Exploring SoC Communication Architectures for Performance and Power

Nikil Dutt
ACES Laboratory
Center for Embedded Computer Systems
Donald Bren School of Information and Computer Sciences
University of California, Irvine

dutt@uci.edu
http://www.ics.uci.edu/~aces
Outline

- Motivation
- CA Exploration at Transaction Level
- Floorplan-aware Bus Architecture Synthesis Approach
- SoC Power/Energy Modeling
- Design Drivers
- Summary
SoC Design Complexity vs. Productivity

SoC designs today are complex, characterized by more and more IPs being integrated on a single chip, and a shrinking time-to-market.
Strategies to handle SoC complexity

◆ IP based design and reuse
  ■ design IPs to be reused in multiple designs
  ■ requires initial investment to create reusable cores; but productivity in subsequent designs can be substantially enhanced with reuse
  ■ e.g. VSIA and OCP-IP core interface standards

◆ Raising modeling abstraction
  ■ simulating design at RTL level for verification or exploration is just not practical anymore
  ■ capturing the system (hardware and software) at a higher level of abstraction is better
    ➢ faster to model
    ➢ quicker to simulate
    ➢ early design visibility reduces time-to-market
  ■ models are typically captured in C/C++/SystemC
Ideal Platform based SoC Design Flow

1. Algorithm selection
2. Optimization
3. HW/SW partitioning
4. Behavior mapping
5. Architecture exploration
6. CA selection/exploration
7. Protocol generation
8. Topology synthesis
9. Interface synthesis
10. Cycle scheduling
11. Implementation model
12. Logic synthesis and physical implementation
Data Flow Replacing Data Processing As Major SoC Design Challenge

- Exploding core counts requiring more advanced Interconnects
- EDA cannot solve this architectural problem easily
- Complexity too high to hand craft (and verify!)

Communication Architecture Design and Verification becoming Highest Priority in Contemporary SoC Design!
Need for Communication-centric Design Flow

Communication architecture consumes upto 50% of total on-chip power!

communication is THE most critical aspect affecting system performance

Communication Architectures in today’s complex systems significantly affect performance, power, cost and time-to-market!

ever increasing number of wires, repeaters, bus components (arbiters, bridges, decoders etc.) increases system cost
Evolution of On-chip Communication Architectures

- **Custom**: 1990
- **Shared Bus**: 1995
- **Hierarchical Bus**: 2000
- **Bus Matrix**: 2005
- **Network-on-chips?**: 2010
Evolution of On-chip Communication Architectures

Focus of this talk!
SoC Bus based Communication Architectures

a) single bus

b) hierarchical bus

c) multiple bus

d) split-bus

e) point-to-point bus

f) bus matrix
Bus Terminology

- **Master (or Initiator)**
  - IP component that initiates a read or write data transfer

- **Slave (or Target)**
  - IP component that does not initiate transfers and only responds to incoming transfer requests

- **Arbiter**
  - Controls access to the shared bus
  - Uses arbitration scheme to select master to grant access to bus

- **Decoder**
  - Determines which component a transfer is intended for

- **Bridge**
  - Connects two busses
  - Acts as *slave* on one side and *master* on the other
Modern SoC Design Flow

- Product requirements from customer
  - Specification Model
    - Algorithm selection optimization
    - Allocation
      - Behavior partitioning
      - Scheduling
    - Protocol selection
      - Channel partitioning
      - Arbitration
    - Cycle scheduling
      - Protocol scheduling
  - Architecture Model
  - Communication Model
  - Implementation Model

Copyright © 2006 UCI ACES Laboratory  http://www.cecs.uci.edu/~aces
Bus-based Communication Architectures

- Several bus based CA commonly used in SoC designs
  - AMBA
  - Wishbone
  - CoreConnect
  - PowerPC Bus

- Key Features
  - High Performance System Bus
    - processors, memory, DMA etc.
  - Low Bandwidth Peripheral Bus
    - timer, interrupt controller, UART etc.
Outline

◆ Motivation

◆ CA Exploration at Transaction Level

◆ Floorplan-aware Bus Architecture Synthesis Approach

◆ SoC Power/Energy Modeling

◆ Design Drivers

◆ Summary
Issues

- Selecting and configuring bus-based CA for optimal performance is a critical activity in a SoC design, requiring **CA exploration**
  
  - **bus architecture**
    (e.g. PPC Bus, AMBA, CoreConnect)
  
  - **architecture parameters**
    (e.g. bus width, burst size)
  
  - **bus topologies**
    (e.g. shared, hierarchical)
  
  - **protocol choices**
    (e.g. arbitration strategies)
## Bus Exploration at what Abstraction?

