Automated Bus Generation for Multiprocessor SoC Design

Dissertation Defense
by
Kyeong Keol Ryu

Advisor: Vincent J. Mooney III

School of Electrical and Computer Engineering
Georgia Institute of Technology

June 2004
Outline

- Introduction
- Related Work
- Methodology for Bus System Generation
- Experiments and Results
- Conclusion
Introduction – Goal

High Performance Multi-processor SoC Design

Our Approach:
• Use custom SoC buses and custom bus interfaces
• Fast design space exploration

Other Approaches:
• Use standard SoC buses and standard bus interfaces as a generic approach
Introduction – Motivation 1

- Hardware-software partitioning

**OFDM Transmitter**
- **Thread 1**: Data input and Symbol Mapping
- **Thread 2**: Inverse Fast Fourier Transform (IFFT)
- **Thread 3**: Data output and Cyclic extension

**Software**

**SoC Hardware**
Introduction – Motivation 2

- Automatic custom bus generation for a multiprocessor System-on-a-Chip (SoC)
  - Easy and quick design of an SoC bus system
  - Fast design space exploration across performance influencing factors
  - Development of a bus synthesis tool (BusSynth)
  - Register-transfer level HDL output
Outline

- Introduction
- Related Work
  - SoC Bus Architectures
  - SoC Bus Interfaces
  - SoC Bus System Design Tools
  - Additional prior work
- Methodology for Bus System Generation
- Experiments and Results
- Conclusion
SoC Bus Architectures

- CoreConnect from IBM: PLB, OPB and DCR
- Advanced Microcontroller Bus Architecture (AMBA) from ARM: AHB, ASB and APB
SoC Bus Architectures (Continued)
- CoreFrame from Palmchip: PalmBus and Mbus
- Wishbone from Silicore: single bus type
SoC Bus Architectures (Continued)

- **SiliconBackplane μNetwork from Sonics**
  - Provision of fixed bandwidth by TDMA-based arbitration

  ![Diagram of SiliconBackplane μNetwork](image)

  - **Our Case:**
    - Custom bus architectures from BusSynth: GBAVI, GBAVIII, BFBA, HybridBA and SplitBA
    - More suitable for user applications and better performance
SoC Bus Interfaces

- Open Core Protocol (OCP) from Sonics
  - Bus interface for IP cores
  - Reconfigurable interface
  - Five versions: basic OCP and its four extensions

- Virtual Component Interface (VCI) from Virtual Socket Interface Alliance (VSIA)
  - Basically a handshake protocol
  - A protocol for cycle-based point-to-point communication
  - A data-orientated protocol (w/o the consideration of interrupt control, and scan test issues)
  - Three versions: PVCI, BVCI and AVCI
SoC Bus Interfaces (Continued)

- Interface logic blocks (wrappers)
  - OCP and VCI: provision of a generic interface
  - Our case:
    - Custom wrappers: provision of a customized interface to each specific IP block
    - Examples: MBI for a memory, CBI for a processing element, and ABI for an arbiter
    - More suitable interfaces due to custom architecture and lead to better system performance
SoC Bus System Tools

- CoWare N2C from CoWare
  - A design environment for an SoC
  - Bus generator and simulator to design a bus architecture for an SoC

- Platform Express from Mentor Graphics
  - An IP block and bus integration tool for an SoC
  - IP block assembling by dragging and dropping library components
  - AMBA and CoreConnect

- CoCentric System Studio from Synopsys
  - A SystemC simulator and specification environment for HW architectures and SW algorithms
  - Bus architecture solutions: DesignWare AMBA IP blocks and ARM processors
SoC Bus System Tools (Continued)

- Magillem from Prosilog
  - A tool for importing IPs and graphically creating SoCs
  - Supports:
    - Standard on-chip buses: AMBA and CoreConnect
    - Standard bus interfaces: OCP and VCI

- BusSynth
  - Generation of SoC bus systems with the standard buses as well as customized buses.
  - Single bus architecture as well as multiple and hybrid bus architectures: GBAVI, GBAVIII, BFBA, HybridBA and SplitBA
  - Interconnect delay aware bus architecture generation
Additional Prior Work

