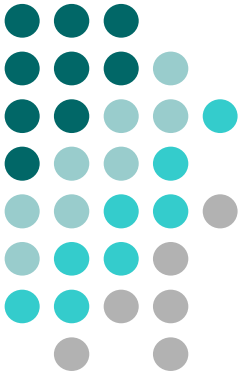


Georgia Institute of Technology

Automated Bus Generation for Multiprocessor SoC Design

Dissertation Defense

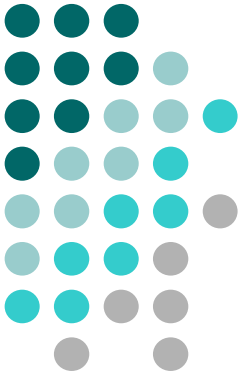
by

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

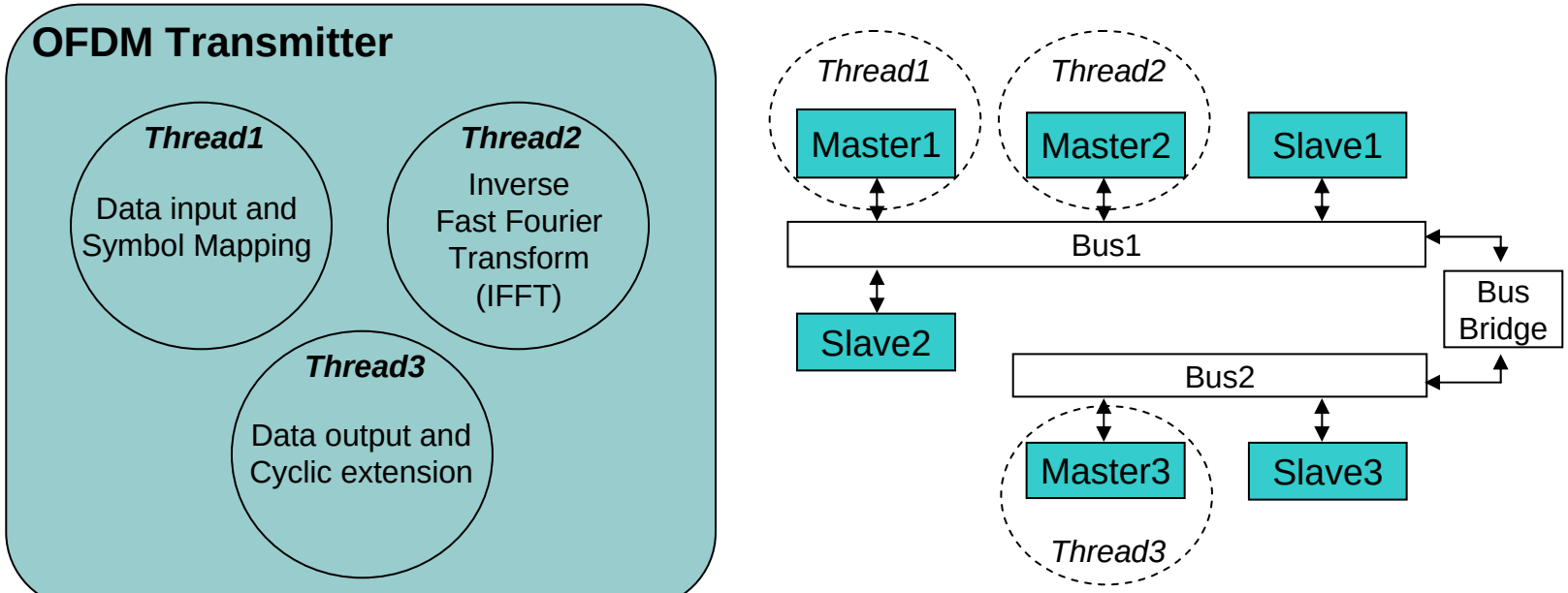
High Performance Multi-processor SoC Design

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

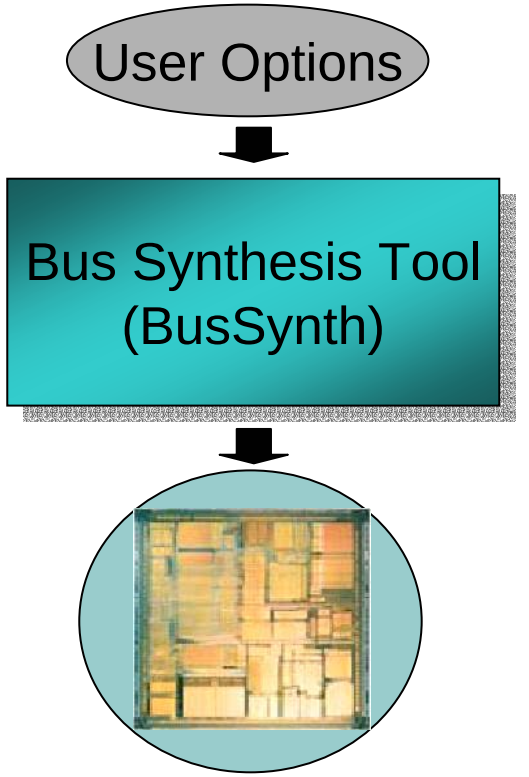
- Hardware-software partitioning



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

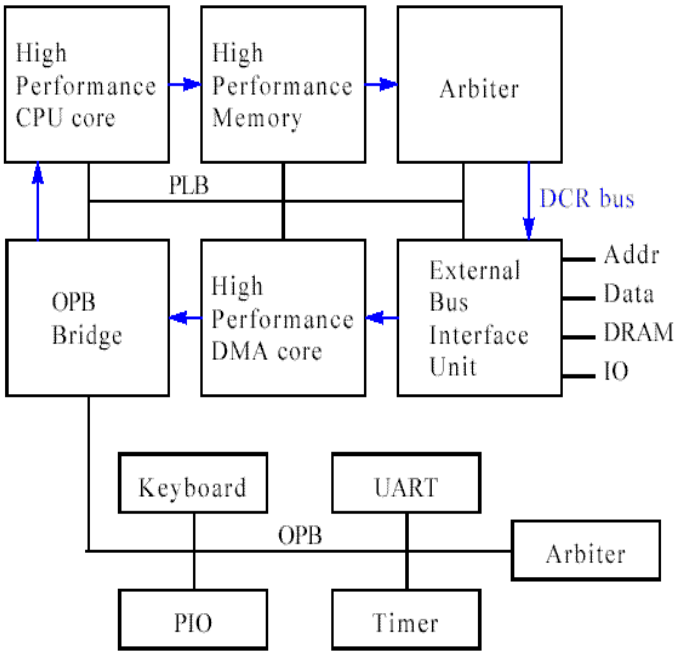


Outline

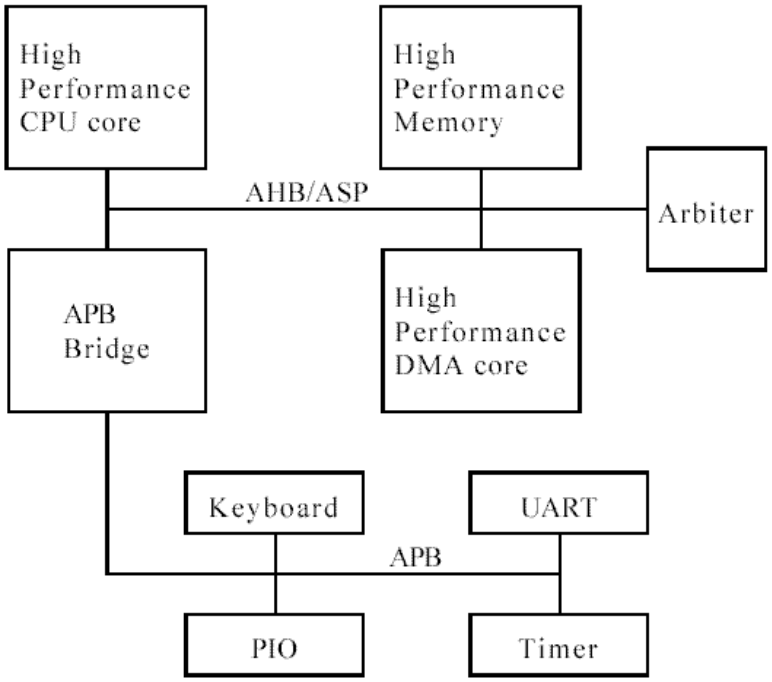
- Introduction
- **Related Work**
  - **SoC Bus Architectures**
  - **SoC Bus Interfaces**
  - **SoC Bus System Design Tools**
  - **Additional prior work**
- Methodology for Bus System Generation
- Experiments and Results
- Conclusion

SoC Bus Architectures

- CoreConnect from IBM: PLB, OPB and DCR
- Advanced Microcontroller Bus Architecture (AMBA) from ARM: AHB, ASB and APB



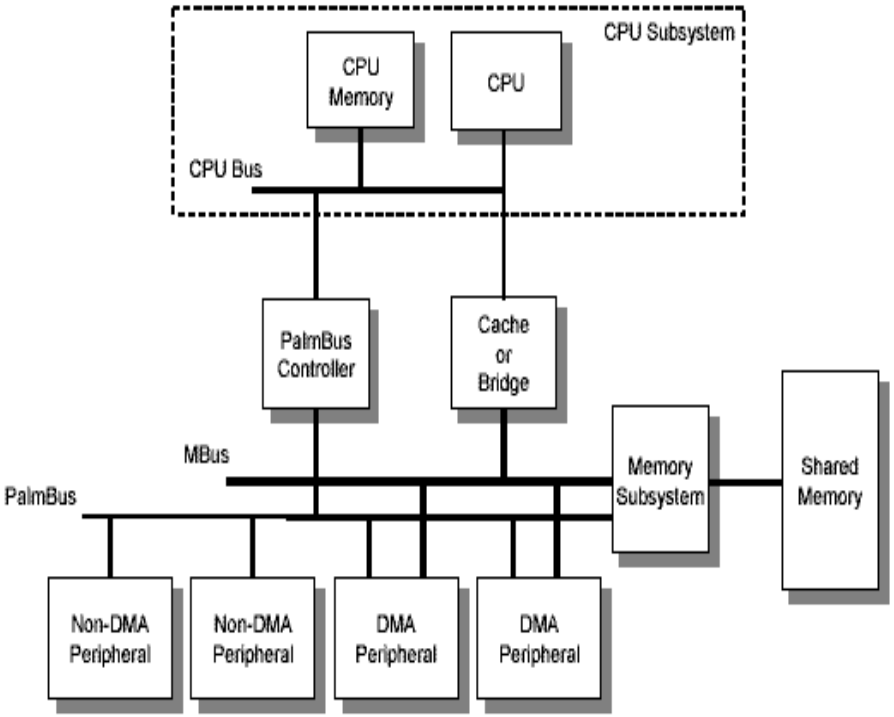
CoreConnect



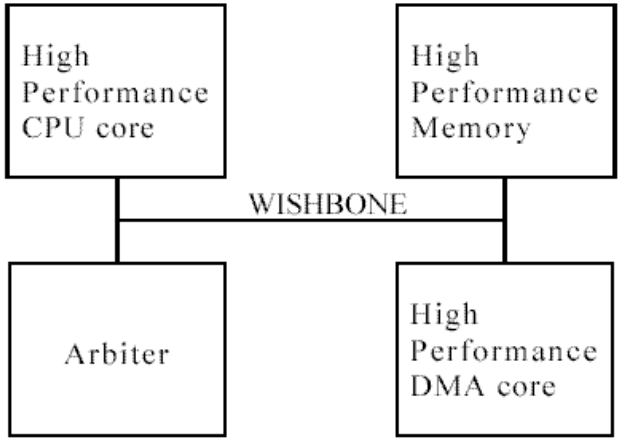
AMBA

SoC Bus Architectures (Continued)

