SoC design methodology

Using SystemC

Bart Vanthournout

Agenda

- ESL trends
  - Network on Chip, Application Specific processors (ASIP)
  - SoC design requirements, Abstraction levels
- SystemC Transaction level modeling
- CoWare tools
Key Challenges With Chip Design

Relative Effort by Designer Role

- Software
- Validation
- Physical
- Verification
- Architecture

Source: IBS, November 2002

What is changing

- Magarshack & Paulin (DAC'03)
  - **Cause:**
    - Shrinking of design technology
    - Increase in NRE cost for manufacturing (>1M$) and design (10-100M$) of SoC
      - For non multi-million chip volume cost needs to be amortized over multiple products
  - **Observation:**
    - IP reuse is not sufficient
    - IP is not fixed with shrinking technologies leading to verification and design-for-test issues
  - **Methodology Requirement:**
    - Need for revolutionary design methods enabling:
      - Faster "Time To Market" through IP reuse, standard communication interfaces and scalable interconnect topology (NoC)
      - Increased flexibility through SW programmability and configurable HW
      - Enable to map an application to a platform to increase the productivity of a platform user
What is changing

- Magarshack & Paulin (DAC'03)
  - **Solution:**
    - Emergence of flexible, domain-specific, SW programmable platforms (processor, IP, interconnect)
    - Methodology based on 4 distinct abstraction levels
      1. System application design
      2. Multiprocessor SoC platform design
      3. Highlevel IP block design
      4. Semiconductor technology and basic IP
  - **Tool requirement:**
    - Correctly map of highlevel abstraction to the lower layers
    - Topdown methodology

- Sangiovanni-Vincentelli & Grant Martin (CASES'01)
  - **Embedded SW design Vision**
    - **Situation:**
      - Embedded SW is an implementation choice for a function that can be implemented as HW as well
      - Increasingly more ESW due to platform based design, where re-use and programmability are used to share development cost
    - **Methodology Requirement:**
      - Capture system requirements at higher levels of abstraction
      - Close interaction between HW/platform definition and the ESW that will run on it
      - Platform definition based on understanding of applications that will run on it
      - SW design with good characterisation of the HW
    - **Solution:**
      - Need to define a topdown methodology taking a global view on all interacting aspects of the system
SoC designs of the future

- Design questions
  - Algorithm mapping
  - SW development
  - Custom HW design
- IP questions
  - Reuse
  - Selection
  - Configuration
- Architecture questions
  - Optimization
  - Exploration

Why NoC?

- Shrinking technology Leads to new interconnect strategies
  - Benini & de Michele (DATE 2002):
    “Delays on wires will dominate: global wires spanning a significant fraction of the chip size will carry signals whose propagation delay will exceed the clock period.”
  - K. Goossens & Van Meerbergen (DATE 2002):
    “A NOC hardware architecture based on a packet-switched router network … breaks the fatal global timing closure loop by separating inter-IP from intra-IP communication, and can so reduce global design iterations.”
- What is NoC?
  - Less but ‘programmable’ wires by introducing switches (routers).
    - Shared bus: communication bottleneck
    - Point to point connection: many under-utilized long wires
  - Structured approach to interconnect; wires are either
    - short to get on the network,
    - router to router.
  - Separation of computation (IPs) and communication (NOC)
Interconnect Trends - today

System on Chip

- connect up to hundred Resources
- I/O requirements GBit/s range
- traffic management / QoS
- platform scalability
- power efficiency
- latency

Future SoC Interconnect Challenges

- Physical
  - Clock distribution
  - Latency management
  - Transaction integrity

- Functional
  - Traffic management
  - Allocation of resources
  - Signalization
Why ASIPs? The Energy-Flexibility Gap

**Log PERFORMANCE**

- General Purpose Processors
  - StrongARM 110
  - TMS320C54x

- Digital Signal Processors
  - 0.4 MIPS/mW
  - 3 MIPS/mW

- Application Specific Signal Processors
  - Application Specific ICs
  - Physically Optimized ICs

- Field Programmable Devices
  - 20-35 MOPS/mW

Source: T. Noll, RWTH Aachen

---

Why ASIPs – technological and economical

**Technological**
- Efficient heterogenous Processing Platforms in Mobile Communication
- Application Specific High Performance Requirements (Network Processors)