- M. Gasteier et al. (’96), “Bus-Based Communication Synthesis on System-Level”
  - Automatic generation of communication topologies on system-level
  - A single global bus topology

- R.A. Bergamaschi et al. (’00), “Designing Systems-on-Chip using Cores”
  - Assembling an SoC using IP blocks and their properties
  - A single type of bus topology

- TIMA lab. (’02): component-based design and wrapper generation
  - Support: point-to-point connection and a shared bus

  - A single type of bus topology

- BusSynth
  - a variety of bus types including multiple and heterogeneous type
  - Interconnect delay aware bus generation
Additional Prior Work (Continued)

  - A component-based approach to SoC system building

- BusSynth
  - various customized bus architectures by using user options
Outline

- Introduction
- Related Work
- **Methodology for Bus System Generation**
  - Overview
  - Bus System Structure
  - Bus System Generation
  - Bus System Examples
  - Interconnect Delay Aware Generation
- Experiments and Results
- Conclusion
Methodology Overview

- **BusSynth**
  - User options
  - Interconnect delay estimation
  - Custom bus systems in Register-Transfer Level (RTL) HDL code

- **Bus systems**
  - Hierarchical structure to build an SoC bus system: module, Bus Access Note (BAN), bus subsystem and bus system
  - Each layer is assembled in a configurable manner
Bus System Structure – an example

Bus System

Bus Subsystem 1

- CPU_A
  - Memory
  - CBI
  - MBI
- CPU_B
  - Memory
  - CBI
  - MBI
- Arbiter
  - ABI
  - GBI

Bus Subsystem 2

- CPU_I
  - Memory
  - CBI
  - MBI
- CPU_J
  - Memory
  - CBI
  - MBI

BANs: BAN G, BAN A, BAN B, BAN J

INs: IL1, IL2, IL3, IL4

G: Generic

M: Memory-Bus Interface

A: Arbiter-Bus Interface

C: CPU/PE-Bus Interface

© 2004 Georgia Institute of Technology
Bus System Generation

BusSynth

User Option Input

For each Subsystem $i$

Bus Access Node (BAN) Generation

Bus Subsystem Generation

# of Subsystem $\geq 1$

Y

N

Bus System Generation

Synthesizable Verilog HDL code

Module Library

Wire Library
Bus System Generation

- **Module Library**
  - PE: MPC750, MPC755, MPC7410 and ARM9TDMI
  - [memory]_comp: SRAM and DRAM
  - CBI_[PE]
  - MBI_[memory]
  - ABI
  - GBI_[bus_type]: GBAVI, GBAVIII and BFBA
  - BB_[bb_type]: GBAVI and SplitBA
  - ARB_[arb_type]: Priority and Round Robin
  - SB_[bus_type]
Bus System Generation (Continued)

- **Wire Library**
  - **Format**
    ```verilog
    %wire <library_name>;
    w_name w_width m1_name m1_pname
      m1_wmsb m1_wlsb m2_name m2_pname
      m2_wmsb m2_wlsb;
    %endwire;
    ```
  - **An example:**
    ```verilog
    %wire ban1;
    w_addr 32 MPC755 addr_pe 31 0
      CBI_MPC755 addr_cbi 31 0;
    %endwire;
    ```
Bus System Generation (Continued)

- User input list
  - Bus System
    - Number of Bus Subsystems
  - Bus Subsystem (for each Bus Subsystem)
    - Number of buses
    - Number of BANs:
  - Bus Properties (for each bus)
    - Bus Type: GGBA, GBAVI, GBAVIII, BFBA, HybridBA or SplitBA
    - address bus width
    - data bus width
    - Bi-FIFO depth for BFBA and HybridBA
  - BAN Properties (for each BAN)
    - CPU Type: MPC750, MPC755, MPC7410 or ARM9TDMI
    - Non-CPU Type: DCT or MPEG2 decoder
    - Number of global memories
    - Number of local memories
  - Memory Properties
    - Type: SRAM, DRAM, DPRAM or FIFO
    - Address bus width
    - Data bus width

