

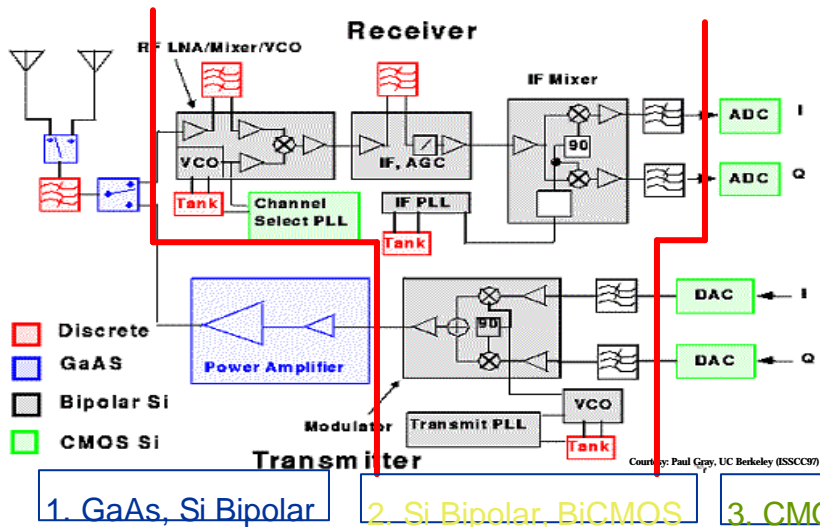
Low Power Wireless Networked System Design

Rajesh K. Gupta

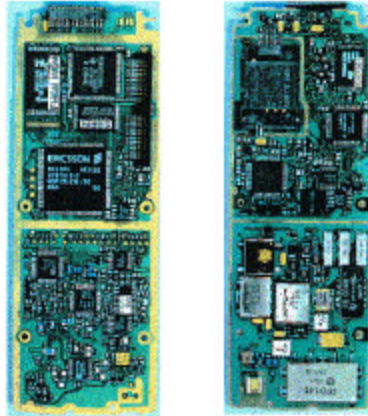
Computer Science and Engineering
University of California, San Diego

Globecom December 2, 2003, San Francisco, CA

RF Transceiver Block Diagram

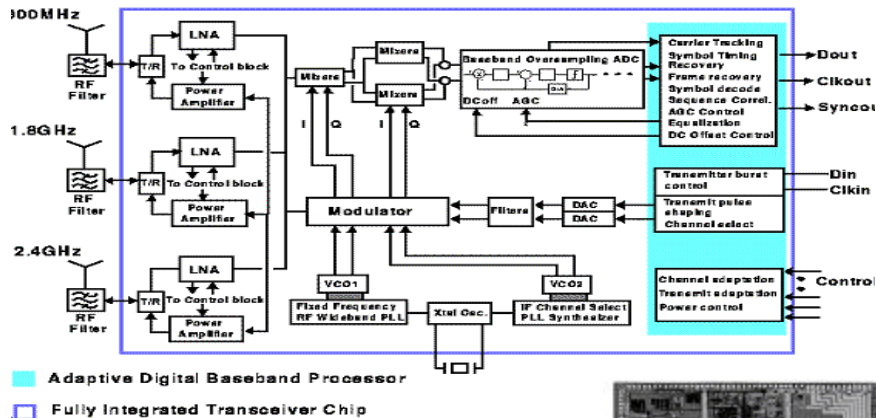


Typical RF Transceiver Implementation



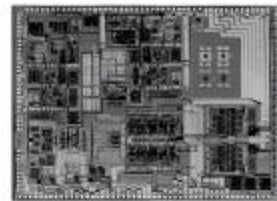
Courtesy: Paul Gray, UC Berkeley (ISSCC'97)

UCB DECT CMOS Adaptive RX



0.6 μm CMOS, 7.5mm x 6.4mm die size.
 55dB image rejection, mostly from IR mixer.
 198 mW power. RX gain max 78 dB, min 26 dB

Courtesy: Rudell, et al, ISSCC'97

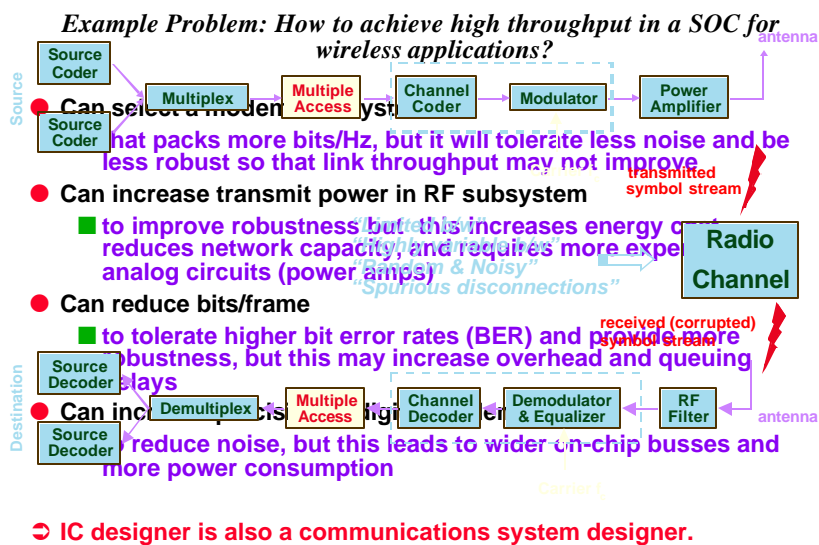


Wireless Systems Designs

- **Key Drivers**
 - **Relentless digitization of signals and systems**
 - ◆ high speed digital circuits, ADCs leading to IF (and even RF) processing in digital domain, direct conversion techniques
 - ◆ complex communication algorithms and increasing available MIPS favor digital implementation
 - **Microelectronic advances in CMOS VLSI**
 - ◆ (RF as well as base-band digital processing), MEMS structures
- ⇒ These trends are making it increasingly possible (and even cost effective) to build **Wireless Systems on a Chip**