- CoreFrame from Palmchip: PalmBus and Mbus
- Wishbone from Silicore: single bus type



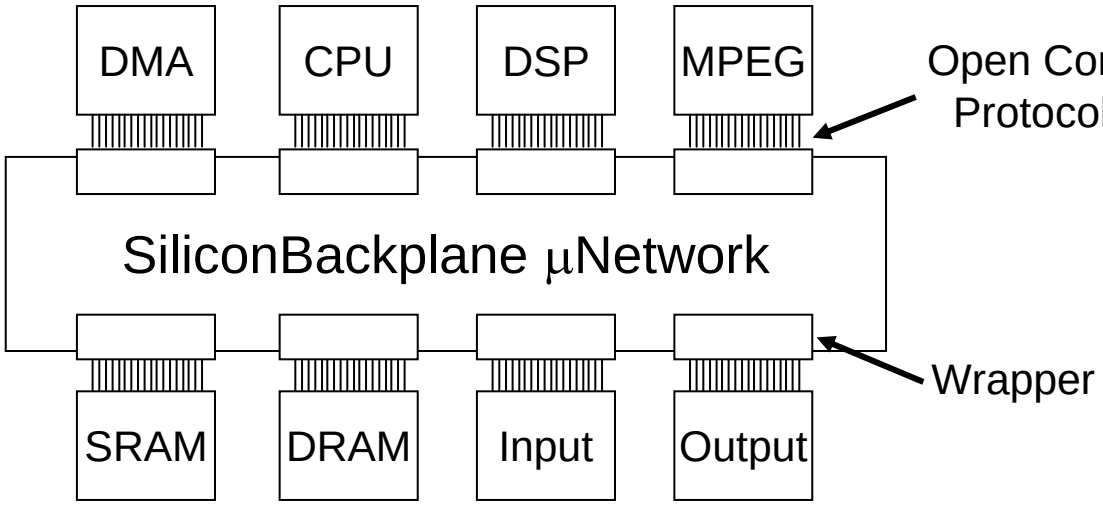
CoreFrame



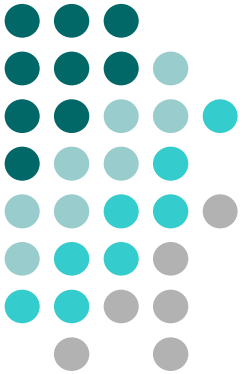
Wishbone

SoC Bus Architectures (Continued)

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

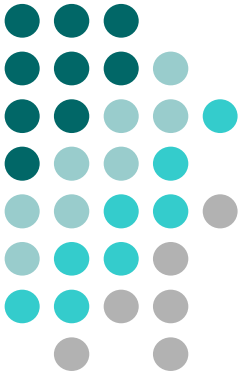


- Our Case:
  - Custom bus architectures from BusSynth: GBAVI, GBAVIII, BFBA, HybridBA and SplitBA
  - More suitable for user applications and better performance



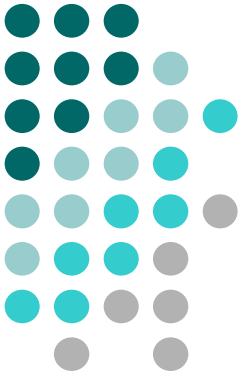
SoC Bus Interfaces

- 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
- Shin *et al*. (’04), “Fast Exploration of Parameterized Bus Architecture for Communication-Centric SoC Design”
  - 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)

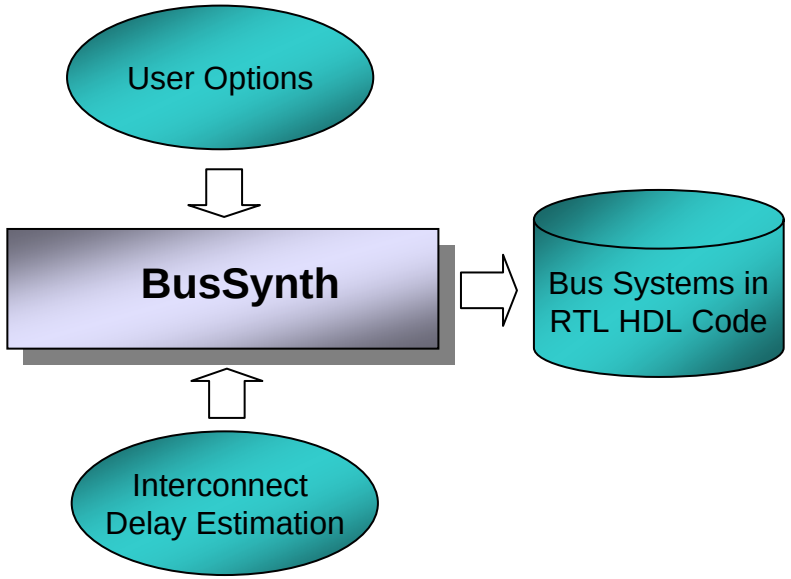
- Pai Chou *et al*. ('99), "IPCHINOOK": An Integrated IP-based Design Framework for Distributed Embedded Systems"
  - A component-based approach to SoC system building
- BusSynth
  - various customized bus architectures by using user options