© 2004 Georgia Institute of Technology 22
Bus System Generation (Continued)

- Example: user input for SplitBA

1. Bus System: # of Bus Subsystems = 2
2. Bus Subsystem:
   - Bus Subsystem1: # of buses = 1 and # of BANs = 3
   - Bus Subsystem2: # of buses = 1 and # of BANs = 3
3. Bus Properties:
   - Bus Subsystem1: GGBA, address bus width = 32 and data bus width: 64
   - Bus Subsystem2: GGBA, address bus width = 32 and data bus width: 64
4. BAN Properties:
   For Bus Subsystem1
   - BAN1: CPU Type = MPC755, non-CPU Type = None,
     # of global memories = 0 and # of local memories = 0
   - BAN2: CPU Type = MPC755, non-CPU Type = None,
     # of global memories = 0 and # of local memories = 0
   - BAN3: CPU Type = None, non-CPU Type = None,
     # of global memories = 1 and # of local memories = 0
   For Bus Subsystem2
   - BAN4: CPU Type = MPC755, non-CPU Type = None,
     # of global memories = 0 and # of local memories = 0
   - BAN5: CPU Type = MPC755, non-CPU Type = None,
     # of global memories = 0 and # of local memories = 0
   - BAN6: CPU Type = None, non-CPU Type = None,
     # of global memories = 1 and # of local memories = 0
5. Memory Properties:
   - BAN3: Type = SRAM, address bus width = 21 and data bus width = 64
   - BAN6: Type = SRAM, address bus width = 21 and data bus width = 64
Bus System Generation (Continued)

- Bus Subsystem Generation

BusSubSys (module_name_array MNA, ban_name_array BN, subsys_no N, wire_library WL, module_library ML)

For i = 1 to N, i = i + 1

For BAN j in BAN name array BN for a Subsystem i

For module k in MNA for BAN j

Look up module k name in Module Library ML and extract or generate the corresponding RTL code for each module k

Call UnitGen (MNA, “ban_i_j”, WL)

Call UnitGen (BN, “bus_subsystem_i”, WL)
Bus System Generation (Continued)

- Example: the generation of Bus Subsystems for SplitBA

For each Subsystem 2

- Bus Access Node 6 (BAN6) Generation
- Bus Subsystem 2 Generation

# of Subsystem > 1

Y

Bus System Generation

Synthesizable
Verilog HDL code

Example: the generation of Bus Subsystems for SplitBA
Bus System Generation (Continued)

**Unit generation**

**UnitGen** (module_name_array \( M \), top_unit_name \( U \), wire_library \( W \))

1. For each module \( i \) in module name array \( M \)

   **Read wires from** \( W \) **for modules in** \( M \) **and save them to LW1**

   **Read ports from module** \( i \) **and save them to LP1**

2. For port \( j \) in LP1

   \( \textbf{flag} = \text{FALSE} \)

3. For wire \( k \) in LW1

   **info in LP1 matches info in LW1**

   **Save wire** \( k \) **and its connection info to LWPM**

   \( \textbf{flag} = \text{FALSE} \)

   **Add port** \( j \) **info to LP2**

Write HDL code for a module to \( U \) using LW1, LP2 and LWPM

```hdl
module BAN1(reset_b, ....);
   input reset_b;
   ....
   wire w_addr[31:0];
   ....
   MPC755 MPC755_0(
      .reset_b(reset_b),
      .addr_pe(w_addr[31:0]),
      ....
    );
   CBI_MPC755 CBI_MPC755_0(
      .addr_cbi(w_addr[31:0]),
      ....
    );
endmodule;
```

LW1

\[ w_{-}addr \ 32 \ \text{MPC755} \ \text{addr}_\text{pe} \ 31 \ 0 \ \text{CBI}_\text{MPC755} \ \text{addr}_\text{cbi} \ 31 \ 0; \]