<table>
<thead>
<tr>
<th>Cycle Rate (Hz)</th>
<th>Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^8$</td>
<td>Silicon Reference Design</td>
</tr>
<tr>
<td>$10^6$</td>
<td>HW Emulator</td>
</tr>
<tr>
<td>$10^5$</td>
<td>Transaction Model</td>
</tr>
<tr>
<td>$10^4$</td>
<td>Cycle Accurate Model</td>
</tr>
<tr>
<td>$10^2$</td>
<td>RTL Model</td>
</tr>
<tr>
<td>10</td>
<td>Gate Level Model</td>
</tr>
</tbody>
</table>

- Capturing a SoC design at RTL level and then simulating for communication space exploration is
  - too slow (~10–100 cycles/s)
  - cumbersome to capture all the detail
  - too late in the design flow for exploration!
Communication Space Exploration
Abstraction Levels

Algorithm

TLM

T-BCA

PA-BCA

CA

Register Transfer Level
Communication Space Exploration
Abstraction Levels

Algorithm

TLM

T-BCA

PA-BCA

CA

Register Transfer Level
Existing Abstractions for Exploration above RTL: Cycle Accurate (CA) Models

- Detailed system debug and analysis
- Time consuming to model - /1 to /3 RTL
- Too slow for exploring SoC designs - 100x RTL

Algorithm
- TLM
- T-BCA
- PA-BCA
- CA

Register Transfer Level
Existing Abstractions for Exploration above RTL: Pin-accurate Bus Cycle Accurate (PA-BCA) Models

- High level system exploration
- Still time consuming to model - /5 to /10 RTL
- Still slow for exploring SoC designs - 100x to 500x RTL
Existing Abstractions for Exploration above RTL: Transaction Level Models (TLM)

- **High level system validation and embedded software development**
  - *Fast to model*
    - / 10 to / 50 RTL
  - *Fast simulation speed, but model not too detailed for exploring SoC designs*
    - >>1000x RTL

```plaintext
var1 = a + b;
d = d << var1;
request(port1);
e = REG4 | 0xff
wait();
...
```
Existing Abstractions for Exploration above RTL: Transaction-based BCA (T-BCA) Models

- Uses Transaction Level Modeling (TLM) techniques to speed up BCA model simulation
- Time to model varies
- Simulation speed generally faster than PA-BCA
Previous work in T-BCA Modeling

- Xinping et al. (ICCAD 2002) use function calls instead of slower signal semantics to model AMBA2 and CoreConnect
  - resulting models not detailed enough for accurate CA exploration

- Caldari et al. (DATE 2003) similarly model AMBA2 using function calls for reads/writes
  - Bus signals are also modeled: slows simulation
  - Clocked threads used extensively: slows simulation

- Ogawa et al. (DATE 2003) also model data transfers in AMBA2 using read/write transactions
  - use low level handshaking semantics

- In mid 2003, ARM released the AHB Cycle-Level Interface Specification
  - for modeling AMBA AHB at CA level in SystemC
  - function calls emulate bus signals at interface
  - Scope for improving speed by reducing number of calls
CCATB Modeling Abstraction (DAC-2004)

- **CCATB**: Cycle Count Accurate at Transaction Boundaries
  - Observe signals at transaction boundaries
  - BUT... maintain overall cycle accuracy
    - essential for system exploration

- Variant of T-BCA Models
  - no pins at IP interface
  - extension of read(), write() transaction interface from TLM
  - IPs modeled at behavioral level
  - protocol details (e.g. burst size, cache hints) need to be passed

- Modeling Language – SystemC
  - fast (C/C++ native execution)
  - provides constructs (concurrency, timing) for hardware modeling
  - extensive commercial tool support (debugging, waveform viewing)

- Trades off intra transaction visibility for simulation speed
  - more than 2x faster than fastest BCA models
wait (REQ + ARB + SLV + BURST_LEN + PPL) = (1 + 1 + 2 + 4 + 1) = 9 cycles

control for burst INCR4
Timing Diagram

CCATB: Observe signals at transaction boundaries only!

wait (REQ + ARB + SLV + BURST_LEN + PPL) = (1 + 1 + 2 + 4 + 1) = 9 cycles
Delays Modeled

Slave delay

Communication delay

Arbitration delay

AMBBA 2.0 CHANNEL (Read, Write)

Master delay

Interface delay
AMBA 2.0 based multimedia subsystem for audio and video encoding

Designer needs to add support for
- audio/video decoding
- additional AVlink interface for streaming data

Maintain bandwidth constraints for USB (480 Mbps) and AVLink interface (768 Mbps)
Extended Architecture Variation 1

<table>
<thead>
<tr>
<th>Arch</th>
<th>RR</th>
<th>TDMA1</th>
<th>TDMA2</th>
<th>SP1</th>
<th>SP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch1</td>
<td>27.24</td>
<td>24.65</td>
<td>25.06</td>
<td>25.72</td>
<td>26.49</td>
</tr>
</tbody>
</table>

Execution cycle count (in millions of cycles)
Extended Architecture Variation 2

<table>
<thead>
<tr>
<th>Arch</th>
<th>RR</th>
<th>TDMA1</th>
<th>TDMA2</th>
<th>SP1</th>
<th>SP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch1</td>
<td>27.24</td>
<td>24.65</td>
<td>25.06</td>
<td>25.72</td>
<td>26.49</td>
</tr>
<tr>
<td>Arch2</td>
<td>24.98</td>
<td>23.86</td>
<td>23.03</td>
<td>23.52</td>
<td>23.44</td>
</tr>
</tbody>
</table>