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

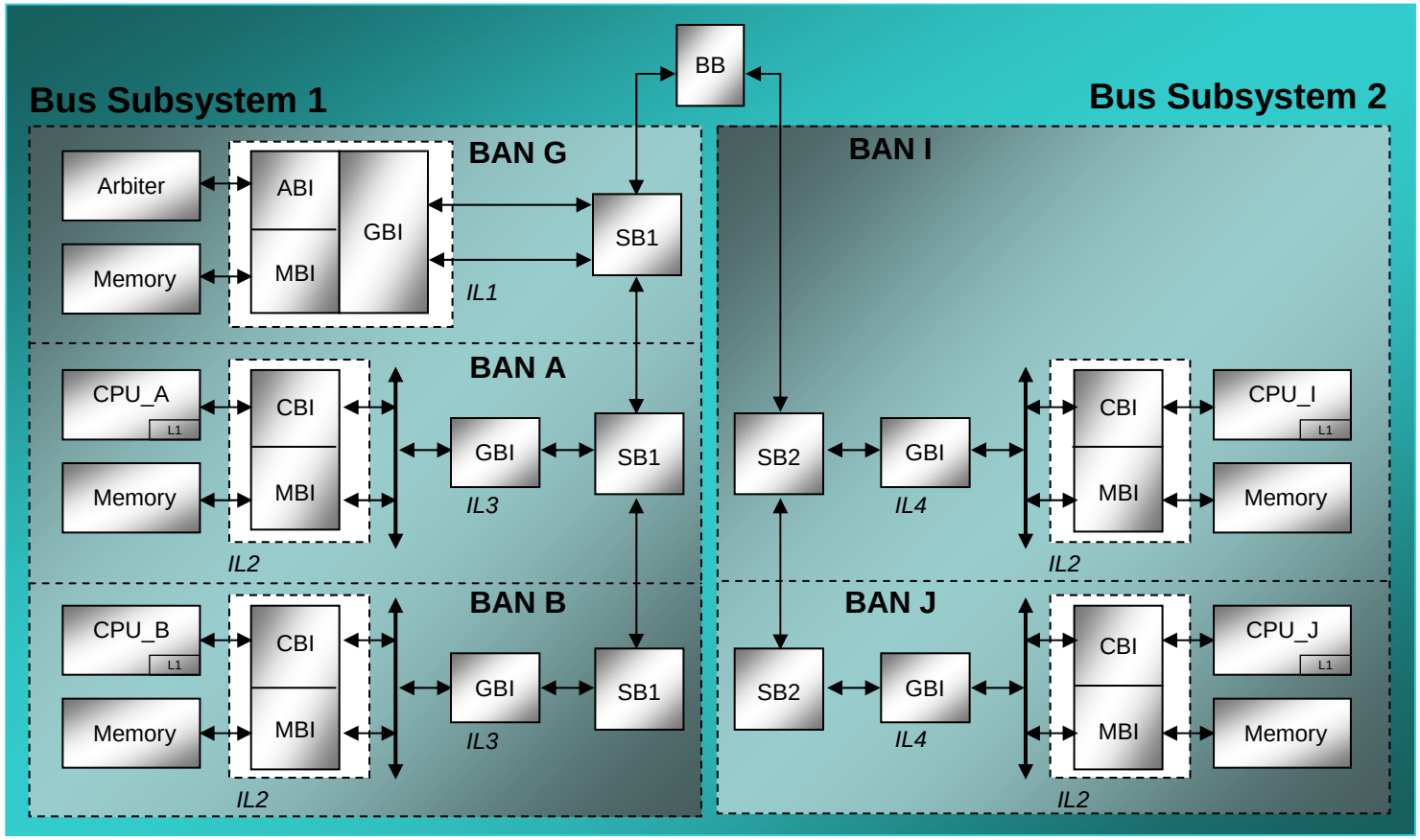
Methodology Overview

- 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



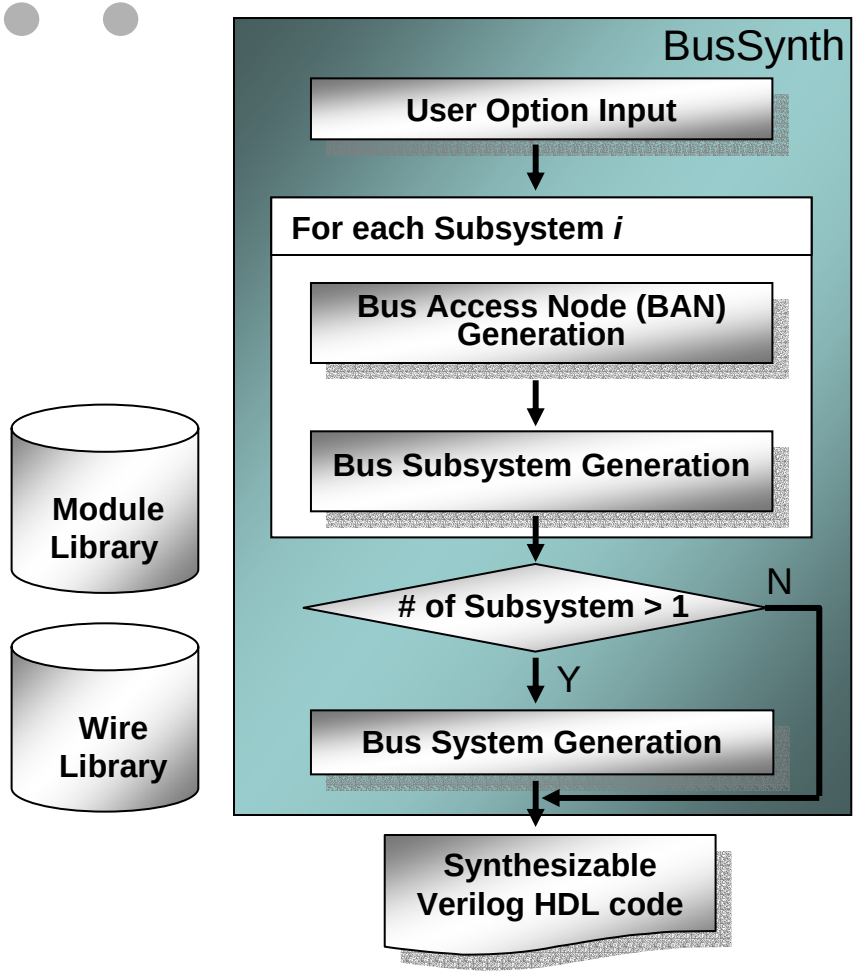
Bus System Structure – an example

Bus System

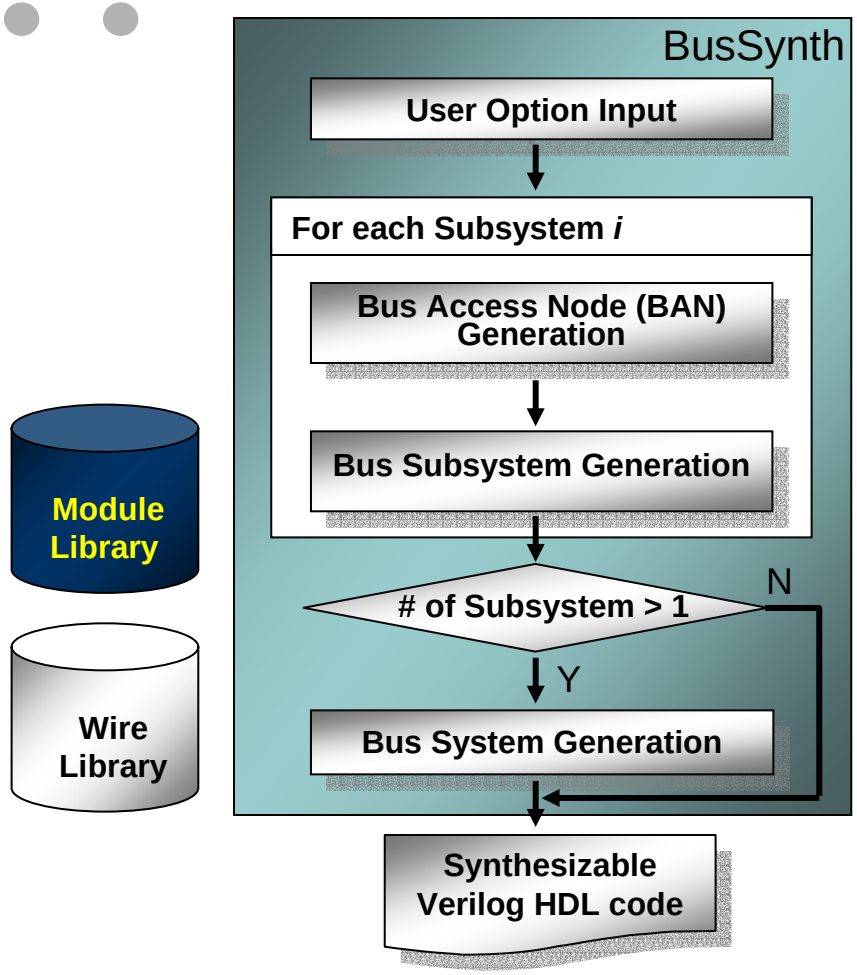


BAN: Bus Access Node, IL: Interface Logic, SB: Segment of Bus, BB: Bus Bridge, MBI: Memory-Bus Interface, CBI: CPU/PE-Bus Interface, GBI: Generic Bus Interface, ABI: Arbiter-Bus Interface,

Bus System Generation

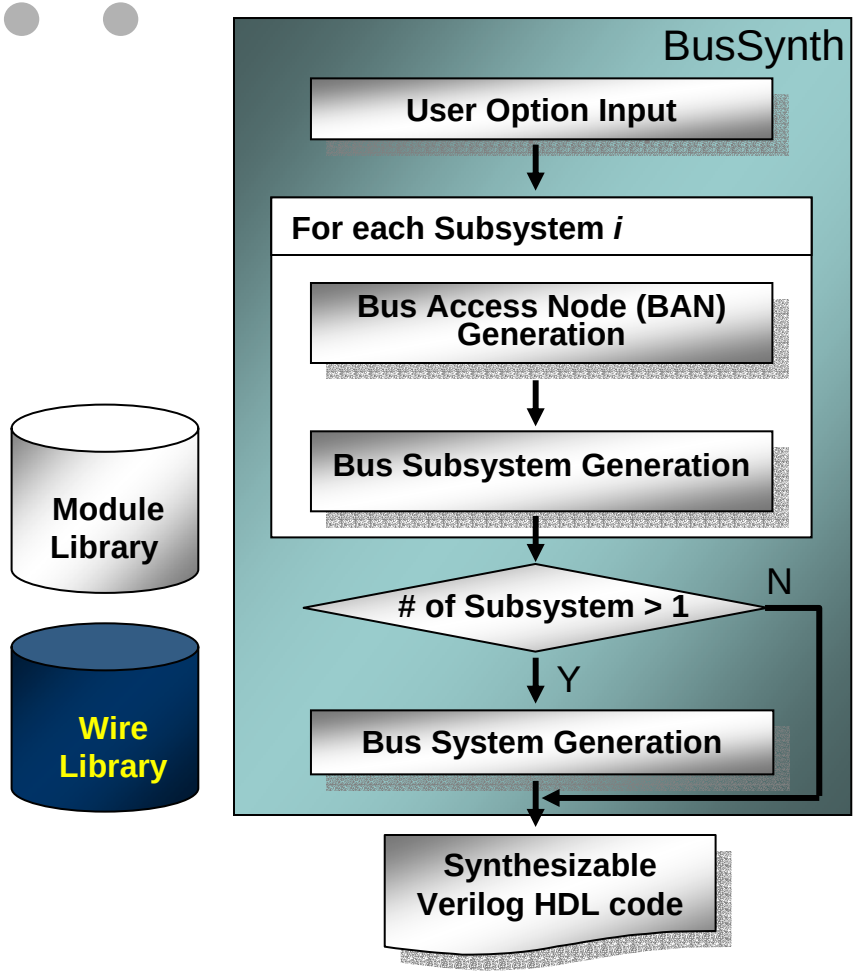


Bus System Generation



- 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]

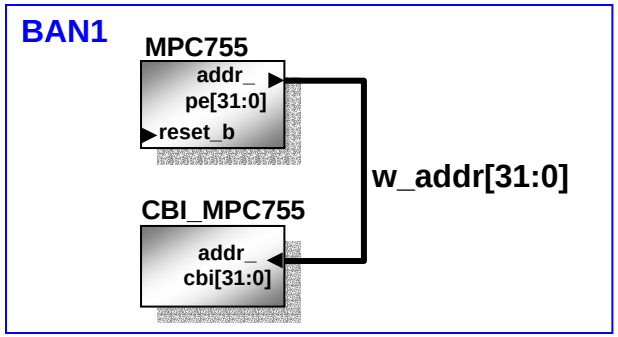
Bus System Generation (Continued)



- Wire Library
  - Format

%wire <library_name>;
w_name w_width m1_name m1_pname
m1_wmsb m1_wlsb m2_name m2_pname
m2_wmsb m2_wlsb;
%endwire;

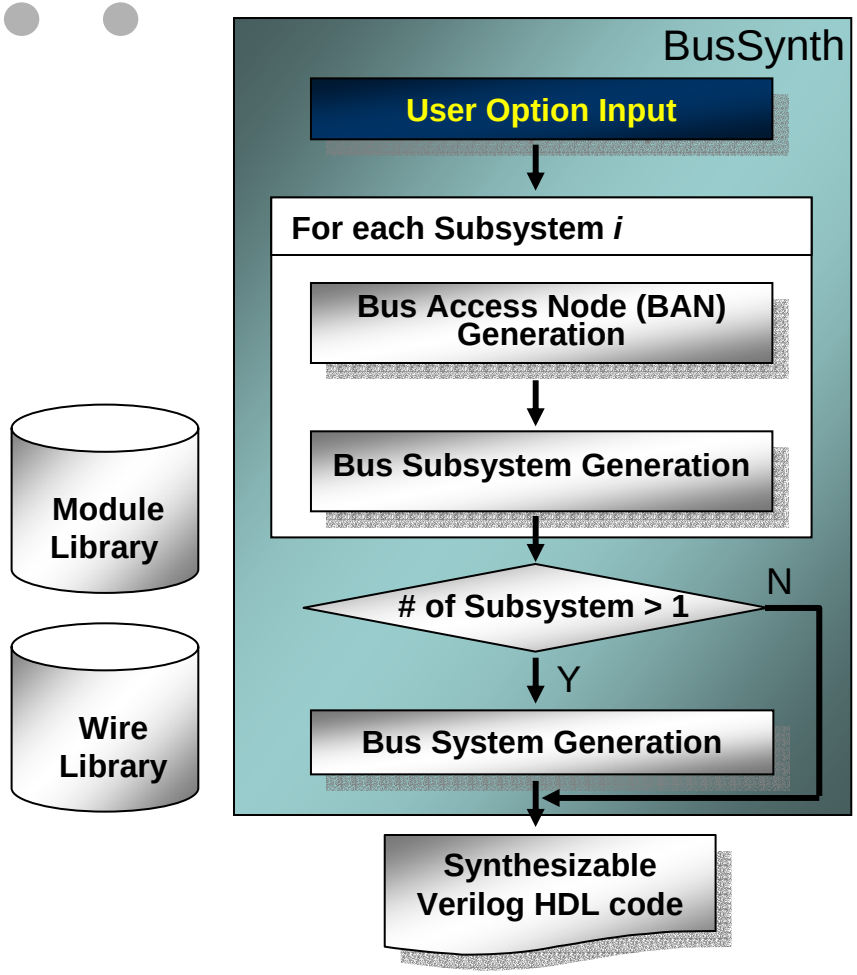
  - An example:



