Intro to High Level Design with SystemC

Aim

- To introduce SystemC, and its associated Design Methodology

Date

- 26th March 2001

Presented By

- Alan Fitch
Silicon Complexity vs. Software Complexity

Silicon complexity is growing 10x every 6 years

Software in systems is growing faster than 10x every 6 years

Growth in software complexity in consumer products

- Microprocessors
- Microcontrollers

© Copyright 2001 Doulos Ltd. All Rights Reserved
Attacking Complexity

- Work at a higher level of abstraction
- Reuse existing designs
  - Cut ‘n’ paste
  - Virtual components (VCs)
  - Capture Intellectual Property in-house
- Platform based design

uP core ROM RAM DSP ATM MPEG INTERFACE
Levels of Abstraction

- System analysis
- System level modeling
- H/W S/W partitioning
- Architectural exploration
  - H/W spec
  - Analog spec
  - Digital spec
  - S/W spec

Integration and verification

Many of these steps are presently manual!
Crossing the Gap

• Different sorts of engineers have different areas of expertise
  • They think differently
  • They use a different design language

• Each area of design (analogue, digital, physical, software) requires a different (but overlapping) set of skills

• Within an area of design there is a series of steps from high levels of abstraction to low levels of abstraction. Each level
  • May use different languages
  • May require a different “mindset”
  • May be capable of automatic translation to the next lower level
Main requirements of hardware systems

- Concurrency: hardware systems are *parallel*. Almost all hardware systems include concurrency.

- Reactivity: hardware systems react at transition on signals in the system.

- Distributiveness: hardware systems are made of a set of basic computation units that may
  - process data at different speeds
  - use different clock rates

- Timing: specification of hardware systems includes timing concepts for the synchronisation of parallel and distributed behaviour.
Language Comparison

System-Level Languages

- C/C++-based
  - SystemC
  - Cynlib
  - SoC++
  - A|RT
  - VHDL+

- VHDL/Verilog Replacements
  - SoC++
  - A|RT
  - VHDL+

- Higher-Level Languages
  - SDL
  - SLDL
  - SUPERLOG

- Entirely New Languages
  - Java-based
  - Java

- Entirely New Languages
  - SLDL
  - SUPERLOG
  - Java
C/C++ Based Languages

Software/System engineer
- Conceptualize
- C/C++ simulation
- Write specification doc

Hand over specification doc
Executable spec
Testbench

Hardware engineer
- Understand
- Refine in C++, SystemC
- Verify reusing testbench
- Synthesize from C++, SystemC

Faster simulation speed
Tools already on the market
C/C++ familiar to system and S/W engineers
Development of SystemC

Synopsys “Scenic”

IMEC

CoWare N2C

Abstract Modelling

SystemC Version 0.9

SystemC Version 1.0

SystemC Version 1.1

SystemC Version 1.0.1

Onward to Version 2!

Synopsys “Fridge”

Frontier Design A|RT Library

Fixed Point Types

© Copyright 2001 Doulos Ltd. All Rights Reserved
Levels of Abstraction

• Untimed Functional (UTF)
  • separates communication from behaviour
  • uses Remote Procedure Call (RPC) paradigm

• Timed Functional (TF)
  • allows allocation of time to behavioural blocks

• Bus Cycle Accurate
  • models interfaces accurately using three handshaking methods

• Cycle Accurate (CA)
  • essentially Register Transfer Level (RTL)
RTL Example

- EXOR gate built hierarchically from 4 NAND gates
- Each design created using
  - header file (contains C++ constructor)
  - source code file (contains algorithm)

```c++
// nand2.cc
#include "nand2.h"

void nand2::do_nand2()
{
    F = !(A & B);
}
```
// nand2.h
#include "systemc.h"

SC_MODULE(nand2)
{
    // ports
    sc_in<sc_bit> A;
    sc_in<sc_bit> B;
    sc_out<sc_bit> F;

    void do_nand2();

    SC_CTOR(nand2)
    {
        SC_METHOD(do_nand2);
        sensitive << A << B;
    }
};
// exor2.h
#include "systemc.h"
#include "nand2.h"

SC_MODULE(exor2)
{
    sc_in<sc_bit> A;
    sc_in<sc_bit> B;
    sc_out<sc_bit> F;

    // 4 nand2 instances
    nand2 n1;
    nand2 n2;
    nand2 n3;
    nand2 n4;

    sc_signal<sc_bit> S1;
    sc_signal<sc_bit> S2;
    sc_signal<sc_bit> S3;
EXOR Gate (2)

SC_CTOR(exor2): n1("N1"), n2("N2"),
       n3("N3"), n4("N4")
{
    n1.A(A);
    n1.B(B);
    n1.F(S1);

    n2.A(A);
    n2.B(S1);
    n2.F(S2);

    n3.A(S1);
    n3.B(B);
    n3.F(S3);

    n4 << S3 << S3 << F;
}
};
//
// exor2.cc
//
// This file has nothing in it:
// the design structural
//
#include "exor2.h"
Conclusions

• C/C++ languages seems to be gaining acceptance
• Presently, synthesis techniques from C to hardware are appearing (e.g. A|RT Designer)
• But the link to software is “informal”

• Taking SystemC as an example
  • The present state of SystemC is that certain tools can synthesise hardware - SystemC allows you to write C++ descriptions that behave like VHDL or Verilog
  • A formal link to software (e.g. perhaps some kind of generic RTOS API) remains in the future

• SystemC has industry “momentum”
  • But methodology and tools need to evolve