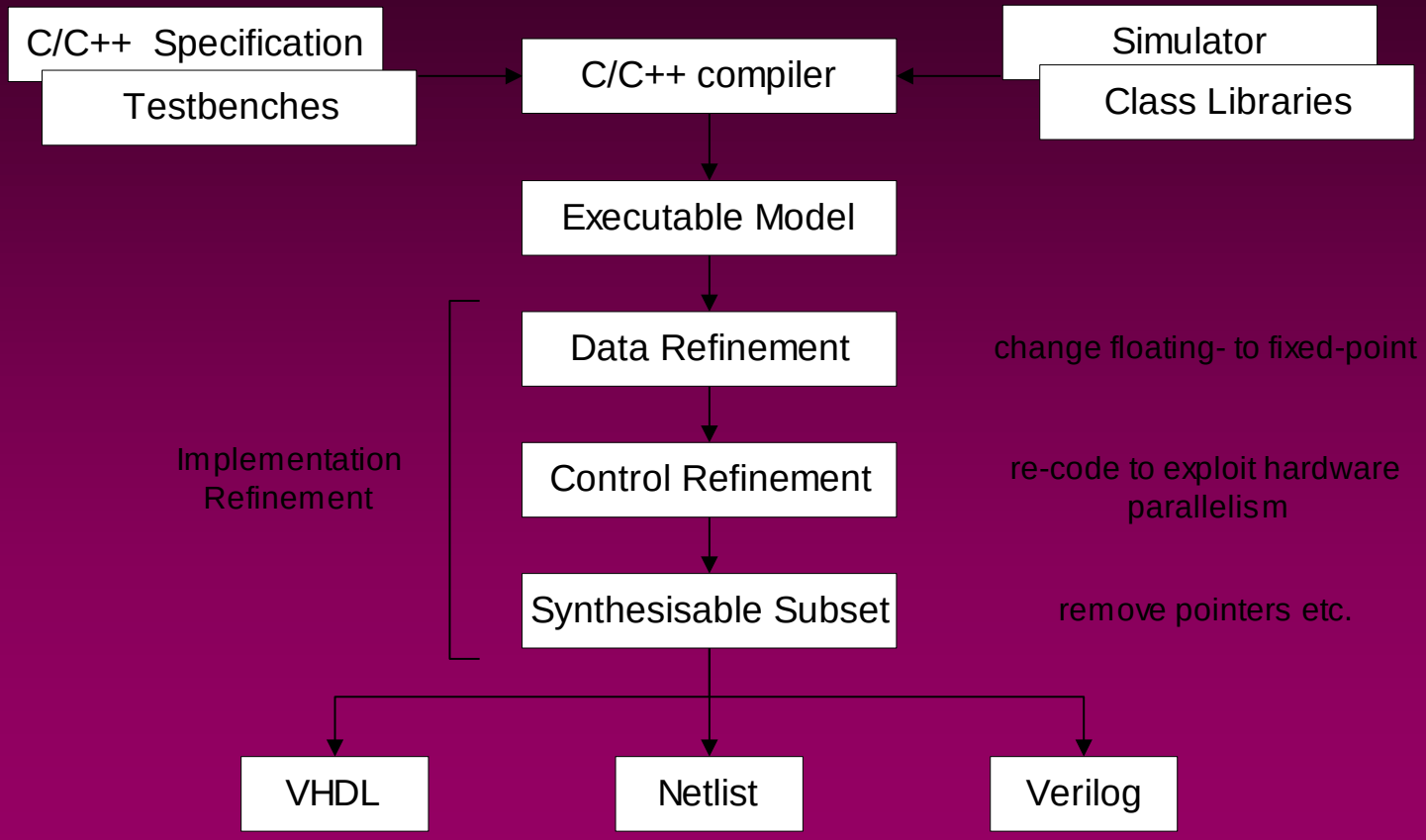
Unsupported C/C++ Features

- Advances in compiler technology needed to address issues such as pointer aliasing
- Some features, such as floating-point computation, are costly to implement in hardware so generally not supported
- Have to reengineer legacy code to conform to a ‘C/C++’ subset - limits the benefit of code reuse
- Now some examples - SystemC and Handel-C