5

Systems Engineering for SOCs



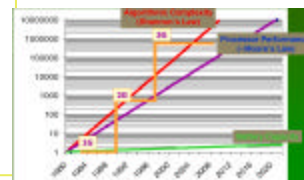
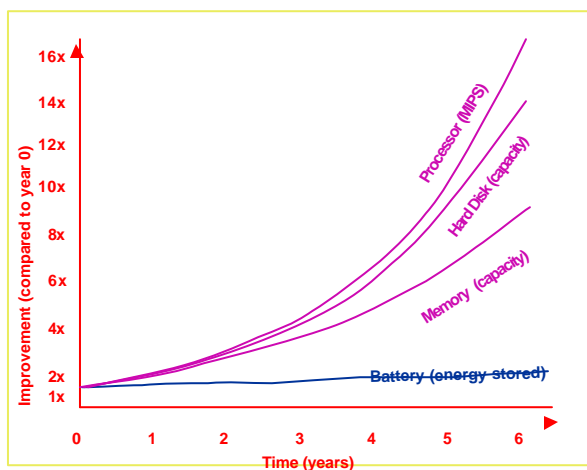
6

Wireless NES System Characteristics

- **Wireless**
 - limited bandwidth, high latency (3ms-100ms)
 - variable link quality and link asymmetry due to noise, interference, disconnections
 - easier snooping
 - ⇒ need for more signal and protocol processing
- **Mobility**
 - causes variability in system design parameters: connectivity, b/w, security domains, location awareness
 - ⇒ need for more protocol processing
- **Portability**
 - limited capacities (battery, CPU, I/O, storage, dimensions)
 - ⇒ need for energy efficient signal and protocol processing

7

Energy Availability Growth limited to 2-3% per year



J. Rabaey, BWRC

➔ Need to be energy efficient at all levels and in all tasks.

8

Computational Efficiency

- Speed power efficiency has indeed gone up

- 10x / 2.5 years for mPs and DSPs in 1990s

- ◆ between 100 mW/MIP to 1 mW/MIP since 1990

- IC processes have provided 10x / 8 years since 1965

- rest from power conscious IC design in recent years

- Lower power for a given function & performance

- e.g. 1.6x / year reduction since early 80s for DSPs (source TI)

- Most optimistic projections at best stop at 60 pJ/op (about 20X)

Processor	MHz	Year	SPECint-95	Watts
P54VRT (Mobile)	150	1996	4.6	3.8
P55VRT (Mobile MMX)	233	1997	7.1	3.9
PowerPC 603e	300	1997	7.4	3.5
PowerPC 604e	350	1997	14.6	8
PowerPC 740 (G3)	300	1998	12.2	3.4
PowerPC 750 (G3)	300	1998	14	3.4
Mobile Celeron	333	1999	13.1	8.6

- ➔ However, circuit gains are nearing a plateau

- circuit tricks & voltage scaling provided a large part of the gains

- while energy needs (functionality, speed) continue to climb

- 10x increases: in gate count (7 years); in frequency (9 years)

9

Efficiency in Communications

- Power Efficiency (or Energy Efficiency) $h_p = E_b/N_0$

- ratio of signal energy per bit to noise power spectral density required at the receiver for a certain BER

- high power efficiency requires low (E_b/N_0) needed for a given BER

- Bandwidth Efficiency $h_b = \text{bit rate} / \text{bandwidth} = R_b/W$ bps/hz

- ratio of throughput data rate to bandwidth occupied by the modulated signal (typically range from 0.33 to 5)

- Often a trade-off between the two

- e.g. for a given BER

- ◆ adding FEC reduces h_b but reduces required h_p

- ◆ modulation schemes with larger # of bits per symbol have higher h_b but also require higher h_p

- for PSK, QAM, generally higher bw efficiency decreases power efficiency

10

Effect of Improving BW efficiency through modulation

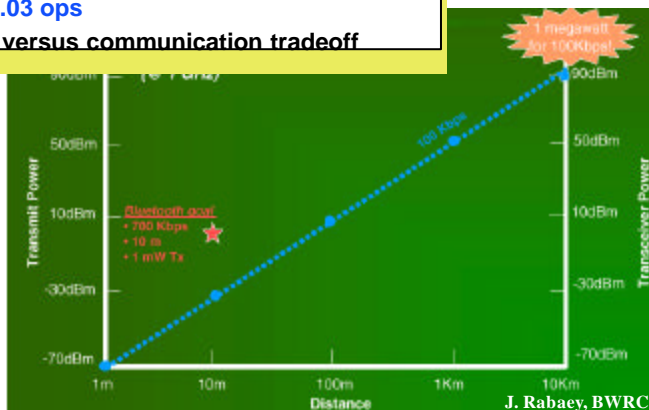
- For a 10^{-5} BER and fixed transmission BW

#bits per symbol	Power Penalty Factor	Bit-rate Gain Factor	Energy Penalty Factor
2	1	1	1
4	2	2	1
8	4.7	3	1.56
16	10	4	2.5
32	20.7	5	4.1
64	42	6	7

11

Communication vs. Computation

- Computation cost (2004 projected): 60 pJ/op
- Minimum thermal energy for communications:
 - 20 nJ/bit @ 1.5 GHz for 100 m
 - ◆ equivalent of 300 ops
 - 2 nJ/bit @ 1.5 GHz for 10 m
 - ◆ equivalent of 0.03 ops
- ➔ significant processing versus communication tradeoff



The Need

- Power consumption, energy efficiency is a system level design concern
 - efficiency in computation, communication and networking subsystems
- The energy/power tradeoffs cut across
 - all system layers: circuit, architecture, software, algorithms
 - need to choose the right metric
- Power awareness goes beyond low power concerns
 - make tradeoffs against performance, quality measures against application constraints

13

Presentation Focus: Techniques & Tools

Goal: A systems view of integrated wireless system design for low power.

- Design technology challenges in networked SOCs
 - ◆ Two views of Networked SOCs
 - > compositional (or ASIC view)
 - > architectural (or network-centric view)
 - ◆ Scope and categories of design tools for NSOCs
 - ◆ Network architectural modeling
 - ◆ System-level composition through OO mechanisms
- Design of Wireless Systems
- Example Platform: TI/OMAP
- Energy awareness: ARM platform

This talk will NOT describe:

- detailed algorithms used in CAD tools
- theory of radio and communications system design
- detailed architecture of any wireless communication system.