%wire ban1;
w_addr 32 MPC755 addr_pe 31 0
CBI_MPC755 addr_cbi 31 0;
%endwire;

Bus System Generation (Continued)

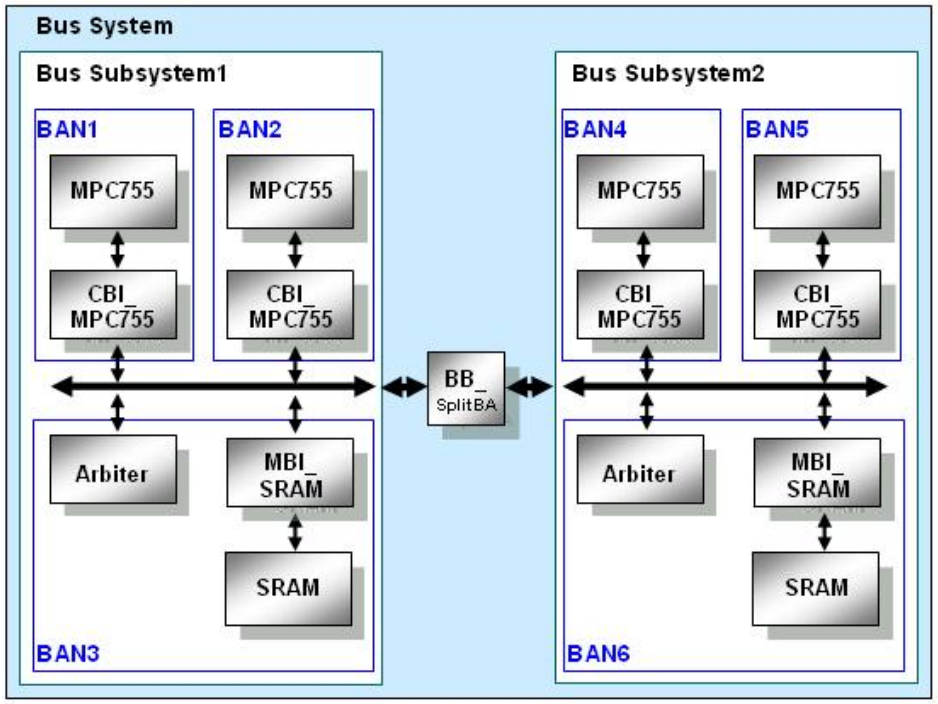
- User input list



- **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

Bus System Generation (Continued)

- Example: user input for SplitBA



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

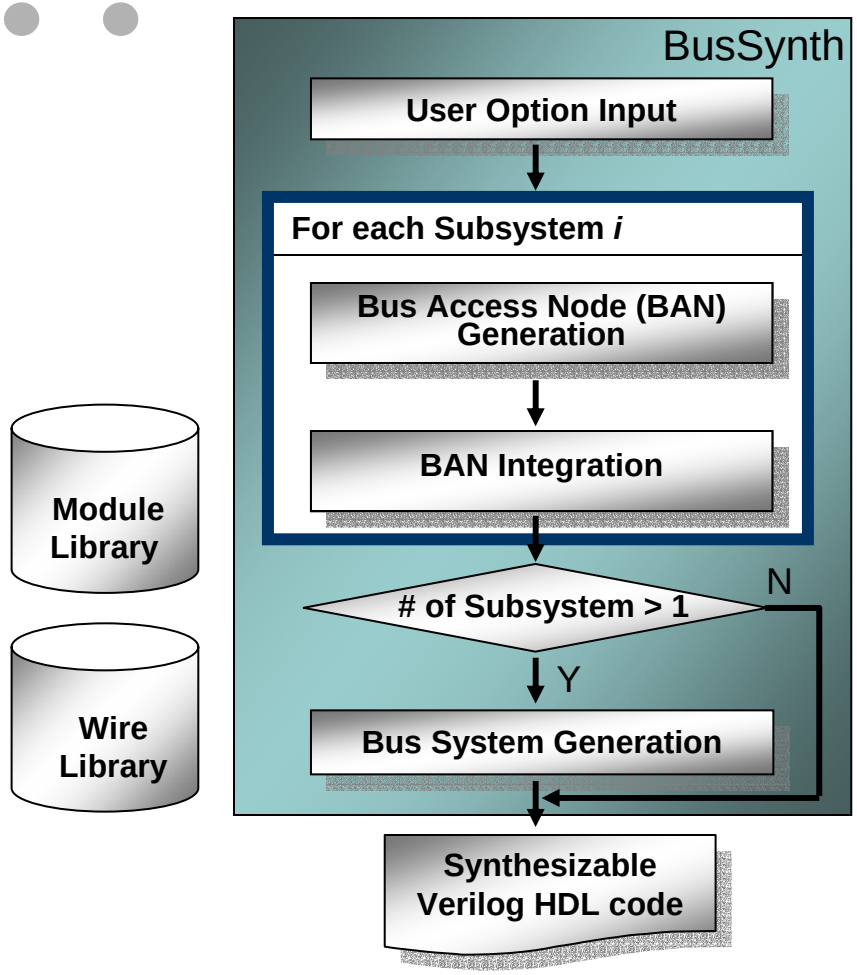
For Bus Subsystem2

- **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:

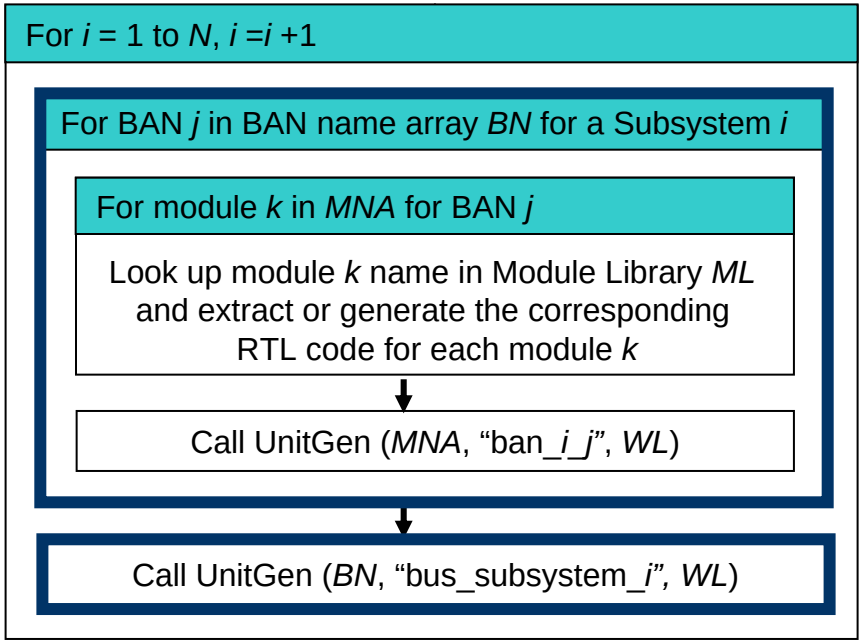
- **BAN3:** Type = SRAM, address bus width = 21 and data bus width = 64
- **BAN6:** Type = SRAM, address bus width = 21 and data bus width = 64

Bus System Generation (Continued)



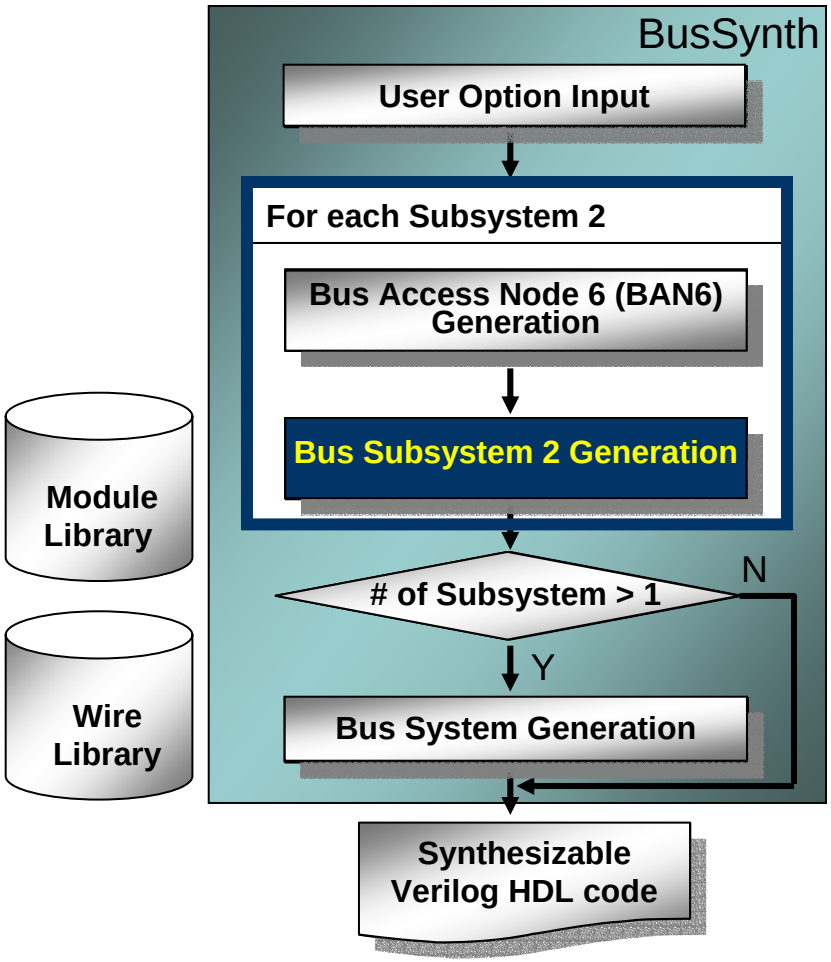
- Bus Subsystem Generation

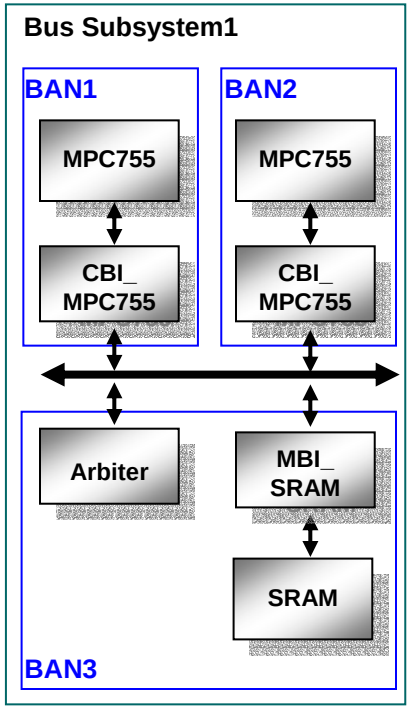
BusSubSys (module_name_array *MNA*, ban_name_array *BN*, subsys_no *N*, wire_library *WL*, module_library *ML*)