LP1 for MPC755

\[ \text{MPC755} \ \text{addr}_\text{pe} \ \text{output} \ 31 \ 0; \]
\[ \text{MPC755} \ \text{reset}_\text{b} \ \text{input} \ 0 \ 0; \]

LP1 for CBI_MPC755

\[ \text{CBI}_\text{MPC755} \ \text{addr}_\text{cbi} \ \text{input} \ 31 \ 0; \]

LWPM

\[ w_{-}addr \ \text{MPC755} \ \text{addr}_\text{pe} \ 31 \ 0; \]
\[ w_{-}addr \ \text{CBI}_\text{MPC755} \ \text{addr}_\text{cbi} \ 31 \ 0; \]

LP2

\[ \text{MPC755} \ \text{reset}_\text{b} \ \text{input} \ 0 \ 0; \]
Bus System Generation (Continued)

Bus System Generation

BusSys (subsystem_name_array SS, bus_bridge_name_array MS, wire_library WL, Module_library ML)

For each Subsystem $i$

Bus Access Node (BAN) Generation

Bus Subsystem Generation

For module $i$ in bus bridge name array MS

Look up module $i$ name in Module Library ML and extract or generate the corresponding RTL code for the module $i$

Call UnitGen ([SS, MS], “bus_system”, WL)

Synthesizable Verilog HDL code

# of Subsystem > 1

Bus System Generation

Y

N

Module Library

Wire Library
Example: the generation of a Bus System (SplitBA)
Bus System Examples

General Global Bus Architecture Version I (GBAVI)

Bi-FIFO Bus Architecture (BFBA)

[Note] BB: bus bridge and HS_REGS: handshake registers

[Note] HS_REGS: handshake registers
Bus System Examples (Continued)

General Global Bus Architecture Version III (GBAVIII)

Hybrid Bus Architecture (HybridBA)
Bus System Examples (Continued)

Split Bus Architecture (SplitBA)
Bus System Examples (Continued)

CoreConnect Bus Architecture (CCBA)

General Global Bus Architecture (GGBA)
A New Bus System Generation

- Different Combination of Bus Components
  - Different combination of BAN components
    - **User Inputs for BAN1**: CPU type: MPC755
      - Non-CPU type: None
      - # of global memories: 0
      - # of local memories: 1
      - Memory type: SRAM
    - **User Inputs for BAN2**: CPU type: MPC755
      - Non-CPU type: None
      - # of global memories: 0
      - # of local memories: 0
    - **User Inputs for Bus Subsystem1**: # of BANs: 4
      - BANs 1, 2, 3 and 4
    - **User Inputs for Bus Subsystem2**: # of BANs: 4
      - BANs 1, 2, 2 and 2

- Different combination of BANs
  - **Bus Subsystem1**: BAN1 → BAN2 → BAN3 → BAN4
  - **Bus Subsystem2**: BAN1 → BAN2 → BAN2 → BAN2
  - **Bus Subsystem3**: BAN1 → BAN3 → BAN4 → BAN4

A New Bus System Generation

(Continued)

- Different Combination of Bus Components
  - Different combination of Bus Subsystems

User Inputs for Bus System1:
- # of Bus Subsystems: 3
  - Bus Subsystems 1, 2, 3

User Inputs for Bus System2:
- # of Bus Subsystems: 2
  - Bus Subsystems 1 and 2

**Note:** BB: Bus Bridge
Interconnect Delay Aware Bus System Generation

- Interconnect delay estimation (e.g., GGBA)

- HSPICE wire model includes:
  - RLC parameters from MOSIS run for TSMC 0.25 um
  - Interconnect length
- Interconnect delay calculation

<table>
<thead>
<tr>
<th>Processing Element</th>
<th>Length [cm]</th>
<th>Delay [ns]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing Element 1</td>
<td>0.2521</td>
<td>0.2848</td>
</tr>
<tr>
<td>Processing Element 2</td>
<td>0.6143</td>
<td>0.5727</td>
</tr>
<tr>
<td>Processing Element 3</td>
<td>1.2753</td>
<td>2.2882</td>
</tr>
<tr>
<td>Processing Element 4</td>
<td>1.9363</td>
<td>3.0472</td>
</tr>
</tbody>
</table>