14

System-Chips

- A “system” consists of parts
 - that are designed independently, and often without knowledge of their eventual application(s)
- System-on-Chip
 - a system built from “pre-designed” parts
 - enormous challenges since “pre-designed” silicon parts do not compose well across multiple design data representation
 - testing methods and ensuring testability is even harder
- SOCs in networking and wireless applications
 - face unique design and design technology challenges due to component heterogeneity.

15

Design Technology Challenges in Networked SOCs

- Inferior CMOS components compared to discrete counterparts using bipolar and GaAs technologies
- Power, size, bandwidth limitations for on-chip processing

● An extremely tight control of chip, package parasitic RF paths is needed for on-chip RF transceivers

- And yet the system-level performance can be higher due to
 - architectural design that is less sensitive to device/technology limitations/variations
 - the ability to integrate passive devices, sophisticated signal processing and even digital computations to adapt to application, environment and even device/technology characteristics.

⇒ This requires ability to carry out rapid architectural and design space explorations.

16

How will we design these system-chips?

● There are two distinct views of NSOC:

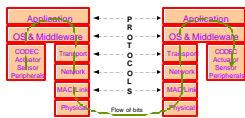
■ **Compositional or ASIC view**

- ◆ SOC design is a ultimately an integrated circuit design
- ◆ demands from mother-nature must be met.



■ **Network centric view**

◆ **Protocol and communication functions are central to chip functionality**



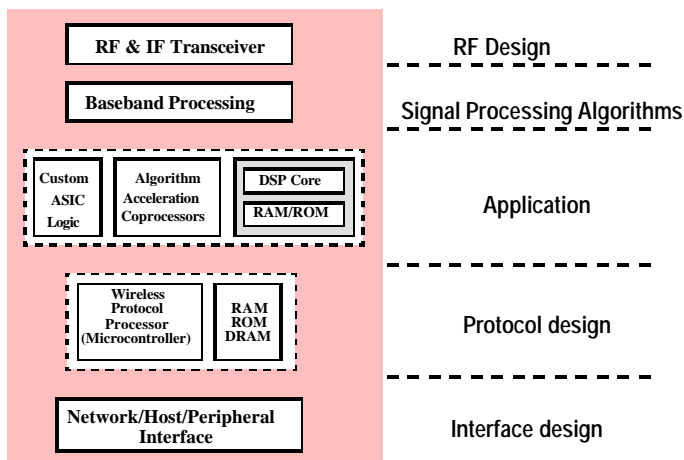
- "The really hard part is figuring out how to relate sub-system performance enhancements to end-user performance."
- "I find the hardest part to be making trade-offs so as to optimize across the various layers (physical, link, network, transport, application) of the communication system. We need tools and techniques to co-design these layers, instead of separate black-box optimizations."

● Regardless of the view, one fact is abundantly clear that:

■ **IC Designer is also a networked systems designer.**

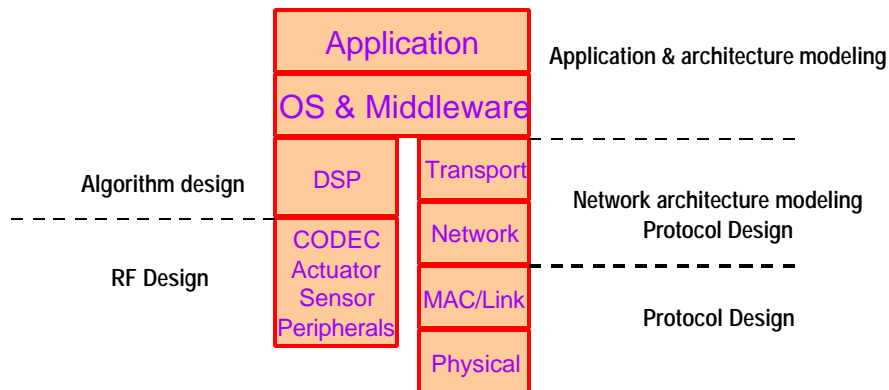
17

Compositional View: ASIC



18

Network Systems View



19

ASIC & Network Models

- Complementary models
 - ASIC models focus on “node” implementation
 - Network model keeps “multi-node” system view
- Example: Synopsys Protocol Compiler, NS models.
- “Theoretically” both models can support either view
- Designers often need the ability
 - to tradeoff across layers (easier in ASIC models) while
 - keeping the system view (easier in network models).
- Hence, a convergence in works on integration of ASIC and Network models
 - MIL3 OPNET, Cadence Bones, Diablo
 - HP EEsof’s ADS, AnSoft HFSS, Cadence Allegro, Anadigics, White Eagle DSP, ...

20

Scope of NSOC Design Tools

- Design of single-chip systems with radio transceivers requires tools
 - to explore new architectures containing heterogeneous elements
 - to explore circuit design containing analog/digital, active/passive components (mixed signal design)
 - to accurately estimate parasitic effects, package effects
- Typically mixed-system design entails
 - antennae design
 - network design: interference, user mobility, access to shared resources
 - algorithmic simulations
 - protocol design
 - circuit design, layout and estimation tools

21

Categories of Design Tools

- Architectural design tools
 - network, protocol simulations
 - algorithmic simulations, partitioning and mapping tools
- Design environment tools
 - encapsulated libraries, library management for design components
- Module design
 - low noise integrated frequency synthesizers
 - base-band over-sampled data converters
 - design of RF, analog, digital VLSI modules
- Modeling, characterization and validation tools
 - characterization of mixed-mode designs, RF coupling paths, EMI
 - simultaneous modeling, design and optimization of antenna, passive RF filter, RF amp, RF receiver, power amp. components

22

Network Architectural Design

or “behavioral design” for wireless systems

- Design network architecture
 - point-to-point, cellular, etc
- Design protocols
 - specification
 - verification at various levels: link, MAC, physical
- Tools in this category
 - Matlab, Ptolemy (and likes)
 - network, protocol simulators
- Tools are designed for simulations specific to a design layer:
 - simulation tools for algorithm development
 - simulation tools for network protocols
 - simulation tools for circuit design, hardware implementation, etc.