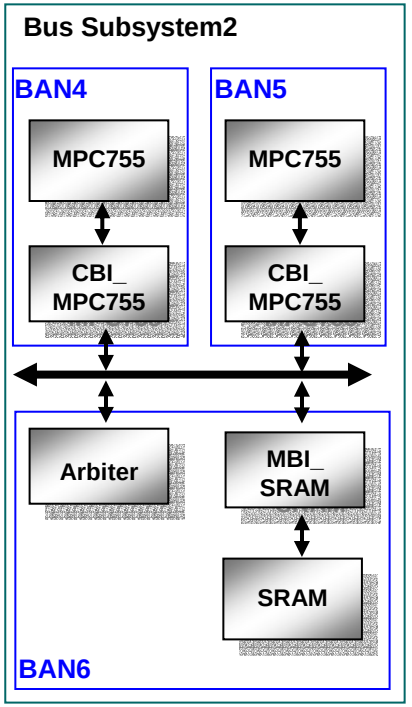


Bus System Generation (Continued)

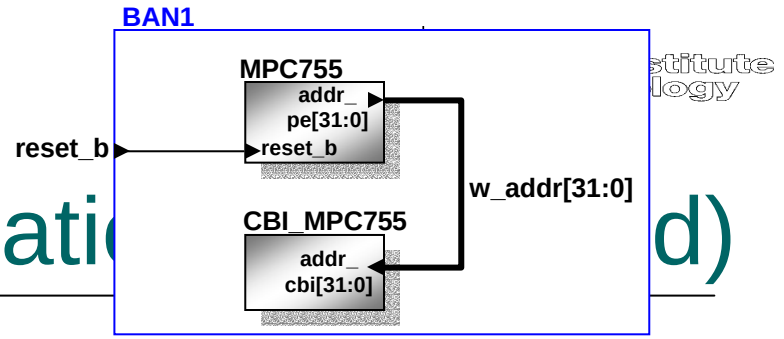
- Example: the generation of Bus Subsystems for SplitBA





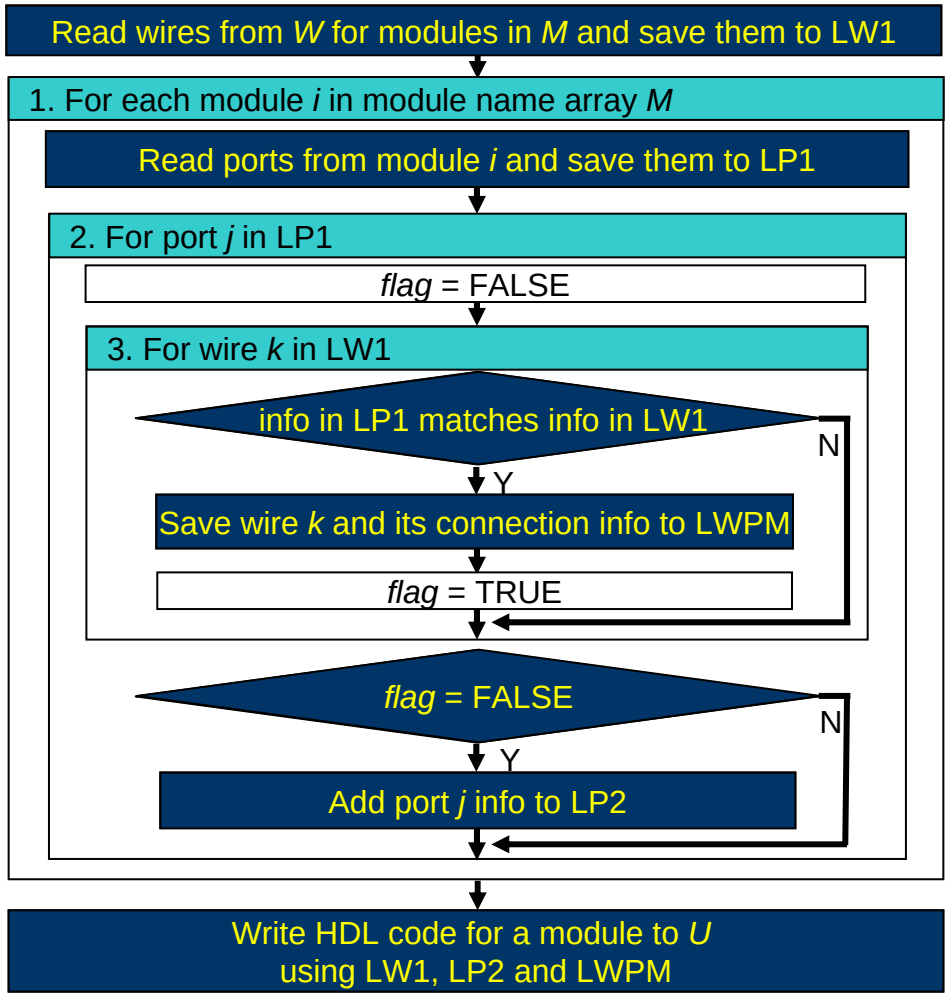


Bus System Generati d)



Unit generation

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




```
module_name_array = {"MPC755", "CBI_MPC755"}
top_unit_name = "BAN1"
```

LW1

```
w_addr 32 MPC755 addr_pe 31 0 CBI_MPC755 addr_cbi 31 0;
```

LP1 for MPC755

```
MPC755 addr_pe output 31 0;
MPC755 reset_b input 0 0;
```

LP1 for CBI_MPC755

```
CBI_MPC755 addr_cbi input 31 0;
```

LWPM

```
w_addr MPC755 addr_pe 31 0;
w_addr CBI_MPC755 addr_cbi 31 0;
```

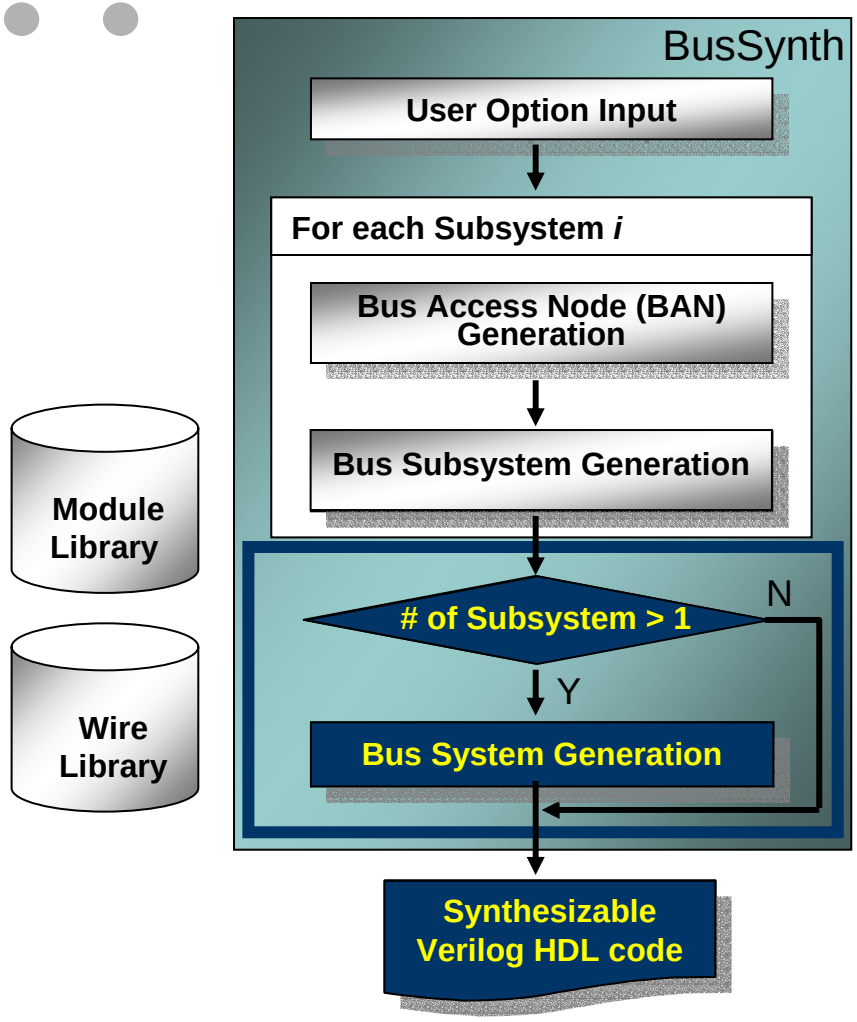
LP2

```
MPC755 reset_b input 0 0;
```

HDL code

```
module BAN1(reset_b, ….);              // "…." : skipped
input  reset_b;                        // from LP2
….
wire  w_addr[31:0];                    // from LWPM
….
MPC755  MPC755_0(
       .reset_b(reset_b),              // from LWPM
       .addr_pe(w_addr[31:0]),         // from LWPM
       ….
);
CBI_MPC755 CBI_MPC755_0(
       .addr_cbi(w_addr[31:0]),        // from LWPM
       ….
);
endmodule;
```

Bus System Generation (Continued)