Execution cycle count (in millions of cycles)
Extended Architecture Variation 3

<table>
<thead>
<tr>
<th>Arch</th>
<th>RR</th>
<th>TDMA1</th>
<th>TDMA2</th>
<th>SP1</th>
<th>SP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch1</td>
<td>27.24</td>
<td>24.65</td>
<td>25.06</td>
<td>25.72</td>
<td>26.49</td>
</tr>
<tr>
<td>Arch2</td>
<td>24.98</td>
<td>23.86</td>
<td>23.03</td>
<td>23.52</td>
<td>23.44</td>
</tr>
<tr>
<td>Arch3</td>
<td>24.73</td>
<td>23.74</td>
<td>22.96</td>
<td>23.11</td>
<td>23.05</td>
</tr>
</tbody>
</table>

Execution cycle count (in millions of cycles)
Extended Architecture Variation 4

<table>
<thead>
<tr>
<th>Arch</th>
<th>RR</th>
<th>TDMA1</th>
<th>TDMA2</th>
<th>SP1</th>
<th>SP2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch1</td>
<td>27.24</td>
<td>24.65</td>
<td>25.06</td>
<td>25.72</td>
<td>26.49</td>
</tr>
<tr>
<td>Arch2</td>
<td>24.98</td>
<td>23.86</td>
<td>23.03</td>
<td>23.52</td>
<td>23.44</td>
</tr>
<tr>
<td>Arch3</td>
<td>24.73</td>
<td>23.74</td>
<td>22.96</td>
<td>23.11</td>
<td>23.05</td>
</tr>
<tr>
<td>Arch4</td>
<td>22.02</td>
<td>21.79</td>
<td>21.65</td>
<td>21.18</td>
<td>21.26</td>
</tr>
</tbody>
</table>

Execution cycle count (in millions of cycles)
Simulation Speed Comparison

- Goal is to compare simulation performance for
  - Pin accurate BCA (PA-BCA)
  - Transaction based BCA (T-BCA)
  - CCATB

- We were interested in exploring effect of changing system complexity on simulation speed
Example SoC Platform
Comparison Graph

![Graph showing performance comparison between CCATB, PA-BCA, and T-BCA across different masters.](image)
## Modeling Effort Comparison

<table>
<thead>
<tr>
<th>Model Abstraction</th>
<th>Average CCATB speedup (x times)</th>
<th>Modeling Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCATB</td>
<td>1</td>
<td>~3 days</td>
</tr>
<tr>
<td>T-BCA</td>
<td>1.67</td>
<td>~4 days</td>
</tr>
<tr>
<td>PA-BCA</td>
<td>2.2</td>
<td>~1.5 wks</td>
</tr>
</tbody>
</table>
CCATB Summary

- CCATB models
  - Faster to simulate than
    - PA-BCA models by 120% (average)
    - T-BCA models by 67% (average)
  - Less modeling effort compared to BCA models
    - Since intra-transaction visibility is not a concern
  - Accurate exploration of CA space
    - Performance figures comparable in accuracy to detailed pin accurate BCA models
  - Conveniently fit into SoC Design Flow
    - Easy to extend TLM level models to get CCATB models
    - Easy to refine down to pin accurate BCA level
Outline

- Motivation
- CA Exploration at Transaction Level
- Floorplan-aware Bus Architecture Synthesis Approach
- SoC Power/Energy Modeling
- Design Drivers
- Summary
Need for Physically-aware BA Synthesis

- Improving process technology has led to increasing number of cores being integrated on a single SoC
  - Tens to hundreds of cores today

- Sharp increase in overall on-chip communication
  - next generation of multimedia, broadband and networking apps
  - Communication is fast becoming a major design bottleneck!

- Standard bus architectures such as AMBA, PPC Bus and CoreConnect are popular choices for handling on-chip communication
  - Relatively simple to design
  - Low area overhead
Manual traversal of this vast exploration space not practical
But designers today still create high level simulation models and manually iterate through different design configurations!

DMA burst sizes
Bus Cycle Time Violation

To meet performance constraints, bus speed set to 333 Mhz (3 ns bus cycle time)
- excessive capacitive load on bus can increase signal propagation delay

For load capacitance $C_L = 2.936$ pF, wire length = 9.9 mm, implying delay of 3.5 ns

Such a violation has adverse effect on system

Since BA synthesis decides cumulative $C_L$ on bus, there is a need to make BA synthesis *physically aware*

- considerable manual rework of RTL
- extensive re-verification effort
Our Approach: FABSYN (DAC-2005)

♦ early BA exploration and timing violation detection / elimination
  ♦ verify feasibility of synthesized BA early in the design flow
  ♦ saves costly design iterations later

♦ increasingly important in the deep submicron era as
  ♦ clock speeds increase
  ♦ lengthy propagation delays cause timing violations
Related Work

- Automating Bus Architecture Synthesis
  - Early work (Narayan et al. [DATE ’94], Daveau et al. [TVLSI ’97], Gasteier et al. [TODAES ’99]) was aimed at
    - minimizing bus width
    - simple synchronization protocol selection
    - topology generation for simple busses without arbitration
  - Pinto et al. [DAC ’02] and Ryu et al. [DATE ’03] focused on automating bus topology synthesis
  - Lahiri et al. [ICCAD ’00] and Shin et al. [DATE ’04] synthesized bus architecture parameters