23

Network Architecture Modeling: NS

- Developed under the Virtual Internet Testbed (VINT) project (UCB, LBL, USC/ISI, Xerox PARC)
- Captures network nodes, topology and provides efficient event driven simulations with a number of “schedulers”
- Interpreted interface for
 - network configuration, simulation setup
 - using existing simulation kernel objects such as predefined network links
- Simulation model in C++ for
 - packet processing
 - changing models of existing simulation kernel classes, e.g., using a special queuing discipline.

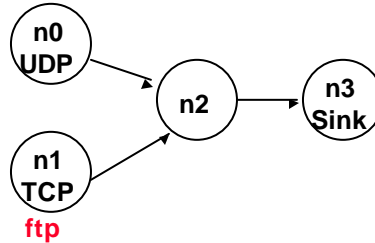
24

Example:

A 4-node system with 2 “agents”, a traffic generator

- “Agents” are network endpoints where network-layer packets are constructed or consumed.

```
set ns [new Simulator]
set f [open out.tr w]
$ns trace-all $f
set n0 {$ns node}
set n1 {$ns node}
set n2 {$ns node}
set n3 {$ns node}
$ns duplex-link $no $n2 5Mb 2ms DropTail
$ns duplex-link $n1 $n2 5Mb 2ms DropTail
$ns duplex-link $n2 $n3 1.5Mb 10ms DropTail
set udp0 [newagent/UDP]
$ns attach-agent $no $udp0
set cbr0 [newapplication/Traffic/CBR]
$scr0 attach-agent $udp0
..
$ns at 3.0 “finish”
proc finish () {
    ...
}
$ns run
```



25

NS v2 Implementation and Use

- A “Split-level” simulator consisting of
 - C++ compiled simulation engine
 - Object Tcl (Otc) interpreted front end
- Two class hierarchies (compiled, interpreted) with 1-1 correspondence between the classes
 - C++ compiled class hierarchy
 - ◆ allows detailed simulations of protocols that need use of a complete systems programming language to efficiently manipulate bytes, packet headers, algorithms over large and complex data types
 - ◆ runtime simulation speed
 - Otc interpreted class hierarchy
 - ◆ to manage multiple simulation “splits”
 - ◆ important to be able to change the model and rerun
- NS pulls off this trick by providing *tclclass* that provides access to objects in both hierarchies.

26

NS Implementation

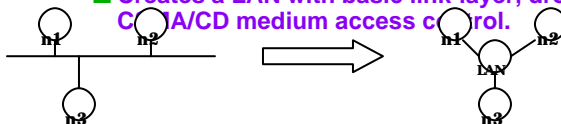
- **Example:**
 - Otcl objects that assemble, delay, queue.
 - Most routing is done in Otcl
 - HTTP simulations with flow started in Otcl but packet processing is done in C++
- **Passing results to and from the interpreter**
 - The interpreter after invoking C++ expects results back in a private variable *tcl_>result*
 - When C++ invokes Otcl the interpreter returns the result in *tcl_>result*
- **Building simulation**
 - Tclclass provides simulator with scripts to create an instance of this class and calling methods to create nodes, topologies etc.
 - Results in an event-driven simulator with 4 separate schedulers: FIFO (list); heap; calendar queue; real-time.
 - Single threaded, no event preemption.

27

NS Usage: LAN nodes

- LAN and wireless links are inherently different from PTP links due to sharing and contention properties of LANs
 - a network consisting of PTP links alone can not capture LAN contention properties
 - a special node is provided to specify LANs
- LanNode captures functionality of three lowest layers in the protocol stack, namely: link, MAC and physical layers.
 - Specifies objects to be created for LL, INTF, MAC and Physical channels.
 - **Example:**

```
$ns make-lan <nodelist> <bw> <delay> <LL> <ifq> <MAC> <channel> <phy>
$ns make-lan "$n1 $n2" $bw $delay LL queue/DropTail Mac/CSMA/CD.
```
 - Creates a LAN with basic link-layer, drop-tail queue and CSMA/CD medium access control.

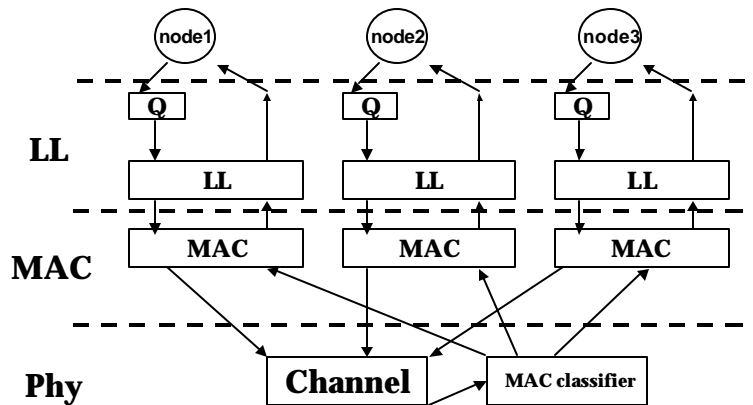


The LAN node collects all the objects shared on the LAN.

28

Network Stack simulation for LAN nodes in ns

Objects used in LAN nodes. Each of the underlying classes can be specialized for a given simulation.



Channel object simulates the shared medium and supports the medium access mechanisms of the MAC objects on the sending side.

On the receiving side, MAC classifier is responsible for delivering and optionally replicating packets to the receiving MAC objects.