- Bus System Generation

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

For module *i* in bus bridge name array *MS*

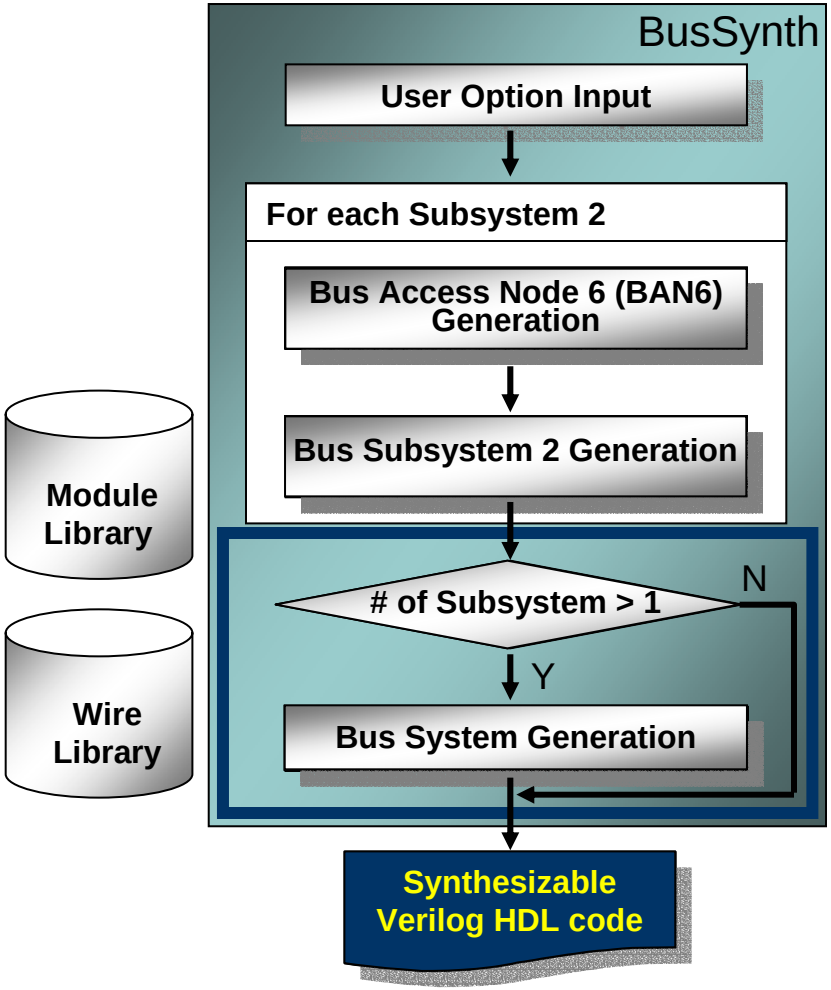
Look up module *i* name in Module Library *ML* and extract or generate the corresponding RTL code for the module *i*

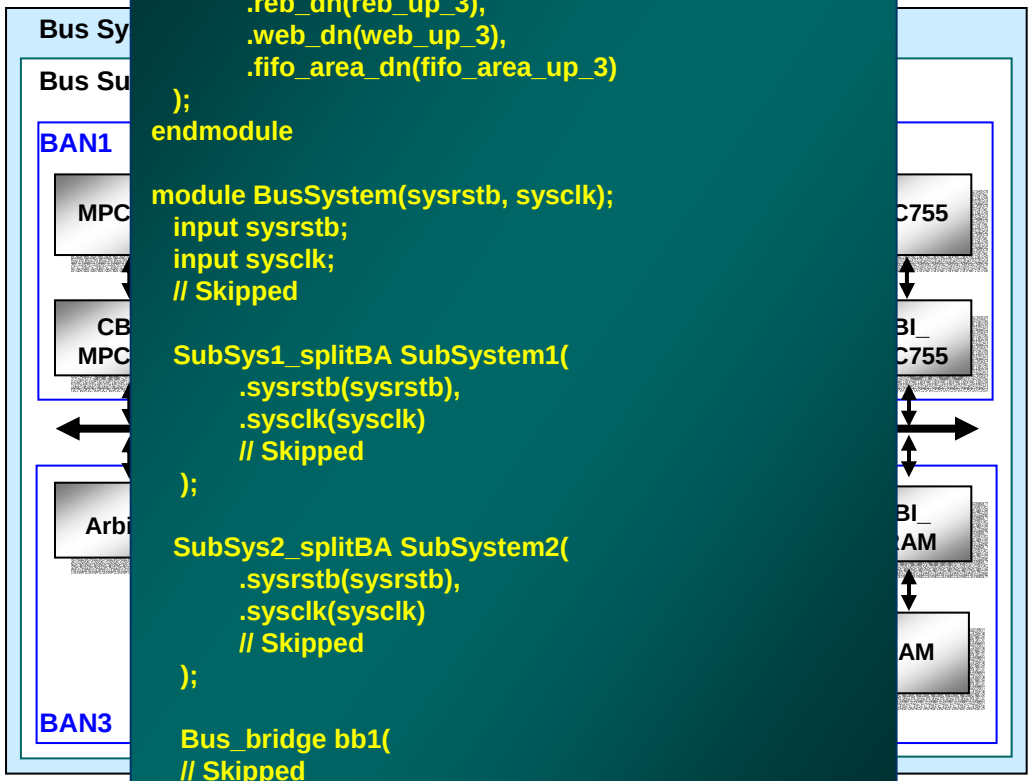
↓

Call UnitGen ({*SS*, *MS*}, “bus_system”, *WL*)

Bus System Gen(Continued)

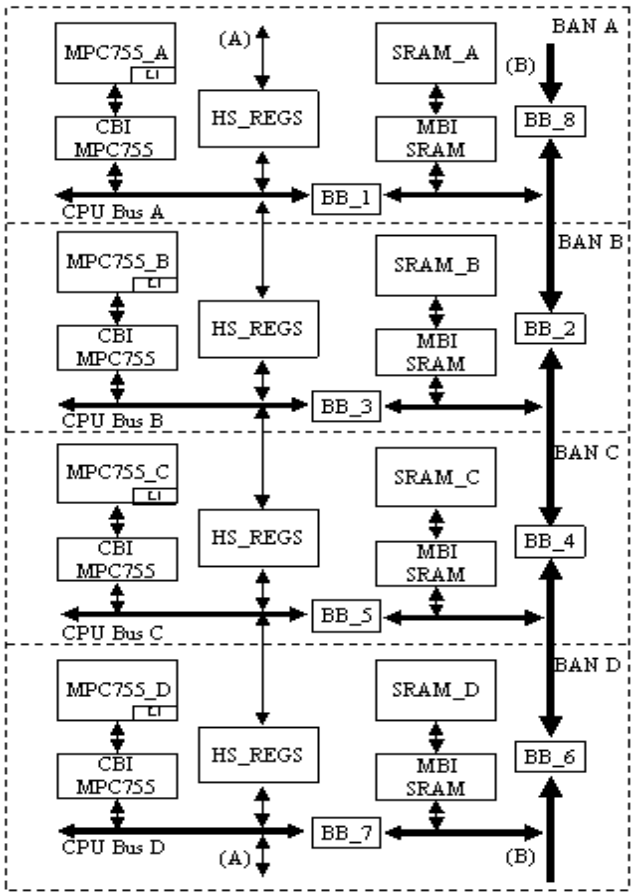
- Example: the generation of





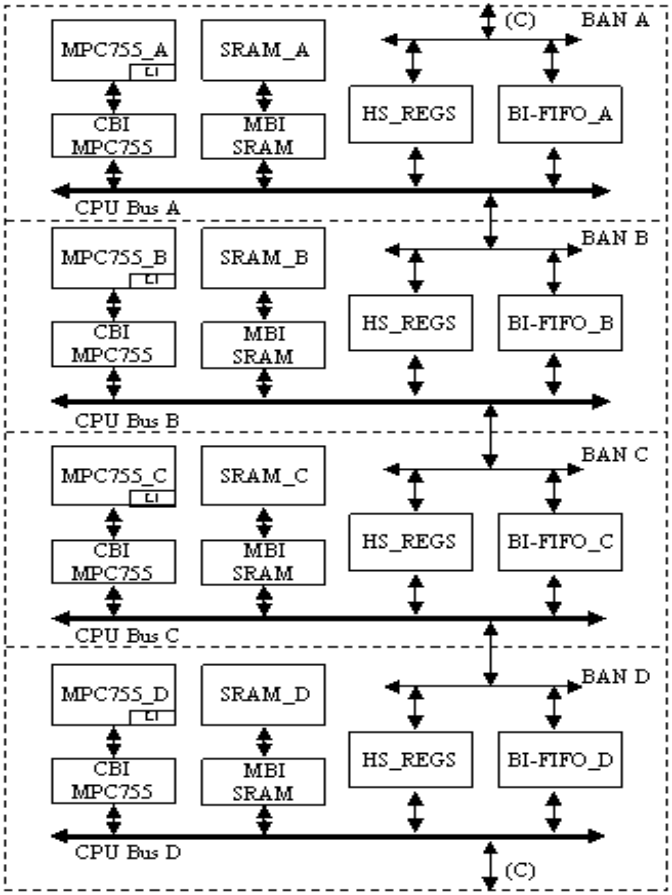

```
        //  Skipped
          .up_dataout(dataout_up_2[FIFO_D_WIDTH-1:0]),
          .up_gen_int(gen_int_up_2),
          .up_isr0_ctlhi(isr0_ctlhi_up_2),
          .up_isr0_ctllo(isr0_ctllo_up_2),
          .dn_datain(datain_up_3[FIFO_D_WIDTH-1:0]),
          .reb_dn(reb_up_3),
          .web_dn(web_up_3),
          .fifo_area_dn(fifo_area_up_3)
    );
endmodule

module BusSystem(sysrstb, sysclk);
    input sysrstb;
    input sysclk;
    // Skipped

    SubSys1_splitBA SubSystem1(
            .sysrstb(sysrstb),
            .sysclk(sysclk)
            // Skipped
     );

    SubSys2_splitBA SubSystem2(
            .sysrstb(sysrstb),
            .sysclk(sysclk)
            // Skipped
     );

     Bus_bridge bb1(
     // Skipped
     );

endmodule
```