- Using High Level Floorplanner in CA Synthesis
  - Dick et al. [DATE ’99], Drinic et al. [ICCAD ’00], Hu et al. [ASPDAC ’02] for estimating wire lengths to determine energy consumption and global delays for real time constraint satisfaction
  - Bergamaschi et al. [CODES+ISSS ’03] and Thepayasuwan et al. [DATE ’04] for generating an early core placement estimate
FABSYN: Our Approach (DAC-2005)

- **FABSYN**: Floorplan Aware Bus Architecture SYNthesis

- FABSYN automates
  - bus topology synthesis, **AND**
  - bus architecture parameter generation
    - arbitration priorities
    - bus widths
    - bus speeds
    - DMA burst sizes

- Unlike previous approaches, we use a floorplanner to identify and eliminate *bus cycle time violations*
Problem Formulation

◆ Given:
  ■ SoC with performance constraints
  ■ a target bus-based communication architecture (e.g. AMBA)

◆ Assumptions:
  ■ hardware-software partitioning has been done already
  ■ IPs are standard non-modifiable “black box” components
  ■ memories can be split and modified

◆ Goals:
  ■ automatically synthesize BA topology AND parameter values
  ■ detect/eliminate BA configurations with bus cycle time violations
  ■ satisfy all throughput constraints in the design
  ■ minimize implementation cost
SoC Performance Constraints

- SoC designs have performance constraints that can be represented in terms of **Data Throughput Constraints**
- **Communication Throughput Graph, CTG = G(V,A)** incorporates SoC components and throughput constraints
- **Throughput Constraint Path (TCP)** is a CTG sub-graph
Bus Architecture Synthesis Flow

Inputs

CTG
IP library
comm arch.
constraint Set ($\Psi$)

preprocess

simple bus mapping

Select unsatisfied TCP from $\Omega$

explore_params

TCP met?

mutate_topology

Output

output synthesized communication arch

$\Omega$ still empty?

no

Run floorplanner and delay estimator

optimize_design

$\Omega$ empty?

yes

no

yes

no

Inputs

Output
preprocess
Bus Architecture Synthesis Flow

- CTG
- IP library
- comm arch.
- constraint Set ($\Psi$)

**preprocess**

**simple bus mapping**

Select unsatisfied TCP from $\Omega$

**explore_params**

TCP met?

**mutate_topology**

$\Omega$ empty?

**optimize_design**

Run floorplanner and delay estimator

output synthesized communication arch

$\Omega$ still empty?

yes

no

Copyright © 2006 UCI ACES Laboratory  http://www.cecs.uci.edu/~aces

UCSD Talk Feb 13 2006 # 49
Simple Bus Mapping

Bus mapping

CPU1

MEM1

M2

MEM2b

S2

MEM3

S1

MEM2a

S4

S3

M3

bridge

main

subsys

MEM1

M2

M3

S1

MEM2b

peripheral

S3

S2

MEM3

Copyright © 2006 UCI ACES Laboratory http://www.cecs.uci.edu/~aces

UCSD Talk Feb 13 2006 # 50
Bus Architecture Synthesis Flow

Communication Parameter Constraint Set \((\Psi)\)

- To ensure that our approach generates realistic BA
- Constraints are in the form of a **discrete set of valid values** for BA parameters to be synthesized
- Allows designer to bias the synthesis process based on knowledge of the design and technology being targeted

**Diagram:**

- **TCP met?**
  - yes → \(\Omega\) empty?
    - yes → \(\text{optimize\_design}\)
    - no → \(\text{mutate\_topology}\)
  - no → \(\text{explore\_params}\)

**Notes:**

- Communication Parameter Constraint Set \((\Psi)\)
- TCP met?
- \(\Omega\) empty?
explore_params

Set (bus speed, bus width) ≤ \( \Psi \)(max_speed, max_width)

Select unselected combination of valid arbitration priority ordering and valid DMA burst size

All valid comb covered?

Y: exit

N: Simulate design

Communication behavior is characterized by unpredictability
- Dynamic bus requests from cores
- Non-deterministic delay arbitration conflicts
- Buffer overflow delays ...

Simulation necessary for accuracy in performance estimation

We use a SystemC based fast transaction-based, bus cycle accurate modeling abstraction (Pasricha et al. [DAC ’04])
Bus Architecture Synthesis Flow

1. **preprocess**
2. **simple bus mapping**
3. **select unsatisfied TCP from \( \Omega \)**
4. **explore_params**
   - **TCP met?**
     - no: **mutate_topology**
     - yes: **optimize_design**
5. **Run floorplanner and delay estimator**
6. **output synthesized communication arch**

**Decision Points:**
- \( \Omega \) empty?
- TCP met?

**Constraint Sets:**
- CTG
- IP library
- communication arch.
- constraint set \( (\Psi) \)
mutate_topology

Create new bus and/or migrate IPs
mutate_topology

Create new bus and/or migrate IPs
Bus Architecture Synthesis Flow

1. **preprocess**
   - simple bus mapping
   - Select unsatisfied TCP from $\Omega$

2. **explore_params**
   - TCP met?
     - yes
     - no: $\Omega$ empty?