**Economical**
- Simple, but cost-sensitive Applications
  - Cost (No Royalty and licence Fee)
  - IP Protection and Reuse (IP is about applications)
  - Verification
  - Time-to-market, development time reduction
  - Flexibility through programmability
Energy Efficiency – Optimization Example (I)

\[ P(t) \]

\[ P_{\text{unopt}} \]

\[ P_{\text{opt}} \]

Intrinsic Energy

Overhead Energy

\[ \text{Runtime} \]

**Intrinsic**: refers to useful arithmetic operations / data routing

- Intrinsic energy nearly scheduling-independent
  - (serial/parallel processing does not matter)

Measurement results:

- **overhead power** nearly constant
- **intrinsic energy** nearly constant (only scheduling changed)

Energy Efficiency – Optimization Example (II)

Measurement results:

- **reduce overhead energy**
Constructive ASIP Design Approach

Start with minimum "basic instruction set architecture" e.g. a register-register RISC architecture:

<table>
<thead>
<tr>
<th>Load/Store</th>
<th>read/write memory and I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>+, -, *, arith. shift, abs</td>
</tr>
<tr>
<td>Logic</td>
<td>and, or, xor, logical shift</td>
</tr>
<tr>
<td>Control</td>
<td>compare-set-status, uncond. and cond branches, branch to subroutine, return from subroutine, stop run</td>
</tr>
</tbody>
</table>

→ apply application-specific optimization ("constructive")

Optimization steps towards "best fit"

- **exploit regularity/parallelism in data flow/data storage**
  - optimize data organization and interfaces
  - use appropriate memory and I/O bandwidth

- **exploit regularity/parallelism in operation**
  - chain, parallelize and pipeline operations
  - optimize frequently executed blocks like loop bodies
  - provide the right degree of parallelism (functional units)
  - use only resources that are really needed

- **optimized control of computations**
  - add instructions/control mechanisms to exploit parallel funct. units
  - use high instruction coding density
  - use low-power guarding techniques and clock gating
  - smart low-power encoding techniques to decrease toggle count
CoWare’s system design vision

Create System specification
- Functional description
- Identify IP reuse and platform requirements

HW/SW architecture specification, Platform design
- Highlevel HW/SW architecture exploration
- 'Programmers view' model, to enable SW design and HW/SW trade-offs

HW implementation
- Platform and HW refinement, interconnect micro architecture definition

Technology mapping
- RTL

It's time to move up, yet again!

Hardware centric history

© CoWare, Inc. 1997-2003
CoWare Confidential
TopDown Design

Executable specification (UT description)

- Take algorithm and map it to an application platform
- Ensure HW/SW communication is implemented correctly on the platform
- Reuse application platform over many designs
- Simulation speed sufficient to do development of large pieces of SW

Transaction level modeling
Transaction Level Modeling: What Is It?  
A Higher Level Of Abstraction For Communication

- **RTL:** The bus is merely wires  
  - Each device on the bus has a pin-accurate interface  
  - Each device interface must implement the bus protocol

- **TLM:** The bus model is key  
  - Bus model enforces the bus protocol  
  - Each device communicates via transaction level API  
  - Less code, fewer pins, fewer events => much faster

---

Range of abstraction levels covered by TLM

- **RTL**  
  - Signal
  - Transfer
  - Transaction
  - Message
  - High Level protocol

- **TLM**  
  - Communication Layers
  - HW Layer
  - High Level protocol
Communication versus Behavior

**Behavior:**
For a complex IP, this is where the real value is. At TLM, IP reuse should be based on behavior reuse.

**Communication:**
TLM is targeted for communication modelling. This is where the TLM API calls are located.

**Boundary:**
This is where synchronization between behavior and communication is done.

---

**Transaction Level Modeling**

- SW developers and HW designers/verifiers work different parts of the curve
- Transaction level moves HW-SW co-design to a new curve

*TLM allows complex SoC platforms to be simulated accurately enough for architectural exploration, firmware development and verification use models*
Transactions & Transfers