Bus System Examples



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

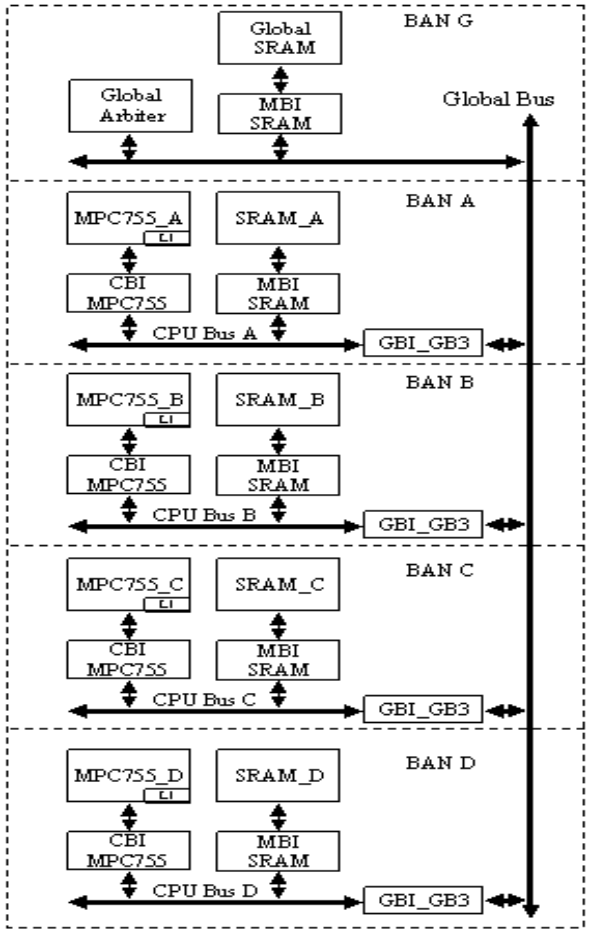
General Global Bus Architecture Version I (GBAVI)



[Note] HS_REGS: handshake registers

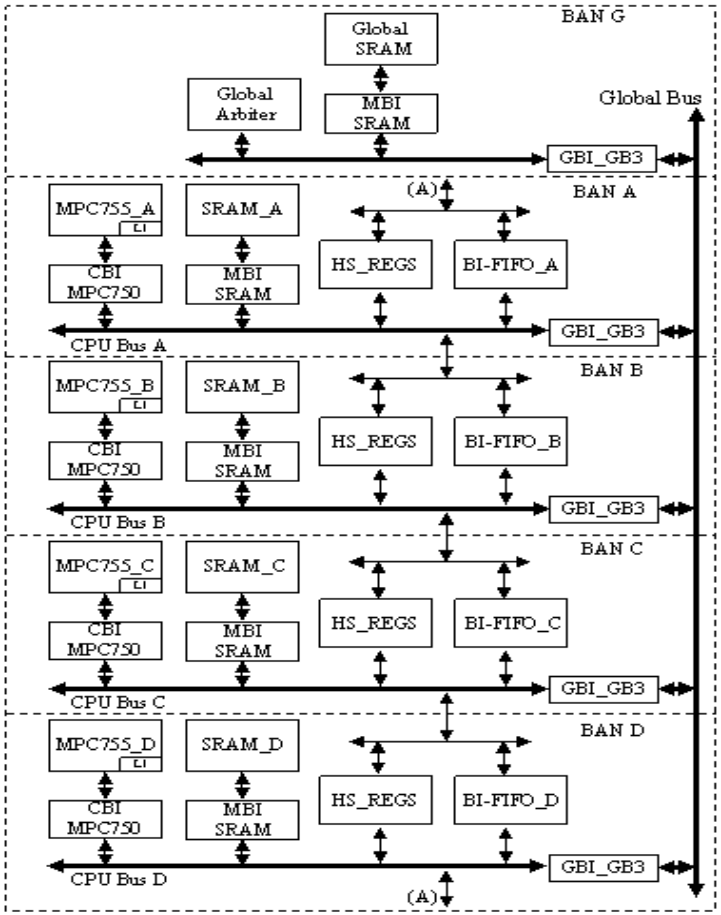
Bi-FIFO Bus Architecture (BFBA)

Bus System Examples (Continued)



[Note] HS_REGS: handshake registers

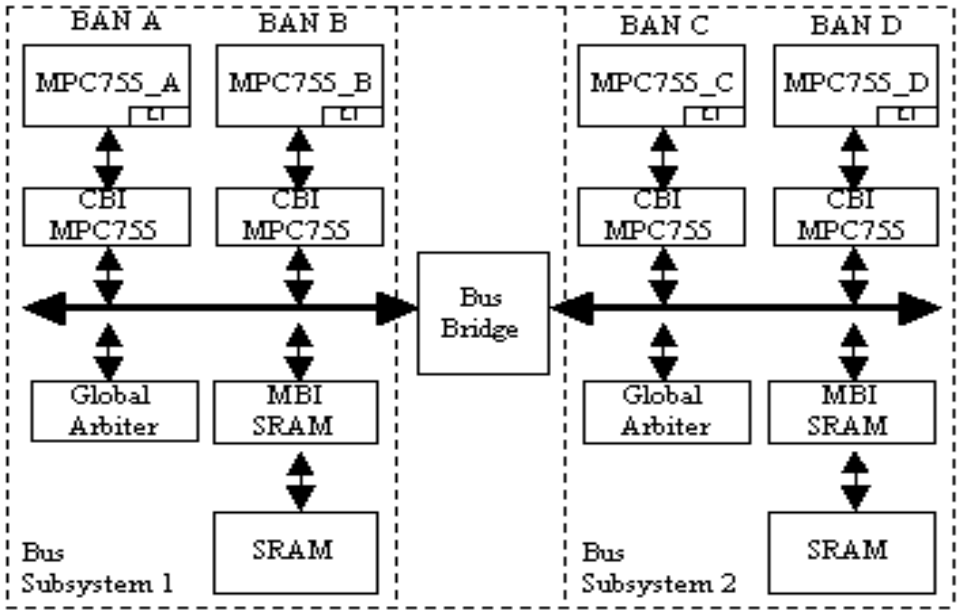
General Global Bus Architecture Version III (GBAVIII)



[Note] HS_REGS: handshake registers

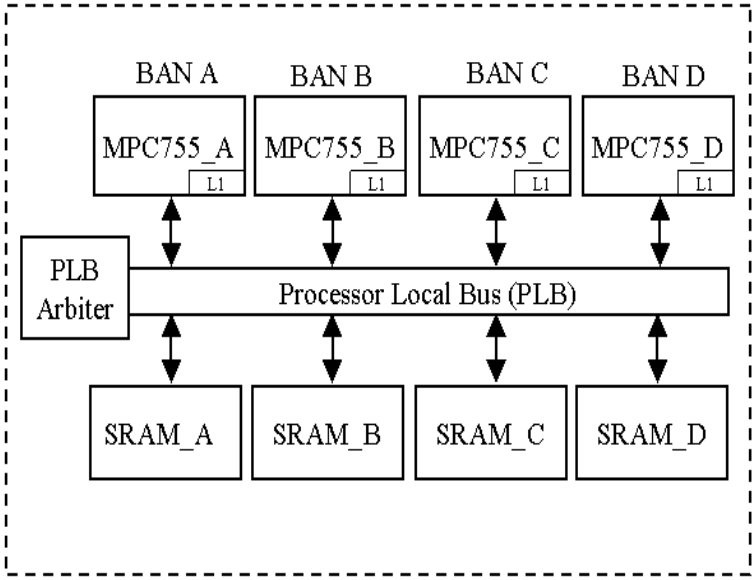
Hybrid Bus Architecture (HybridBA)

Bus System Examples (Continued)

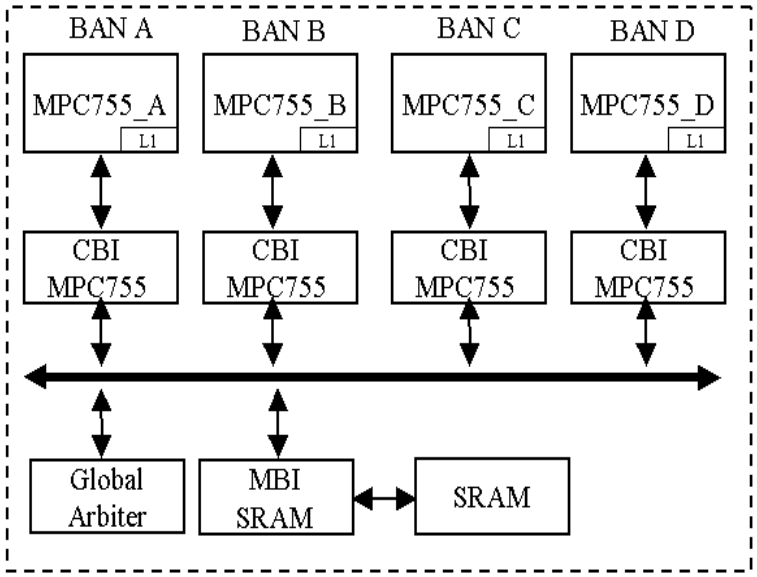


Split Bus Architecture (SplitBA)

Bus System Examples (Continued)



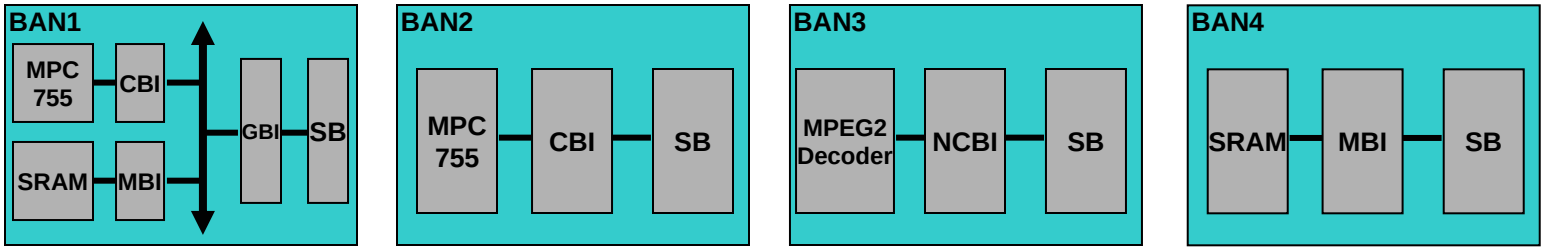
CoreConnect Bus Architecture (CCBA)



General Global Bus Architecture (GGBA)

A New Bus System Generation

- Different Combination of Bus Components
  - Different combination of BAN components



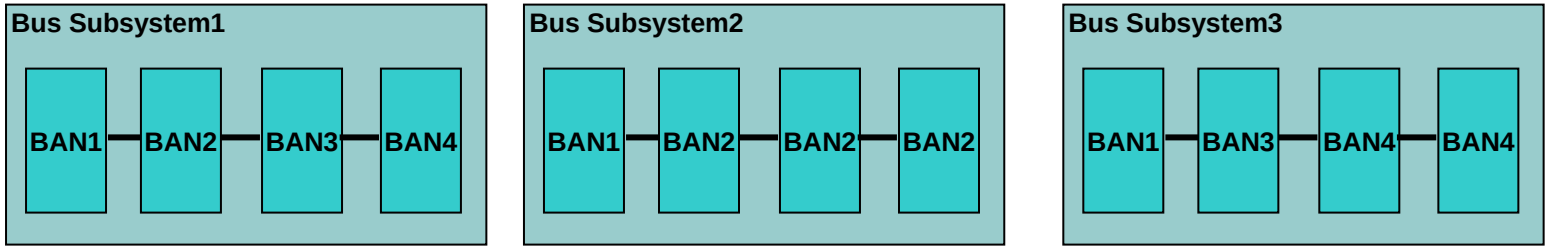
User Inputs for BAN1:
CPU type: MPC755
Non-CPU type: None
# of global memories: 0
# of local memories: 1
Memory type: SRAM