3. **mutate_topology**
   - yes
   - no

4. **optimize_design**
   - yes
   - no

5. **Run floorplanner and delay estimator**

6. **output synthesized communication arch**
   - yes
   - no: $\Omega$ still empty?

Input:
- CTG
- IP library
- comm arch.
- constraint Set ($\Psi$)
Bus Architecture Synthesis Flow

- CTG
- IP library
- comm arch.
- constraint Set (Ψ)

1. preprocess
2. simple bus mapping
3. Select unsatisfied TCP from Ω
4. explore_params
5. TCP met?
6. mutate_topology
7. optimize_design
8. Run floorplanner and delay estimator
9. output synthesized communication arch

Ω still empty?
Ω empty?

Copyright © 2006 UCI ACES Laboratory http://www.cecs.uci.edu/~aces
Bus Architecture Synthesis Flow

- **CTG**
- **IP library**
- **communication arch.**
- **constraint Set (Ψ)**

**preprocess**

**simple bus mapping**

Select unsatisfied TCP from Ω

**explore_params**

TCP met? yes no

**mutate_topology**

**optimize_design**

Run floorplanner and delay estimator

**output synthesized communication arch**

Ω still empty? yes no

Ω empty? yes no
Floorplanning and Wire Delay Estimation

- Our floorplanner is adapted from the simulated annealing based floorplanner proposed by Adya and Markov et al. [TVLSI ’03]

- The input to the floorplanner is
  - a list of components and their interconnections in the system
  - area of components
  - dimensions of components (widths/heights or aspect ratios)
  - maximum die size (optional)
  - fixed locations for hard macros (optional)

- We use the following cost function with the floorplanner:
  \[
  \text{Cost} = w_1 \cdot \text{Area} + w_2 \cdot \text{BusWL} + w_3 \cdot \text{TotalWL}
  \]

- The wire delay estimation is adapted from the models proposed by Cong and Pan [ICCAD ’01]
Bus Architecture Synthesis Flow

CTG
IP library
comm arch.
constraint Set (Ψ)

preprocess
simple bus mapping
Select unsatisfied TCP from Ω
explore_params

TCP met?
mutate_topology

mutate_topology optimize_design

ω empty?

Run floorplanner and delay estimator

output synthesized communication arch

ω still empty?

yes

no

yes

no

Copyright © 2006 UCI ACES Laboratory http://www.cecs.uci.edu/~aces UCSD Talk Feb 13 2006 # 60
Synthesized Bus Architecture

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus width</td>
<td>main1: 32, main2: 32, main3: 32, periph: 32</td>
</tr>
<tr>
<td>bus speed</td>
<td>main1: 133, main2: 133, main3: 133, periph: 66</td>
</tr>
<tr>
<td>arb priority</td>
<td>CPU1 &gt; M3 &gt; M2 (static)</td>
</tr>
</tbody>
</table>
## Case Study 1

**Communication Parameter Constraint Set** $(\Psi)$

<table>
<thead>
<tr>
<th>Set</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus width</td>
<td>8, 16, 32</td>
</tr>
<tr>
<td>bus speed</td>
<td>33, 66, 100, 133, 166, 200</td>
</tr>
<tr>
<td>DMA burst size</td>
<td>1, 2, 4, 8, 16</td>
</tr>
<tr>
<td>arbitration strategy</td>
<td>static priority</td>
</tr>
</tbody>
</table>
## Case Study 1

### Communication Parameter Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
<td>AHB1</td>
</tr>
<tr>
<td>bus width</td>
<td>32</td>
</tr>
<tr>
<td>bus speed</td>
<td>133</td>
</tr>
<tr>
<td>dma size</td>
<td>16</td>
</tr>
<tr>
<td>arb priority</td>
<td>ARM&gt;USB&gt; DMA&gt; EXT_IF&gt;ASIC1&gt;SWITCH</td>
</tr>
</tbody>
</table>
Case Study 1
Case Study 2

Communication Parameter Constraint Set ($\Psi$)

<table>
<thead>
<tr>
<th>Set</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>bus width</td>
<td>8, 16, 32, 64</td>
</tr>
<tr>
<td>bus speed</td>
<td>33, 66, 100, 133, 166, 200</td>
</tr>
<tr>
<td>DMA burst size</td>
<td>1, 2, 4, 8, 16</td>
</tr>
<tr>
<td>arbitration strategy</td>
<td>static priority</td>
</tr>
</tbody>
</table>
Case Study 2

Excessive capacitive load causes bus cycle time violation for AXI1
## Communication Parameter Values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI1</td>
<td>AXI2</td>
</tr>
<tr>
<td>bus width</td>
<td>32</td>
</tr>
<tr>
<td>bus speed</td>
<td>100</td>
</tr>
<tr>
<td>dma size</td>
<td>16</td>
</tr>
<tr>
<td>arb scheme</td>
<td>SWITCH &gt; ASIC2 &gt; ARM &gt; USB &gt; EXT_IF &gt; DMA &gt; ASIC1</td>
</tr>
</tbody>
</table>