- Transfer
  - Groups all attributes ("Pins") that have the same timing
  - Represents the relevant events of a transaction
    - A transfer can be send when its attributes are allowed to be driven on the bus, but not later.
    - A transfer can be send only once
  - Performs uni-directional data exchange
    - Between bus master, bus slave and the bus
  - The attributes have the same direction as the transfer

Arbitration

Transfer: Atomic operation

Transaction
**TLM API**

All the API are non-blocking

<table>
<thead>
<tr>
<th>Transactions</th>
<th>Transfers</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>port.getTransaction()</code></td>
<td><code>port.getTransactionName()</code></td>
</tr>
<tr>
<td><code>port.canSendTransaction()</code></td>
<td><code>port.canSendTrf(Name)</code></td>
</tr>
<tr>
<td><code>port.canReceiveTransaction()</code></td>
<td><code>port.canReceiveTrf(Name)</code></td>
</tr>
<tr>
<td><code>port.sendTransaction()</code></td>
<td><code>port.sendTrf(Name)</code></td>
</tr>
<tr>
<td><code>port.sendDelayedTrf(Name,delay)</code></td>
<td><code>port.sendDelayedTrf(Name,delay)</code></td>
</tr>
</tbody>
</table>

`port` = TLM port

---

**TLM API**

- **Transfer sensitivity**
  - allows to query the timing of the bus to find out when attributes can be send/received
  - Prevents the user from violating the bus timing
  - Does not force the user to code the timing of the bus in an FSM in every peripheral
  - Provides a generic coding style that allows to re-use peripherals within a certain class of buses

- **All the attributes of the transaction can be accessed from a transfer**
  - Prevents unnecessary bookkeeping in every peripherals in case of pipelined protocols
TLM : Bus simulator

Sending and Receiving transfers

Initiator is allowed to send a Write Data transfer during this time slot.

The bus synchronizes the transfers with the target according to the timing of the protocol.

Concepts: Sending a transaction

```cpp
if (P.getTransaction()) {
    P.Transaction->setAddress(0x1000);
    P.Transaction->setType(tlmWrite);
    P.Transaction->setWriteData(0x2000);
    P.sendTransaction();
}
```
**Concepts: Sending/Receiving transfers**

- **AddrTrf**
- **HADDR**
- **HBURST / HTRANS**
- **HSIZE / HPROT / HWRITE**

// Initiator can send transfer
if (P.getAddrTrf()) {
    P.AddrTrf->setAddress(1234);
    P.AddrTrf->setType(tlmWrite);
    P.sendAddrTrf();
}

// Target can receive transfer
if (P.getAddrTrf()) {
    addr = P.AddrTrf->getAddress();
    ftype = P.AddrTrf->getType();
}

---

**TLM process**

A systemC process can be sensitive to a transfer event. The bus simulator will trigger this process whenever an action from the initiator or the target is allowed.

- Send or receive transfer

```systemC
SC_MODULE (MyModule) {
    TLMTargetPort port;
    void send_address() {
        ...
    }
    SC_CTOR (MyModule) {
        SC_METHOD (send_address);
        Sensitive << port.getSendAddressTrfEventFinder();
    }
}
```
Target : Write transaction, 0 wait, Ok response (2)

Coding style: static sensitivity to transfer
(Equivalent to previous example)

```c
SC_METHOD(receiveWriteData);
sensitive << p_bus.getReceiveWriteDataTrfEventFinder();
dont_initialize();

SC_METHOD(sendEot);
sensitive << p_bus.getSendEotTrfEventFinder();
dont_initialize();

void receiveWriteData() {
    P1.getWriteDataTrf();
    myVar = P1.WriteDataTrf->getWriteData();
}

void sendEot() {
    P1.sendEotTrf();
    myVar = P1.getWriteDataTrf()->getWriteData();
}
```

Either one can be used in case of sensitivity to the specific transfer.