29

Modeling of Mobile Nodes

From CMU Monarch Group

- Allows simulation of multihop ad hoc networks, wireless LANs etc.
- Basic model is a *MobileNode*, a split object specialized from ns class *Node*
 - allows creation of the network stack to allow channel access in *MobileNode*
- A mobile node is not connected through “Links” to other nodes
- Instead, a *MobileNode* includes the following mobility features
 - node movement (two dimensional only)
 - periodic position updates
 - maintaining topology boundary

30

Mobile Nodes

- As in wireline, the network “plumbing” is scripted in Otc1
 - Four different routing protocols (or routing agents) are available
 - destination sequence distance vector (DSDV)
 - dynamic source routing (DSR)
 - Temporally ordered routing algorithm (TORA)
 - Adhoc on-demand distance vector (AODV)
 - A mobile node creation results in
 - a mobile node with a specified routing agent, and
 - creation of a network stack consisting of
 - ◆ LL (with ARP), INT Q, MAC, Network Interface with an antenna.
- ➔ Enables integrated event driven simulation of mixed networks.

31

Mobile Node

```
Node/MobileNode instproc add-interface {channel pmodel lltype mactype qtype qlen iftype anttype } {

$self instvar arptable_ nifs_
$self instvar netif_ mac_ ifq_ ll_
set t $nifs_

set netif_($t) [new $iftype]           ;# net-interface
set mac_($t) [new $mactype]           ;# mac layer
set ifq_($t) [new $qtype]             ;# interface queue
set ll_($t) [new $lltype]             ;# link layer
set ant_($t) [new $anttype]
..
}

set topo [topography]
$topo bind_flatgrid $opt(x) $opt(y)
$node set x_ <x1>
$node set y_ <y1>
..
$ns at $time $node setdest <x2> <y2> <speed>
or
$mobilenode start
```

32

Network Simulation using OPNET

- Commercially available from MIL3
- Heterogenous models
 - for network
 - for node
 - for process
- Network, node, process editors
- Network models consist of node and link objects
- Nodes represent hardware, software subsystems
 - processors, queues, traffic generators, RX, TX
- Process models represent protocols, algorithms etc
 - using state-transition diagrams
- Simulation outputs typically include
 - discrete event simulations, traces, first and second order statistics
 - presented as time-series plots, histograms, prob. density, scattergrams etc.

33

OPNET Wireless System Modeling

- OPNET modeler with radio links and mobile nodes
- Mobile nodes include three-dimensional position attributes that can change dynamically as the simulation progresses.
- Node motion can be scripted (position history) or by a position control process.
- Links modeled using a 13-stage model where each stage is a function (in C)
- Transmitter stages:
 - Transmission delay model: time required for transmission
 - Link closure model: determine reachable receivers
 - Channel match model: determine which RX channel can demodulate the signal (rest treat it as noise)
 - Transmitter antenna gain: computes gain of TX antenna in the direction of the receiver
 - Propagation delay model: time for propagation from TX to RX.

34

Link Model Stages

- **Receiver stages:**
 - **RX antenna gain:** in the direction of the receiver
 - **Received power model:** avg. received power
 - **Background Noise Model:** computes the in-band background noise for a receiver channel
 - **Interference noise model:** typically total power of all concurrent in-band transmission
 - **SNR model:** SNR of transmission fragment based on the ratio of received power and interference noise
 - **BER model:** computes mean BER over each constant SNR fragment of the transmission
 - **Error Allocation Model:** determines the number of bit error in each fragment of the transmission
 - **Error Correction Model:** determines whether the allocated transmission errors can be corrected and if the transmitted data should be forwarded in the node for higher level processing.

35

Communications Toolbox (MATLAB)

- **Part of the MATLAB DSP workshop suite**
 - **functionality models from MATLAB**
 - ◆ sources, sinks and error analysis
 - ◆ coding, modulation, multiple access blocks, etc.
 - **communication link models from SIMULINK**
 - ◆ channel models: Rayleigh, Rician fading, noise models
- **Good front-end simulations through vector processing**
 - handles data at different time-points in large vectors
 - used in modeling physical layer component such as modems
 - useful in algorithm development and performance analysis
 - ◆ for modulation, coding, synchronization, equalization, filter design.

<http://www.mathworks.com/products/communications/index.shtml>

36

Integrated Models

- Package boundary is enlarging
 - analog/RF, digital baseband, applications, RTOS, DSP, ...
- Hardware-type 'behavioral' modeling just does not cut it
 - Substantial networking, communications, infrastructure software needs to be modeled as well.
- Learning from practice
 - People generally use C or C++ to model at system Level
 - Typically performance model and ISA models are built with C/C++
- Why not 'standardize' use of C++ for system modeling purposes?
 - We already do software, network modeling.

37

Enter SystemC

- SystemC developed by Synopsys, Coware
 - Initially Scenic project (Synopsys and UC Irvine)
 - SystemC-0.9 (Sept 1999) based on Scenic
 - SystemC-1.0 (Early 2000) performance enhancements
 - SystemC-2.0 (mid 2001) – ideas from SpecC (UC Irvine) incorporated
 - SystemC-3.0 (yet to be released) – software APIs
- Other players that influenced SystemC
 - OCAPI library (IMEC Belgium)
 - Cynlib (FORTE Design Systems, formerly CynApps)
 - SpecC
 - SuperLOG (now SystemVerilog) from Coware (now Synopsys)

38

What Is SystemC?

- A C++ library that helps designers to use C++ to model/specify synchronous digital hardware
 - **Reactivity: waiting and watching**
 - **Data types**
- Built in simulation libraries (simulation kernel) that can be used to run a SystemC program
- Any C++ compiler can compile SystemC
 - **Simulation is free in comparison to Verilog/VHDL**
- A compiler that translates the “synthesis subset” of SystemC into a netlist (Synopsys, FORTE)
- Language definition is publicly available
 - **(Open SystemC Initiative or OSCI)**
- Libraries are freely distributed
- Compiler is an expensive commercial product

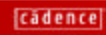
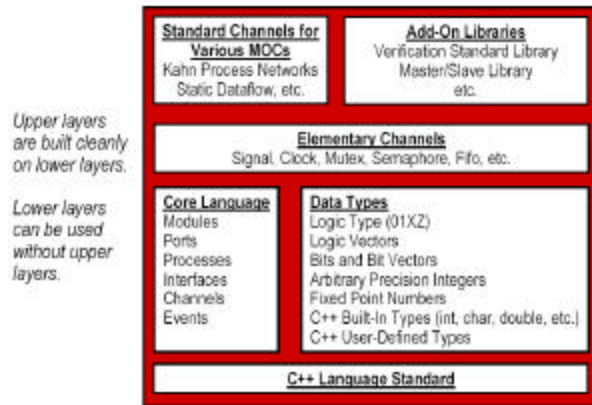
39