- **AXI1** bus width: 32
- **AXI2** bus width: 32
- **AXI3** bus width: 64
- **APB1** bus width: 32
- **AXI1** bus speed: 100
- **AXI2** bus speed: 100
- **AXI3** bus speed: 200
- **APB1** bus speed: 66
- **AXI1** dma size: 16
- **AXI2** dma size: 16
- **AXI3** dma size: 16
- **APB1** dma size: 16
- **AXI1** arb scheme: SWITCH > ASIC2 > ARM > USB > EXT_IF > DMA > ASIC1
- **AXI2** arb scheme: SWITCH > ASIC2 > ARM > USB > EXT_IF > DMA > ASIC1
- **AXI3** arb scheme: SWITCH > ASIC2 > ARM > USB > EXT_IF > DMA > ASIC1
- **APB1** arb scheme: SWITCH > ASIC2 > ARM > USB > EXT_IF > DMA > ASIC1
Case Study 2
## Synthesis Result Comparison

### CaseStudy1 Designs

<table>
<thead>
<tr>
<th></th>
<th>initial</th>
<th>ABS</th>
<th>manual</th>
<th>FABSYN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Busses</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>TCP constr. satisfied</td>
<td>0/2</td>
<td>2/2, not feasible</td>
<td>2/2</td>
<td>2/2</td>
</tr>
<tr>
<td>Exec. cycles (millions)</td>
<td>49.76</td>
<td>24.51</td>
<td>18.8</td>
<td>20.32</td>
</tr>
<tr>
<td>Time to synthesize</td>
<td>~mins</td>
<td>~hours</td>
<td>~days</td>
<td>~hours</td>
</tr>
</tbody>
</table>

### CaseStudy2 Designs

<table>
<thead>
<tr>
<th></th>
<th>initial</th>
<th>ABS</th>
<th>manual</th>
<th>FABSYN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Busses</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>TCP constr. satisfied</td>
<td>0/3</td>
<td>3/3, not feasible</td>
<td>3/3</td>
<td>3/3</td>
</tr>
<tr>
<td>Exec. cycles (millions)</td>
<td>88.48</td>
<td>47.63</td>
<td>26.58</td>
<td>29.10</td>
</tr>
<tr>
<td>Time to synthesize</td>
<td>~mins</td>
<td>~hours</td>
<td>~days</td>
<td>~hours</td>
</tr>
</tbody>
</table>
FABSYN Summary

- FABSYN: Floorplan-Aware BA Synthesis
  - bus topology and bus architecture parameter synthesis
  - detect and eliminate bus cycle time violations
  - satisfy performance constraints
  - minimize implementation cost

- Results from BA synthesis for SoC case studies show usefulness of approach when compared to
  - approaches without integrated floorplanners
  - manual or semi-automated synthesis approaches

- Although experiments have been performed on AMBA BA, approach is portable to other standard BA such as PowerPC Bus and CoreConnect
Outline

◆ Motivation

◆ CA Exploration at Transaction Level

◆ Floorplan-aware Bus Architecture Synthesis Approach

◆ SoC Power/Energy Modeling

◆ Design Drivers

◆ Summary
Power/Energy Modeling

Key Objective: SOC Power Optimization Framework

- Develop *early power exploration environment* for SOC designers
- Provide meaningful power-aware exploration with estimates that combine
  - Previously characterized IP blocks
  - New/customized IP blocks
  - On-chip communication architectures
- Allow *qualitative and quantitative* comparison for power/energy of alternative SOC architectures
SoC Power: Key Challenges

- **SOC Component-level challenges**
  - Power characterization methodology
    - Accuracy
    - Variability
    - Efficiency

- **SOC-level system-level modeling challenges**
  - Interconnections/communication architectures
    - Early Analysis and Modeling (physically aware!)
  - Statistical vs. simulation tradeoffs
    - Accuracy
    - Variability
    - Efficiency

- **SOC-level system-level exploration challenges**
  - Impact of power budgeting
    - Static
    - Dynamic (power management)
  - Tradeoffs between power, performance, cost..
    - Accuracy
    - Variability
    - Efficiency
SoCPower Framework: Our Approach

- SOC-level power modeling
  - IP components
  - Interconnections/communication architecture

- Memory architecture
  - Sizing, partitioning, banking, etc.

- Hardware/software partitioning and allocation
  - ASIC, ASIP, coprocessor, DSP, etc.

- Interconnection/bus architecture exploration
  - Single, multiple, hierarchical, crossbar, etc.