Target : Write transaction, 0 wait, Ok response (3)

```c
void receiveWriteData() {
    P1.getWriteDataTrf();
    myArray[P1.WriteDataTrf->getAddrTrf()->getAddress()] =
    P1.WriteDataTrf->getWriteData();
}

void sendEot() {
    P1.sendEotTrf();
}
```

The bus simulator provides a link between a transfer and its transaction.
Transactional Bus Simulator

- A complete off-the-shelf solution
- Fully models the AMBA 2.0 bus specification at the transaction-level
- Innovative CoWare technology optimizes performance while retaining cycle accuracy
- Fully SystemC 2.0 compliant
1. Open IP libraries

2. Drag and drop blocks and bus nodes from the IP libraries into the Design Editor window. The block appears in your Workspace.

3. Connect elements with Connection Tool.

4. Select blocks and nodes to set memory map and other parameters.

5. Export system design (Tools menu).
Platform Creator Usage – Top-down Design

1. New Workspace
2. Open Scenario Library
3. Open TLM Platform
4. Open UT specification (application)
5. Partition the HW and SW blocks
6. Resolve the abstract channels
7. Set the memory map and parameters
8. Export the system

System-Level Analysis
ConvergenSC System Designer

- Superior Hardware, Software, Memory, and Bus Analysis for SystemC
  - Which masters and slaves should be on which bus layer?
  - Is the cache the right size?
  - How much memory is needed?
- Comprehensive APIs
  - Fully customizable data collection and display
  - Fully accessible to designer

Enables the Right System Architecture, Performance, and Embedded SW Trade-offs Sooner
Mixed Language SystemC-HDL Simulation

- Use Verilog and VHDL models
  - To verify implementation
  - For legacy IP re-use
- ConvergenSC supports the following simulators:
  - Cadence NC-Sim
  - Synopsys VCS
  - MTI modelsim
- Capabilities:
  - Automated SystemC-HDL executable generation
  - Fast, single process simulation
  - Multiple HDL blocks
  - Mixed HDL language (if supported by HDL sim.)

LISATEK product family

- Flexible platforms...include more processors
- Embedded FPGA to allow customers to extend processor instruction set
- LISATek offers a complete design flow for ASIPs
  - Allows existing designers to automate processor development
  - Creates ISS, C Compiler and S/W development tools
  - Synthesises RTL from processor description
- ConvergenSC offers the best solution for designing platforms
LISA 2.0 – Instruction Set Modelling

- **Design complex instruction sets**
  - Integrated design environment (GUI)
  - Hierarchical description style
  - Instruction (binary) encoding
    - Arbitrary bitwidth and format
  - Instruction (assembly) syntax
  - Hardware Behavior
    - Pure C or C++ code
    - Integration of existing libraries
    - Additional data-types ease modeling

Integrated Simulator-Debugger-Profiler

- **Immediately explore the architecture prototype**
  - Source-code & Disassembly
  - Command-line control
  - Loop & Instruction Profiler
  - Memories & Registers
  - Model Debugging
  - Unit & Resource Utilization
  - Tracing
CoWare Confidential

Debugger

Coverage

Source Profiling

C/C++ Variables

Backtrace

GDB Command Line

CoWare Confidential

CoSy compiler system (ACE)

- Universal retargetable C/C++ compiler
- Extensible intermediate representation (IR)
- Modular compiler organization
- Generator (BEG) for code selector, register allocator, scheduler
- Permits building working compilers quickly

CoWare Confidential
C-Compiler GUI approach

- Analysis tools extract as much compiler information as possible from LISA model (e.g. instruction latencies, registers, ASM syntax)
- GUI guided extension and refinement by user (e.g. code selection, stack layout, type bitwidths)
- Emission of compiler description file

RTL generation GUI
CoWare ESL Solution Flow

- **System Functional Validation**
  - CoWare ESL Solution Flow

- **Architectural Validation**
  - ConvergenSC
    - System Designer
    - System Verifier
    - Platform Creator

- **Refinement & Verification**
  - SPW
    - Vertical Apps.
    - Libs

- **Implementation**
  - RTL2GDSII
  - HW Impl. Option
  - ESP
  - ISS & S/W tools

**IP**

- Bus Compiler
- C Compiler
- Processor Spec
- ISP & S/W tools
- Periph & Mem IP
- Vendor PPs
- Bus Models
- LIb
- LISATek

**Algorithm**

- Algorithms
- Vertical Apps.
- Libs

© CoWare, Inc. 1997-2003 CoWare Confidential