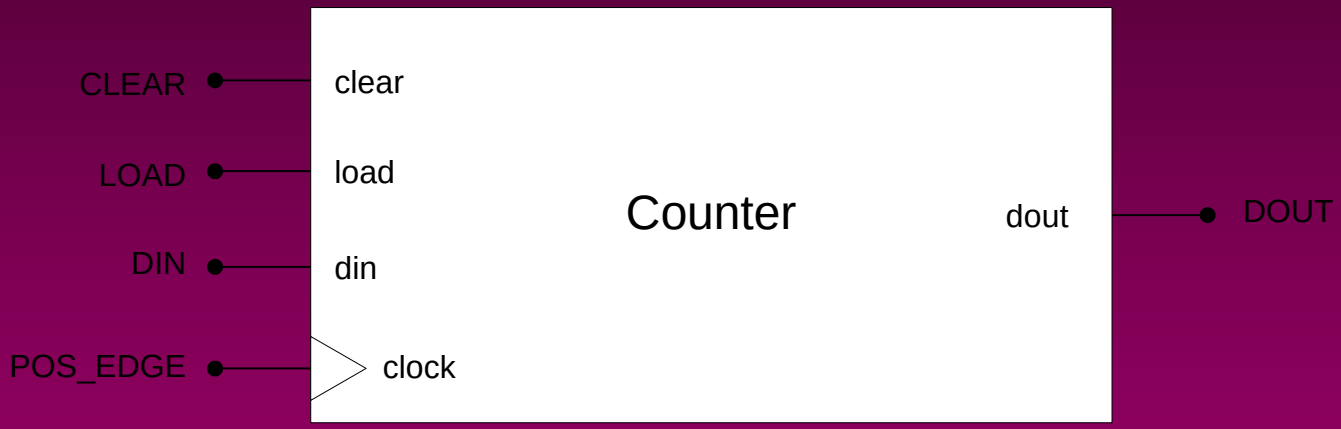
SystemC

- Industry led de facto standard, addresses the need for multi-vendor tool interoperability
- Founding members:
  - Synopsys: ‘Scenic’ modeling environment
  - CoWare: interface abstraction technology
  - Frontier Design: fixed-point data type expertise
- Class libraries and simulator supplied on the basis of no-fee license. Hardware synthesis and value-added tools provided by third-party vendors

SystemC Design Flow



Example: A Simple Counter



SystemC Source Code

Module Definition: Counter

```
SC_MODULE(counter) {
    sc_in<bool>             clock;
    sc_in<bool>             load;
    sc_in<bool>             clear;
    sc_in<sc_int<8> >       din;
    sc_out<sc_int<8> >      dout;

    int countval;

    void onetwothree();

    SC_CTOR(counter) {
        SC_METHOD(onetwothree);
        sensitive_pos(clock);
    }
};
```

port declarations

process

Run Method: onetwothree()

```
void counter::onetwothree() {

    if (clear == '1'){
        countval = 0;
    } else if (load == '1') {
        countval = din.read();
    } else{
        countval++;
    }

    dout = countval;
}
```