- Floorplanning and Thermal Effects
  - Considering leakage power and temperature variations

- Algorithmic level tradeoffs
  - Alternative algorithmic implementations with varying power, performance, cost
SoCPower Framework

Power Modeling/Prediction Approach

- SoC Template (e.g. AMBA)
  - Provides Early Area, size, length and performance estimates

- Software Test Bench

- SoC Specs
  - SoC Modeling/Simulation
  - Estimation

- IP Library
  - Area
  - Timing
  - Power

- Power management Strategy
  - Explores bus, memory and component varieties

- Pre-characterized components
  - E.g. Powerwise, IEM, etc…

- area vs. performance vs. power

Power, area, performance
Outline

- Motivation
- CA Exploration at Transaction Level
- Floorplan-aware Bus Architecture Synthesis Approach
- SoC Power/Energy Modeling
- Design Drivers
- Summary
Design Drivers

- **Case Studies**
  - JPEG2000 encoder
  - H.264 video decoder

---

**JPEG 2000 Encoder**

- Preprocessing
- DWT Transform
- Quantization
- EBCOT encoder
- Tier-1 coder
  - Context Modeling
  - Arithmetic Coder
- Tier-2 coder

**H.264 Decoder**

- Input video signal
- (split into macroblocks of 16x16 pixels)
- Coder control
- Transform/scal/quant.
- Scaling & inv. transform
- Intra-frame prediction
- Motion compensation
- Motion estimation
- Entropy coding
Outline

◆ Motivation

◆ CA Exploration at Transaction Level

◆ Floorplan-aware Bus Architecture Synthesis Approach

◆ Power/Energy Modeling

◆ Design Drivers

◆ Summary
Summary

- Presented work on SoC Performance and Power Modeling

- Key Concepts
  - Communication Architecture Exploration for IP-based Design
  - Transaction-Level Modeling Abstraction
  - Integration of Physical Design Concerns
  - Power/Energy Characterization at SoC Level

- Related Efforts in My Lab
  - Specifications/Requirements Capture using SoC ADL
    - ADL: Architecture Description Language
  - Validation/Verification of SoC Specifications
    - Formal, Semi-formal and Simulation Based Techniques
  - ADL-driven SoC Performance and Power Exploration
Acknowledgements

- CCATB and FABSYN research done jointly with
  - PhD student Sudeep Pasricha
  - Conexant collaborator Dr. Mohamed Ben-Romdhane

- SOC Power Optimization Framework
  - Research project jointly with Prof. Fadi Kurdahi, EECS, UCI

- Sponsors
  - Conexant, Inc. and UC MICRO program
  - NSF
  - SRC
Thank You!
Related Publications


Back-up slides from ASAP
# CCATB Transaction Token Fields

<table>
<thead>
<tr>
<th>Request field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m_data</td>
<td>pointer to an array of data</td>
</tr>
<tr>
<td>m_burst_length</td>
<td>length of transaction burst</td>
</tr>
<tr>
<td>m_burst_type</td>
<td>type of burst (incr, fixed, wrapping etc.)</td>
</tr>
<tr>
<td>m_byte_enable</td>
<td>byte enable strobe for unaligned transfers</td>
</tr>
<tr>
<td>m_read</td>
<td>indicates whether transaction is read/write</td>
</tr>
<tr>
<td>m_lock</td>
<td>lock bus during transaction</td>
</tr>
<tr>
<td>m_cache</td>
<td>cache/buffer hints</td>
</tr>
<tr>
<td>m_prot</td>
<td>protection modes</td>
</tr>
<tr>
<td>m_transID</td>
<td>transaction ID (needed for OO access)</td>
</tr>
<tr>
<td>m_busy_idle</td>
<td>schedule of busy/idle cycles from master</td>
</tr>
<tr>
<td>m_ID</td>
<td>ID for identifying the master</td>
</tr>
</tbody>
</table>
Back-up slides from DAC
Wire Delay Estimation

Then the delay for a wire of length $l$, is given by

$$T = R_d C_o + \left( \frac{\alpha_1 l}{W^2(\alpha_2 l)} + \frac{2\alpha_1 l}{W(\alpha_2 l)} + R_d C_f + \sqrt{R_d r C_a C_f l} \right) . l$$

where

$$\alpha_1 = \frac{1}{4} r C_a$$

$$\alpha_2 = \frac{1}{2} \sqrt{\frac{r C_a}{R_d C_L}}$$

$$C_o = \sum_{j=1}^{k} C_j - C_L$$

$$C_L = \sum_{j=1}^{k} \sum_{i=1}^{j} \frac{l_i}{l} . C_j$$
Wire Delay Estimation

- Inputs to the wire delay estimation engine are
  - wire lengths from the floorplanner and
  - the capacitive loads (CL) of component output pins

- The wire delay estimation is adapted from the models proposed by Cong and Pan [ICCAD '01]
Wire Delay Estimation

- Other parameters include
  - $W(x)$ is Lambert’s $W$ function defined as the value of $w$ which satisfies $we^w=x$
  - $R_d$ is the resistance of the driver
  - $l$ is the wire length
  - process technology dependent parameters (shown in Table)
    - $r$ is the sheet resistance in $\Omega$/sq,
    - $c_a$ is unit area capacitance in fF/$\mu$m$^2$
    - $c_f$ is unit fringing capacitance in fF/$\mu$m
      (sum of fringing and coupling cap.)

<table>
<thead>
<tr>
<th>Tech (µm)</th>
<th>0.18</th>
<th>0.15</th>
<th>0.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>0.068</td>
<td>0.073</td>
<td>0.081</td>
</tr>
<tr>
<td>$c_a$</td>
<td>0.060</td>
<td>0.054</td>
<td>0.046</td>
</tr>
<tr>
<td>$c_f$</td>
<td>0.064</td>
<td>0.054</td>
<td>0.043</td>
</tr>
</tbody>
</table>
Detecting Bus Cycle Time Violations

- IP1 and IP2 are connected to the same bus as ASIC1, Mem4, ARM, VIC and DMA.