(a) Estimated Floorplan of GGBA
(b) Interconnect length estimation
Memory Bus Interface (MBI) module generation

- One of effects in interconnect delay insertion: memory access cycles
- Memory controller to adapt access cycles due to interconnect delay
Interconnect Delay Aware Bus System Generation (Continued)

- Memory Bus Interface (MBI) module generation

<table>
<thead>
<tr>
<th></th>
<th>Estimated bus delay between a PE and SRAM [ns]</th>
<th>Delay in a read operation [ns]</th>
<th>SRAM (2Mbyte) access time [ns]</th>
<th>Total interconnect delay in a read operation [ns]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PE 1</td>
<td>0.2848</td>
<td>0.5696</td>
<td>8.00</td>
<td>8.5696</td>
</tr>
<tr>
<td>PE 2</td>
<td>0.5727</td>
<td>1.1454</td>
<td>8.00</td>
<td>9.1454</td>
</tr>
<tr>
<td>PE 3</td>
<td>2.2882</td>
<td>4.5764</td>
<td>8.00</td>
<td>12.5764</td>
</tr>
<tr>
<td>PE 4</td>
<td>3.0472</td>
<td>6.0944</td>
<td>8.00</td>
<td>14.0944</td>
</tr>
</tbody>
</table>

Note: the access time of a shared SRAM (2Mbytes) is estimated by CACTI 3.0

(a) Estimated total delay of paths between each PE and a shared memory

<table>
<thead>
<tr>
<th></th>
<th>Number of clock delays in each PE for a read operation [clock]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>100 MHz (10.00ns) system clock</td>
</tr>
<tr>
<td>PE 1</td>
<td>1 (0.8570)</td>
</tr>
<tr>
<td>PE 2</td>
<td>1 (0.9145)</td>
</tr>
<tr>
<td>PE 3</td>
<td>2 (1.2576)</td>
</tr>
<tr>
<td>PE 4</td>
<td>2 (1.4094)</td>
</tr>
</tbody>
</table>

(b) Number of clock delays in data paths
Interconnect Delay Aware Bus System Generation (Continued)

- Memory Bus Interface (MBI) module generation

(a) Sequence of MBI Generation

- Input of interconnect delays
- Calculation of the number of clocks to be inserted
- Extraction of MBI module from Module Library
- Update of memory access delay parameters in an MBI module

(b) Bus System Generation

- User Option Input
- For each Bus Subsystem
  - Module Generation
  - Bus Access Node (BAN) Generation
  - Bus Subsystem Generation
- # of Subsystem > 1
- Y
- Bus System Generation
- N
- Synthesizable Verilog HDL code
Outline

- Introduction
- Related Work
- Methodology for Bus System Generation
- Experiments and Results
  - Application Examples
  - Experimental Setup
  - Performance Evaluation
  - Generation Time and Logic Area
- Conclusion
Application Examples

- OFDM transmitter
  - Wireless application
  - One packet: (2048+512)-complex samples
- MPEG2 decoder
  - A video stream decoder
- Database example
  - Multitask clients and server over PEs: total 41 tasks over four PEs
  - RTOS: Atalanta version 0.4
Experimental Setup

BUS GENERATION TOOL

INPUT
User options

BusSynth

SYNTHESISABLE
VERILOG HDL
CODE

LIBRARIES

INTERCONNECT
Delay Estimation

Floorplan
Design

SIMULATION ENVIRONMENT

VCS

SEAMLESS
CVE

XRAY

GCC

USER
C-CODE

SYNTHESIS ENVIRONMENT

DESIGN
COMPILER

Note: VCS and Design Compiler from Synopsys, Seamless CVE and Xray from Mentor Graphics and GCC from GNU
(a) Pipelined Parallel Algorithm (PPA)  (b) Functional Parallel Algorithm (FPA)

Note: Each of E, F, G and H specifies a function group partitioned from a software
Performance Evaluation