Quick Overview

- A SystemC program consists of **module** definitions plus a top-level function that starts the simulation
- Modules contain **processes** (C++ methods) and instances of other modules
- Ports on modules define their interface
 - **Rich set of port data types (hardware modeling, etc.)**
- **Signals** in modules convey information between instances
- **Clocks** are special signals that run periodically and can trigger **clocked processes**
- Rich set of numeric types (fixed and arbitrary precision numbers)

40

SystemC Architecture



Source: Stuart Swan, Cadence/OSCI

41

Modules and Ports

Modules:

- Hierarchical entity
- Similar to Verilog's module
- Actually a C++ class definition
- Simulation involves
 - Creating objects of this class
 - They connect themselves together
 - Processes in these objects (methods) are called by the scheduler (simulation kernel) to perform the simulation

Ports:

- Define the interface to each module
- Entities through which data is communicated
- Port consists of a direction
 - input sc_in
 - output sc_out
 - bidirectional sc_inout

- and any C++ or SystemC type

Signals:

- Information transfer within a module.

```
SC_MODULE(mymod) {
    /* port definitions */
    /* signal definitions */
    /* clock definitions */

    /* storage and state variables */

    /* process definitions */

    SC_CTOR(mymod) {
        /* Instances of processes and modules */
    }
};
```

```
SC_MODULE(mymod) {
    /* port definitions */
    sc_signal<sc_uint<32>> s1, s2;
    sc_signal<bool> reset;

    /* ... */
    SC_CTOR(mymod) {
        /* Instances of modules that connect
        to the signals */
    }
};
```

42

Processes

- Procedural code with the ability to **suspend** and **resume**
- **Methods of each module class**
- Three types:
 - **METHOD**
 - ◆ Usually Models combinational logic
 - ◆ Triggered in response to changes on inputs
 - **THREAD**
 - ◆ Usually Models testbenches
 - **CTHREAD**
 - ◆ Usually Models synchronous FSMs

43

METHOD Processes

```
SC_MODULE(onemethod) {  
    sc_in<bool> in;  
    sc_out<bool> out;  
  
    void inverter();  
  
    SC_CTOR(onemethod) {  
        SC_METHOD(inverter);  
        sensitive(in);  
    }  
};
```

Process is simply a method of this class

Instance of this process created

and made sensitive to an input

44

METHOD Processes

- Invoked once every time input “in” changes
- Runs to completion: should not contain infinite loops
 - No mechanism for being preempted

```
void onemethod::inverter() {  
    bool internal;           ← Read a value from the port  
    internal = in;          ← Write a value to an  
    out = ~internal;        output port  
}
```

45

THREAD Processes

- Triggered in response to changes on inputs
- Can suspend itself and be reactivated
 - Method calls **wait** to relinquish control
 - Scheduler runs it again later
- Designed to model just about anything

46

THREAD Processes

```
SC_MODULE(onemethod) {
  sc_in<bool> in;
  sc_out<bool> out;

  void toggler();

  SC_CTOR(onemethod) {
    SC_THREAD(toggler);
    sensitive << in;
  }
};
```

Process is simply a method of this class

Instance of this process created

alternate sensitivity list notation

47

THREAD Processes

- Reawakened whenever an input changes
- State saved between invocations
- Infinite loops should contain a wait()

```
void onemethod::toggler() {
  bool last = false;
  for (;;) {
    last = in; out = last; wait();
    last = ~in; out = last; wait();
  }
}
```

Relinquish control until the next change of a signal on the sensitivity list for this process

48

CTHREAD Processes

- Triggered in response to a single clock edge
- Can suspend itself and be reactivated
 - Method calls wait to relinquish control
 - Scheduler runs it again later
- Designed to model clocked digital hardware

49

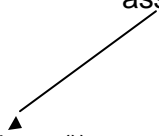
CTHREAD Processes

```
SC_MODULE(onemethod) {
    sc_in_clk clock;
    sc_in<bool> trigger, in;
    sc_out<bool> out;

    void toggler();

    SC_CTOR(onemethod) {
        SC_CTHREAD(toggler, clock.pos());
    }
};
```

Instance of this
process created and
relevant clock edge
assigned



50

SystemC Built-in Types

- `sc_bit, sc_logic`
 - Two- and four-valued single bit
- `sc_int, sc_uint`
 - 1 to 64-bit signed and unsigned integers
- `sc_bigint, sc_biguint`
 - arbitrary (fixed) width signed and unsigned integers
- `sc_bv, sc_lv`
 - arbitrary width two- and four-valued vectors
- `sc_fixed, sc_ufixed`
 - signed and unsigned fixed point numbers

51

SystemC Semantics

- Cycle-based simulation semantics
- Resembles Verilog, but does not allow the modeling of delays
- Designed to simulate quickly and resemble most synchronous digital logic

52

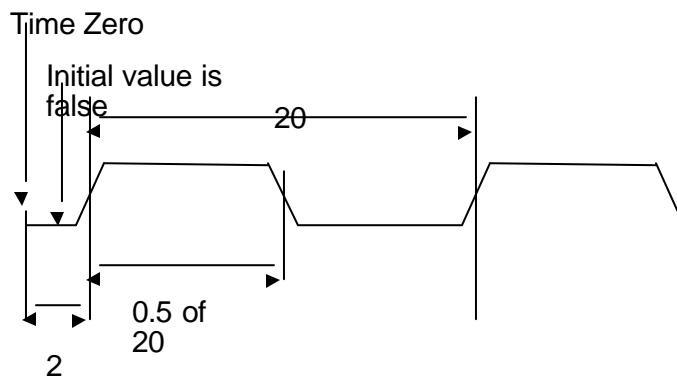
Clocks

- The only thing in SystemC that has a notion of real time
- Triggers SC_CTHREAD processes
 - or others if they decided to become sensitive to clocks