- To meet throughput constraints, bus speed is set to 333 Mhz
  - Implies a bus cycle time of 3 ns.

- For a 0.13 µm process, $R_d = 0.4 \, k\Omega$, $C_L = 2.936 \, pF$ and $C_O = 0.988 \, pF$ the floorplanner finds wire length = 9.9 mm between pins connecting the two IPs to the bus
  - Implies a wire delay of 3.5 ns.
  - This is a violation of the clock cycle time constraint of 3 ns.

- Our BA synthesis flow attempts to automatically eliminate such violations once they are detected.
Related Work

- Other approaches have made use of high level floorplanner before, but for different reasons
  - Dick et al. [DATE ‘99] invoked it to obtain global wiring delays to ensure that real time deadlines were met during custom bus topology synthesis
  - Drinic et al. [ICCAD ‘00] used it to determine design feasibility by comparing estimates of wire length with an upper bound on wire length
  - Hu et al. [ASPDAC ‘02] used it to estimate wire length, for calculating energy consumption in point to point networks
  - Bergamaschi et al. [CODES+ISSS ‘03] and Thepayasuwan et al. [DATE ‘04] used it to generate an early core placement estimate
SoC Performance Constraints

- SoC designs have performance constraints that can be represented in terms of *Data Throughput Constraints*

- **Communication Throughput Graph (CTG)** incorporates SoC components and throughput constraints, where
  - each edge connects 2 communicating components
  - each vertex represents a component and information about its
    - area
    - dimensions
    - capacitive loads on output pins
    - which bus type it connects to

- **Throughput Constraint Path (TCP)** is a sub-graph of a CTG that
  - contains a master for which data throughput must be maintained, and includes other masters, slaves and memories in the critical path
Select previously unselected bus from BA

Reduce bus width. Simulate

TCP violation?

Y

Undo bus width reduction

N

Reduce bus speed. Simulate

all busses examined?

Y

exit

N

– Reducing bus widths and speeds
  ■ reduces system cost
  ■ lower bus speed implies larger bus cycle time, (less probability of bus cycle time violation)
Why worry about power? -- Chip Power Density

Source: Borkar, De Intel®
Why worry about power? -- Standby Power

Drain leakage will increase as $V_T$ decreases to maintain noise margins and meet frequency demands, leading to excessive battery draining standby power consumption.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Power supply $V_{dd}$ (V)</td>
<td>1.5</td>
<td>1.2</td>
<td>0.9</td>
<td>0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>Threshold $V_T$ (V)</td>
<td>0.4</td>
<td>0.4</td>
<td>0.35</td>
<td>0.3</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Source: Borkar, De Intel®

...and phones leaky!
Multimedia Controller SoC Example

Communication between IPs significantly affects system performance and power!
Communication Architectures

NOC based

Bus based
CCATB Transaction Example

```c
process lcdc()
{
    ...
    if (enable.read() == 1) {
        read(port, SDRAM_addr1, token);
        wait(wait_period);
        size_info = token->data;
    }
    ...
}
```

```c
channel_status_slave *
read (SDRAM_ADDR_TYPE addr_in,
slave_data_and_control * packet)
{
    ...  
    switch (addr_in - m_start_address)
    {
    case SDRAM_CONTR_MODE:
        *(packet->data) = m_mode;
        slave_status->status = BUS_OK;
        slave_status->wait_cyc = 4;
        return slave_status; break;
    case SDRAM_CONTR_RESET: ...
```
Modeling Abstractions for CA Exploration

\[
v_1 = a + b; \\
\text{wait}(1); \text{/cycle 1} \\
\text{REG} = d \ll v_1; \\
\text{wait}(1); \text{/cycle 2} \\
\text{REQ.set}(1); \\
\text{ADDR.set(REG);} \\
\text{WDATA.set}(v_1); \\
\text{wait}(1); \text{/cycle 3}
\]

Simulation speed: ~10 - 100x RTL

Modeling effort: /1 - /3 RTL

\[
\text{bus} \\
\text{arb}
\]

Pin Accurate Bus Cycle Accurate (PA-BCA)

\[
\text{bus} \\
\text{arb}
\]

Cycle Accurate (CA)

\[
\text{bus} \\
\text{arb}
\]

Transaction based Bus Cycle Accurate (T-BCA)

\[
\text{bus} \\
\text{arb}
\]

Transaction level Model (TLM)

Simulation speed: >>1000x RTL

Modeling effort: ~/20 RTL

Increasing simulation accuracy

Increasing simulation speed

Simulation speed:

- ~1000x RTL
- ~100 - 500x RTL
- ~10 - 100x RTL
- ~100x
- >>1000x

Modeling effort:

- /10 RTL
- /5 - /10 RTL
- /3 RTL
- /1 - /3 RTL
- ~/20 RTL
- ~/10 RTL
- /1 - /3 RTL