'run' onetwothree() on positive edge of clock

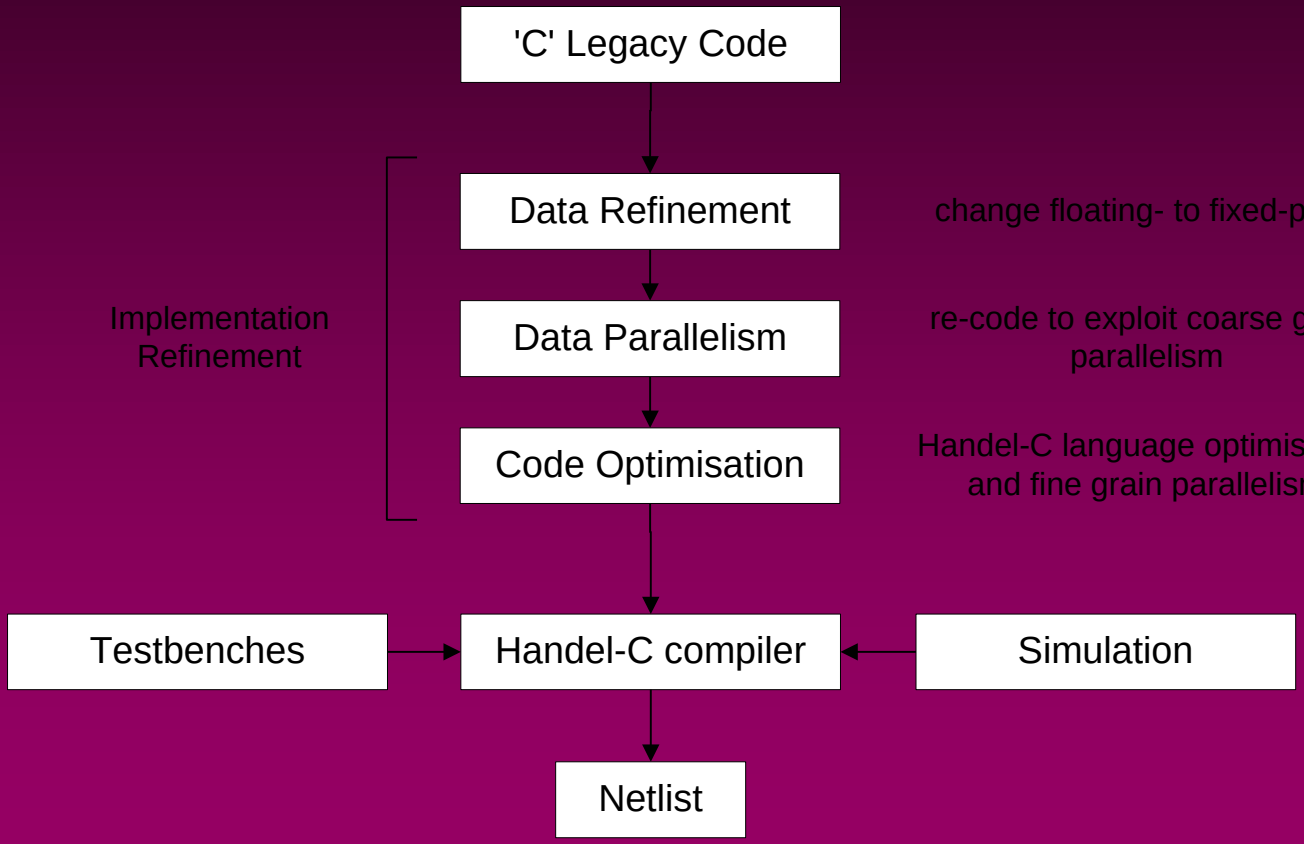
SystemC Summary

- Familiar C++ language constructs and programming style
- Ability to use existing compiler tools leverage existing C++ engineering skills and tool investment
- Class library approach offers flexible and simple enhancement mechanism
- Integrating executable spec. and simulator at binary-level improves simulation speed

Handel-C

- Intended for rapid hardware prototyping and educational use
- Integrates hardware compiler, synthesis, and simulation. Targets FPGA devices
- ‘C’ language syntax, supplement by Occam constructs
  - Channel types for IPC and external bus connections
  - ‘PAR’ statement implements hardware concurrency

Handel-C Design Flow



Handel-C Source Code

Macro Definition: Counter

```
macro proc counter(load, clear, din,
    dout){
    unsigned 8 countval;
    while(true){
        if(clear){
            countval = 0;
        }
        else if(load){
            din ? countval;
        }
        else{
            countval++;
        }
        dout ! countval;
    }
}
```

data width

channel read

channel write

internal channel

Main program:

```
void main(void)
{

    /* declare channels & variables */
    chan unsigned 8 ca_in, ca_out;
    chan unsigned 8 cb_in, cb_out;
    unsigned 1 clr, ld;

    /* instantiate processes */
    par{
        /* testbench */
        tb(ld, clr, ca_in, cb_in);

        /* define two counters */
        counter(ld, clr, ca_in, ca_out);
        counter(ld, clr, cb_in, cb_out);
    }
}
```

'PAR' statement creates processes

Handel-C Summary

- Raises the abstraction level
  - No registers, busses, or clocks
- Synchronous single clock timing model
  - Assignment and communication take 1 clock cycle
  - Expression and remaining statements take zero time
- Integrated solution for FPGA targeted development
- Not pure ‘C’ - significant learning curve due to Occam influence

SystemC & Handel-C

- SystemC: currently just an HDL, but extendable to higher levels of abstraction via class libraries - RPC semantics in next release
- Handel-C: Avoids hardware specifics, but model complexity limited, since only basic data types are available - complex data structures not supported i.e. no ‘structs’

Codesign - Current Status

- Separate hardware and software design flows fragment the development process
- Limits the opportunity for design exploration and design innovation
- Unable to perform system-level optimisation - leads to sub-optimal solutions
- Sub-system interoperability issues only become evident during final system testing
  - Leads to costly redesign and delay

Codesign - General Issues

- Mindset switch from sequential to concurrent programming
- Performance of backend place & route tools
- Bottom-up approach - currently need to implement designs in order to evaluate viability
  - Simulation is low-level and slow - often impractical to evaluate complex design scenarios
  - Backend tool performance compromises scalability and iterative development cycle

System-Level Design

- System-level approach to design
  - Top-down design strategy
  - Evaluate codesign scenarios by modeling instead of implementation
- Currently investigating ‘Models of Computation’ concept
- Heterogeneous simulation using Ptolemy framework
  - Evaluates interoperability between abstract models i.e. CSP and discrete event models