53

Clocks

- `sc_clock clock1("myclock", 20, 0.5, 2, false);`



54

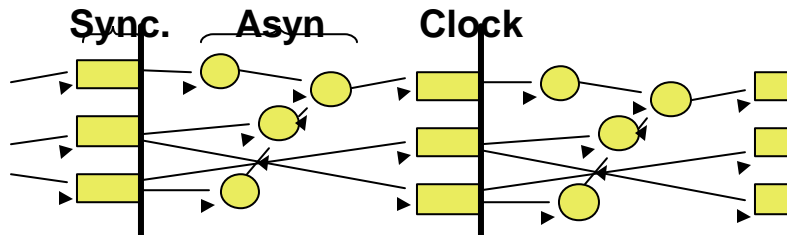
SystemC 1.0 Scheduler

- Assign clocks new values
- Repeat until stable
 - Update the outputs of triggered SC_CTHREAD processes
 - Run all SC_METHOD and SC_THREAD processes whose inputs have changed
- Execute all triggered SC_CTHREAD methods. Their outputs are saved until next time

55

Scheduling

- Clock updates outputs of SC_CTHREADs
- SC_METHODs and SC_THREADS respond to this change and settle down
- Bodies of SC_CTHREADs compute the next state



56

Recap: SC can connect to 'anything'

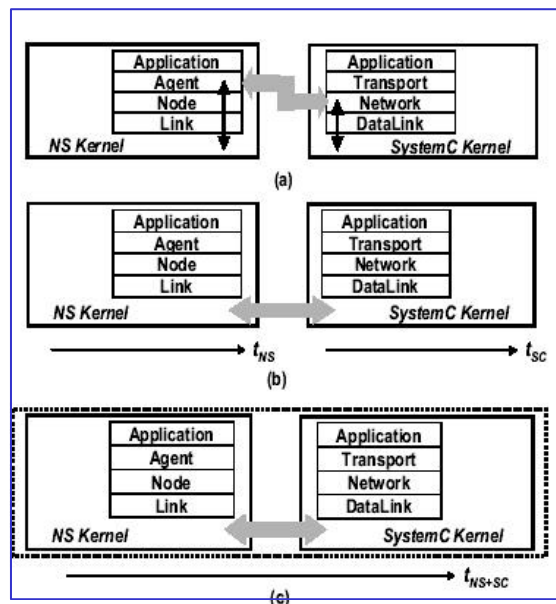
- **SC_METHOD**
 - Designed for modeling purely functional behavior
 - Sensitive to changes on inputs
 - Does not save state between invocations
- **SC_THREAD**
 - Designed to model anything
 - Sensitive to changes
 - May save variable, control state between invocations
- **SC_CTHREAD**
 - Models clocked digital logic
 - Sensitive to clock edges
 - May save variable, control state between invocations

57

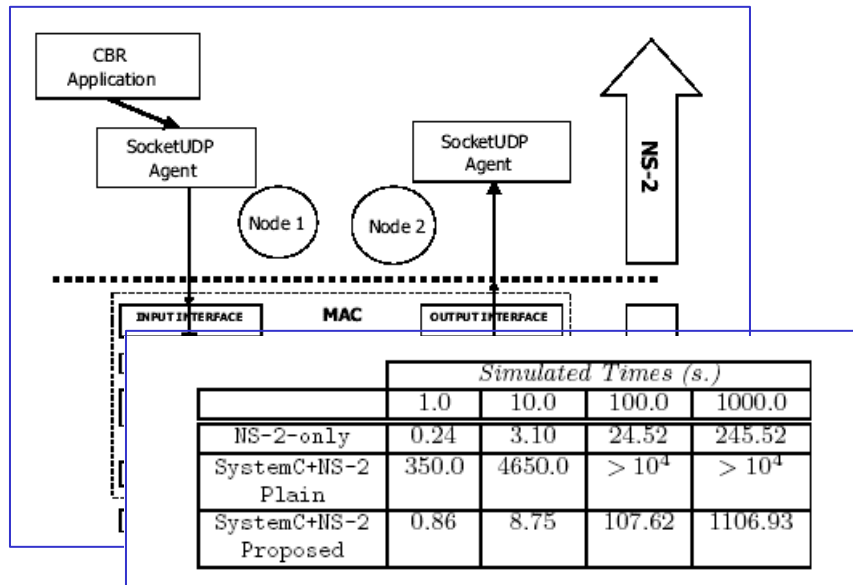
SystemC and NS-2

Fummi *et al* in DAC 2003

- Integration possible because of DE MOC used in both
 - Different notion of events and event handling
- Various levels of simulator integration
 - Packet level to integrated kernels
- Time Synchronization is usually the toughest problem
 - Various co-simulation approaches

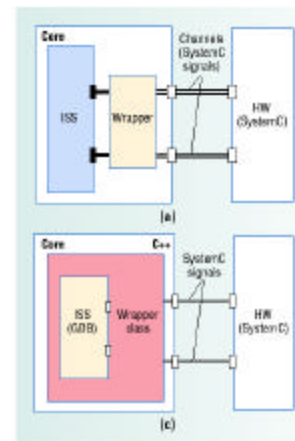
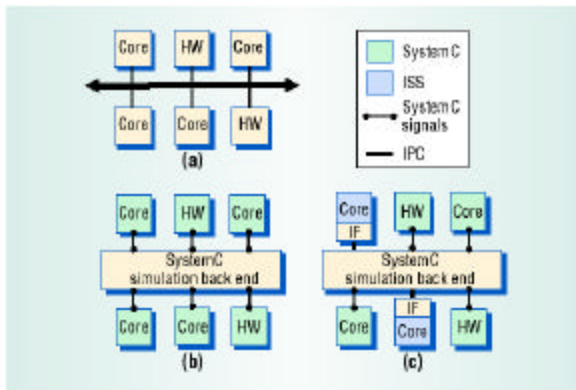