- **OFDM Transmitter**

<table>
<thead>
<tr>
<th>Case</th>
<th>Bus System</th>
<th>Application Throughput [Mbps]</th>
<th>Software Programming Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>BFBA</td>
<td>2.6504</td>
<td>PPA</td>
</tr>
<tr>
<td>2</td>
<td>GBAVI</td>
<td>2.1087</td>
<td>PPA</td>
</tr>
<tr>
<td>3</td>
<td>GBAVIII</td>
<td>4.5599</td>
<td>FPA</td>
</tr>
<tr>
<td>4</td>
<td>GBAVIII</td>
<td>2.2567</td>
<td>PPA</td>
</tr>
<tr>
<td>5</td>
<td>HybridBA</td>
<td>4.5599</td>
<td>FPA</td>
</tr>
<tr>
<td>6</td>
<td>HybridBA</td>
<td>2.6504</td>
<td>PPA</td>
</tr>
<tr>
<td>7</td>
<td>SplitBA</td>
<td>5.1132</td>
<td>FPA</td>
</tr>
<tr>
<td>8</td>
<td>GGBA</td>
<td>4.3913</td>
<td>FPA</td>
</tr>
<tr>
<td>9</td>
<td>GGBA</td>
<td>2.1880</td>
<td>PPA</td>
</tr>
</tbody>
</table>

Note: 1. PPA: Pipelined Parallel Algorithm, FPA: Functional Parallel Algorithm.  
2. Data: 2048 complex samples and 512 guard complex samples per packet.  
3. Each Bus System having four PowerPCs supports instruction and data cache.

- **SplitBA and GBAVIII** outperform **GGBA** by 16.44% and 13%, respectively.

- Pipelined parallel algorithm (PPA) and functional parallel algorithm (FPA)
Performance Evaluation (Continued)

- **MPEG2 Decoder**
  - HybridBA shows the best in performance (15.54% against CCBA)

<table>
<thead>
<tr>
<th>Case</th>
<th>Bus System</th>
<th>Application Throughput [Mbps]</th>
<th>Software Programming Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>BFBA</td>
<td>0.8594</td>
<td>FPA</td>
</tr>
<tr>
<td>11</td>
<td>GBAVI</td>
<td>0.8271</td>
<td>FPA</td>
</tr>
<tr>
<td>12</td>
<td>GBAVIII</td>
<td>1.1444</td>
<td>FPA</td>
</tr>
<tr>
<td>13</td>
<td>HybridBA</td>
<td>1.1650</td>
<td>FPA</td>
</tr>
<tr>
<td>14</td>
<td>CCBA</td>
<td>1.0083</td>
<td>FPA</td>
</tr>
</tbody>
</table>

Note: Picture size: 16 x 16

- **Database Example**
  - SplitBA outperforms GGBA by 41% reduction in time

<table>
<thead>
<tr>
<th>Case</th>
<th>Bus System</th>
<th>Execution Time [ns]</th>
<th>Software Programming Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>GGBA</td>
<td>2,241,100</td>
<td>FPA</td>
</tr>
<tr>
<td>16</td>
<td>SplitBA</td>
<td>1,317,804</td>
<td>FPA</td>
</tr>
</tbody>
</table>

Note: 1. Each Bus System is composed of 1 server task and 40 client tasks
2. Each task accesses one-hundred data to or from a shared memory

© 2004 Georgia Institute of Technology
Performance Evaluation
- Interconnect Delay Aware Generation

- Three configurations of GGBA for performance comparison
  - **GGBA I** is a GGBA system with no regard to interconnect delay on the bus
    - Used as a baseline of performance comparison
  - **GGBA II** is a GGBA system that works with different estimated interconnect delays on the shared bus
  - **GGBA III** is a GGBA system that operates with a maximum estimated delay on all connections between PEs and a shared memory
Performance Evaluation (Continued)
- Interconnect Delay Aware Generation