Experiment: 802.11 +MAC in SC



Complete Models Through Integration With ISS

- Too frequent communication with ISS can slow down the system simulation (usually through IPC)
- ISS 'wrapper' can be SystemC (interface) modules (IF)



Source: Benini, Drago, Fummi, Computer-03

“Platform-based” Simulations

- Multiple simulators running simultaneously
 - Many attempts are reducing need for coordination and improving simulation efficiency
 - ◆ E.g., using bus functional model to advance internal progress
 - Still maintain some level of “cycle accuracy”
 - A similar reasoning on “bit true” accuracy
 - ↳ (movement toward TLM which basically use function calls as component interfaces, instead of signals)
 - Consider, for instance, integration with processor ISS models
 - ◆ Can be “linked” through IPC, remote activation, shared memory accesses
 - ◆ Can be “embedded” through class extensions
 - Often need a greater access to simulator source
 - Currently about 100K cycles/sec speed on HP platforms

61

Integrating ISS: How C++ Can Help?

- A close linking possible by embedded ISS within SystemC simulator as a C++ cloass
- Use a ‘wrapper’ module that instantiates ‘core’ object, launches its simulation, and ‘traps’ shared accesses
- Example:
 - Wrapper instances CPU object
 - Starts ISS simulation through SC_THREAD
 - ◆ Run method continues until a memory access
 - That triggeres Bus Interface process in SystemC
 - Generating cycle accurate bus signals outside the CPU wrapper

62

Example

```
#include <systemc.h>
#include "CPU.H"

SC_MODULE(WRAPPER){
  SC_in .....
  SC_out ...
  SC_inout ...
  CPU cpu; // ISS Class allocation
  void Start_Simulation();
  void Bus_Interface_Out();
  void Bus_Interface_In();

  SC_CTOR(WRAPPER) {
    SC_THREAD(Start_Simulation);
    SC_THREAD(Bus_Interface_In, clock_pos());
    sensitive_pos << clock;
    SC_METHOD(Bus_Interface_Out);
    sensitive <<cpu.mem_access;
  }
}
```

```
#include <systemc.h>
#include "CPU.H"

CPU::CPU(...) {
  // Initialize simulation
}

CPU::Run() {
  ...
  ...
}

uint Memory_Read(uint add)
{
  ...
  address = add;
  mem_access.write(true);
  while (return_from_mem_access == 0)
    wait();
  ...
}
```

```
class CPU {
  .....
public:
  CPU(..) // Constructor

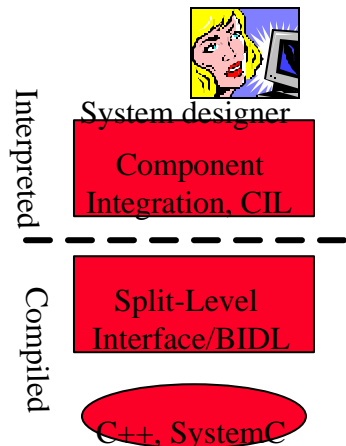
  SC_SIGNAL<bool> return_from_mem_access;
  SC_SIGNAL<bool> mem_access;
  uint address;
  uint data;
  .....
}
```

Source: Benini, Drago, Fummi, Computer 03₆₃

Going Forward: CCF

- **Component Composition Frameworks**
- **A composition environment**
 - Built upon existing class libraries, to add a software layer for manipulation and configuration of C++ IP models
 - Ease software module connectivity
 - Run-time environment structure
- **SW Architecture That Enables**
 - composition of structural and functional info.
 - Notions of component substitutability based on type theory extensions.

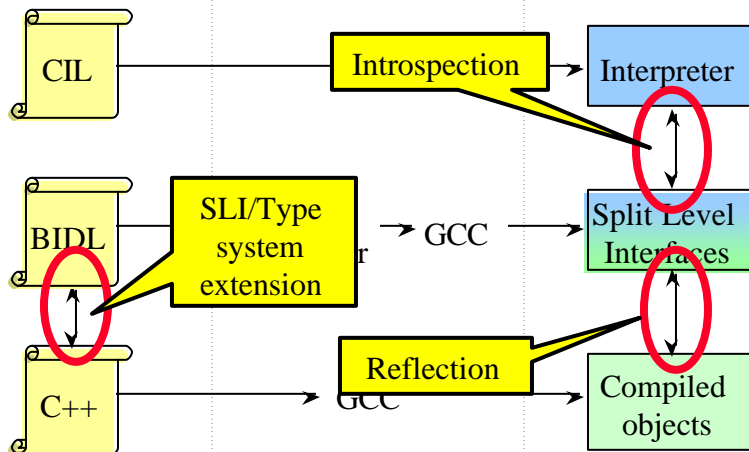
→ modeling of the application and the platform



<http://mesl.ucsd.edu>

Language and Run-time Layers

Language Tools Run-time structure



65

Summary

- Design tools offerings for on-chip networked and wireless systems are an area of growing importance due to inherent complexity of on-chip design and multi-level tradeoffs
- Wireless system design tools are strongly influenced by the layer at which the design is being done
- System modeling tools are quite common and advanced
- Design environments, circuit design tools lag significantly behind the design practice
- Current efforts at “integrated” design (exploration) environments.

66

References

- Models of computation:
 - <http://ptolemy.eecs.berkeley.edu>
- Language and methodology :
 - SpecC: <http://www.ics.uci.edu/~specc>
 - OCAPI: <http://www.imec.be/ocapi>
- Languages :
 - SystemC : <http://www.systemc.org>
 - CynApps: <http://www.cynapps.com>
 - CoWare: <http://www.coware.com>
 - Objective VHDL:
- Language semantics:
 - R. Gupta and S. Liao, Using a programming language for digital system design, IEEE D&T April-June 97
- Interface based design:
 - Alberto DAC97 Paper
- VSIA system level design workgroup: <http://www.vsi.org>
- System level issue: Gajski's silicon compilation and blue books
- Software design pattern book
- Architecture description language: www.ics.uci.edu/~aces/expression

67

Acknowledgements

- Sandeep Shukla, Virginia Tech
- Mani Srivastava, UCLA

68