<table>
<thead>
<tr>
<th>GGBA System</th>
<th>Execution Time [ns/packet]</th>
<th>Comparison I [increase in execution time]</th>
<th>Comparison II [decrease in execution time]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GGBA I (no interconnect delay)</td>
<td>1,218,455</td>
<td>0.0%</td>
<td>-</td>
</tr>
<tr>
<td>2. GGBA II (3, 3, 4 and 5 clock delays in each data path from PE 1 to PE 4)</td>
<td>2,057,487</td>
<td>68.9%</td>
<td>35.3%</td>
</tr>
<tr>
<td>3. GGBA III (5 clock delays in all data paths)</td>
<td>3,180,220</td>
<td>161.0%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

(a) 300MHz Bus Clock

<table>
<thead>
<tr>
<th>GGBA System</th>
<th>Execution Time [ns/packet]</th>
<th>Comparison I [increase in execution time]</th>
<th>Comparison II [decrease in execution time]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GGBA I (no interconnect delay)</td>
<td>1,825,751</td>
<td>0.0%</td>
<td>-</td>
</tr>
<tr>
<td>2. GGBA II (2, 2, 3 and 3 clock delays in each data path from PE 1 to PE 4)</td>
<td>2,323,679</td>
<td>27.3%</td>
<td>27.4%</td>
</tr>
<tr>
<td>3. GGBA III (3 clock delays in all data paths)</td>
<td>3,198,620</td>
<td>75.2%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

(b) 200MHz Bus Clock

<table>
<thead>
<tr>
<th>GGBA System</th>
<th>Execution Time [ns/packet]</th>
<th>Comparison I [increase in execution time]</th>
<th>Comparison II [decrease in execution time]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. GGBA I (no interconnect delay)</td>
<td>3,644,003</td>
<td>0.0%</td>
<td>-</td>
</tr>
<tr>
<td>2. GGBA II (1, 1, 2 and 2 clock delays in each data path from PE 1 to PE 4)</td>
<td>3,862,686</td>
<td>6.0%</td>
<td>10.1%</td>
</tr>
<tr>
<td>3. GGBA III (2 clock delays in all data paths)</td>
<td>4,297,056</td>
<td>17.9%</td>
<td>0.0%</td>
</tr>
</tbody>
</table>

(c) 100MHz Bus Clock
Generation Time and Logic Area (no wires)

- Bus system generation with BusSynth
- Design Compiler with LEDA TSMC 0.25µm standard cell library

<table>
<thead>
<tr>
<th>Bus System</th>
<th>1 processor</th>
<th>8 processors</th>
<th>16 processors</th>
<th>24 processors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time [ms]</td>
<td>Gate count</td>
<td>Time [ms]</td>
<td>Gate count</td>
</tr>
<tr>
<td>BFBA</td>
<td>509</td>
<td>800</td>
<td>534</td>
<td>6,401</td>
</tr>
<tr>
<td>GBAVI</td>
<td>417</td>
<td>872</td>
<td>432</td>
<td>6,899</td>
</tr>
<tr>
<td>GBAVIII</td>
<td>513</td>
<td>2,070</td>
<td>542</td>
<td>14,746</td>
</tr>
<tr>
<td>HybridBA</td>
<td>763</td>
<td>2,973</td>
<td>859</td>
<td>21,869</td>
</tr>
<tr>
<td>SplitBA</td>
<td>N/A</td>
<td>N/A</td>
<td>413</td>
<td>4,297</td>
</tr>
</tbody>
</table>

Note: Time: Bus generation time, N/A: Not Applicable
    Gate count: NAND2 gate count in TSMC 0.25µm standard cell library
Conclusions

- SoC bus system design aid
  - Expert guide to design an SoC bus system
- Automated bus generation tool: BusSynth
  - Solution: how to easily and quickly design a multi-processor SoC bus system
  - User option based tool that generates diverse custom bus systems
  - Synthesizable Verilog HDL output
- Interconnect delay aware bus system generation
- A case study of an SoC design in a component-based design approach
- Fast design space exploration across performance influencing factors
  - Generation of bus systems in a matter of seconds
- Practical implementation
  - RTL-level HDL output from BusSynth
  - Realistic user application: OFDM and MPEG2
  - Real-time operating system
Publications


Poster Presentation and Demonstration


Thank you