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

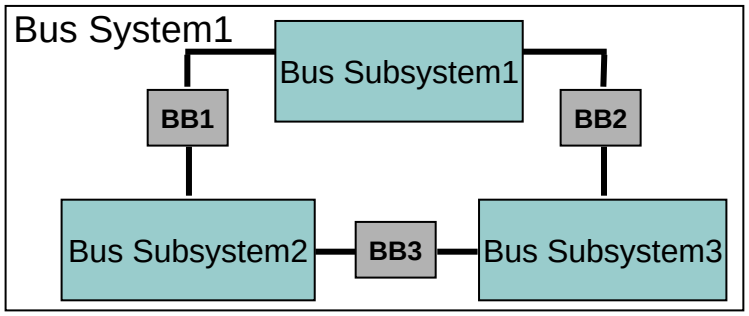
  - Different combination of BANs

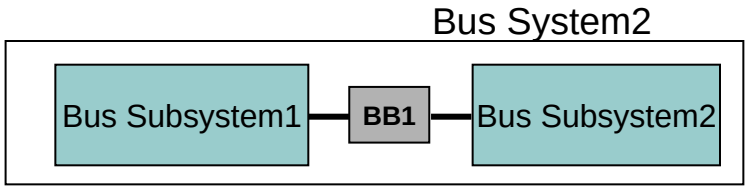


Note: BAN: Bus Access Node, MBI: Memory Bus Interface, CBI: CPU Bus Interface, GBI: Generic Bus Interface, SB: Segment of Bus, NCBI: Non-CPU Bus Interface

A New Bus System Generation (Continued)

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





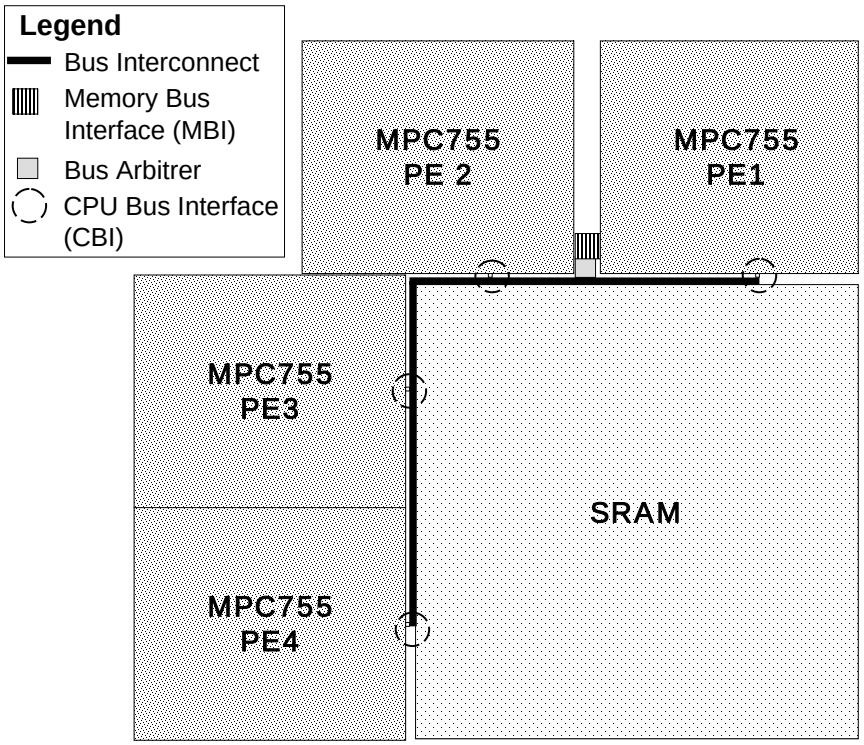
User Inputs for Bus System1:
# of Bus Subsystems: 3
Bus Subsystems 1, 2, 3

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)



(a) Estimated Floorplan of GGBA

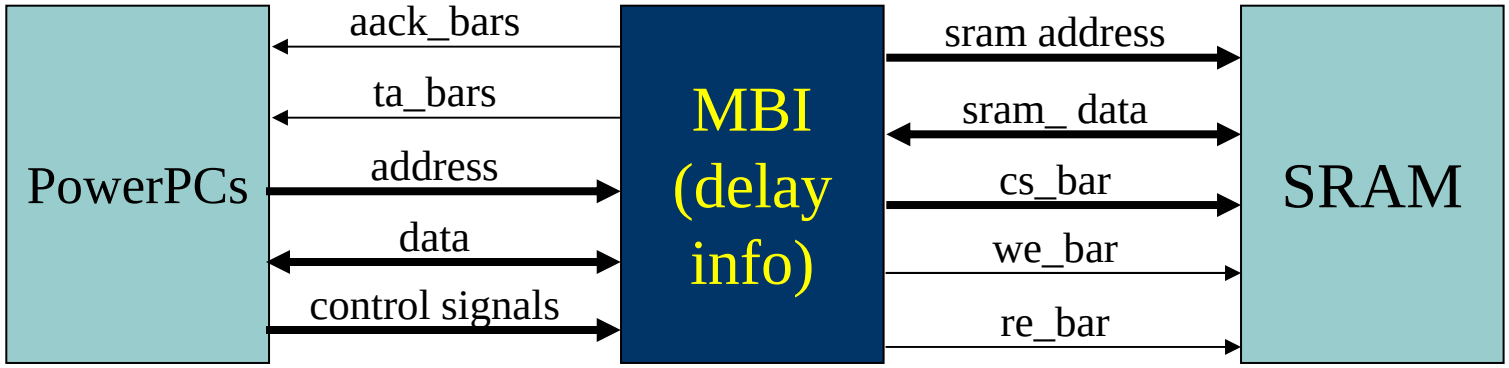
- HSPICE wire model includes:
  - RLC parameters from MOSIS run for TSMC 0.25 um
  - Interconnect length
- Interconnect delay calculation

	SRAM Memory	
	Length [cm]	Delay [ns]
Processing Element 1	0.2521	0.2848
Processing Element 2	0.6143	0.5727
Processing Element 3	1.2753	2.2882
Processing Element 4	1.9363	3.0472

(b) Interconnect length estimation

Interconnect Delay Aware Bus System Generation (Continued)

- 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

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

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

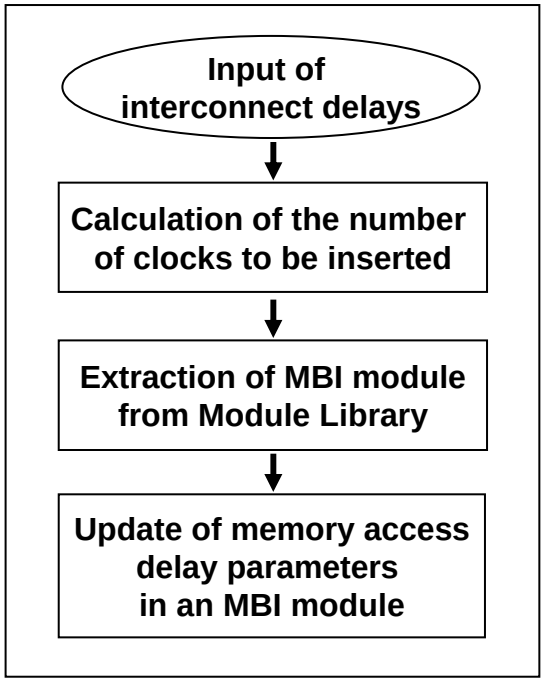
(a) Estimated total delay of paths between each PE and a shared memory

	Number of clock delays in each PE for a read operation [clock]		
	100 MHz (10.00ns) system clock	200 MHz (5.00ns) system clock	300 MHz (3.33ns) system clock
PE 1	1 (0.8570)	2 (1.7139)	3 (2.57345)
PE 2	1 (0.9145)	2 (1.8291)	3 (2.74636)
PE 3	2 (1.2576)	3 (2.5153)	4 (3.77669)
PE 4	2 (1.4094)	3 (2.8189)	5 (4.23255)

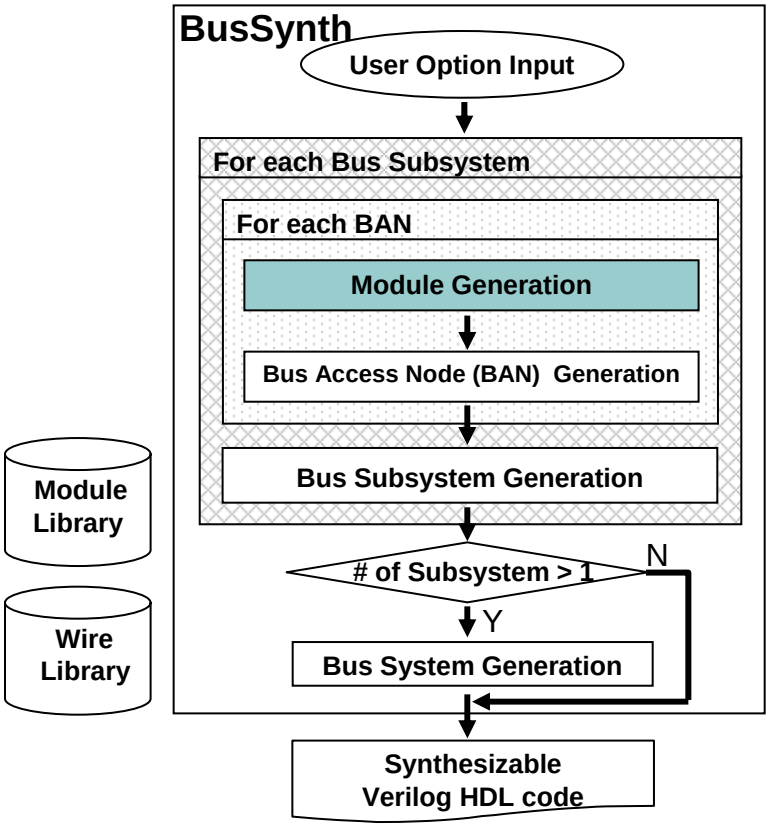
(b) Number of clock delays in data paths

Interconnect Delay Aware Bus System Generation (Continued)

- Memory Bus Interface (MBI) module generation



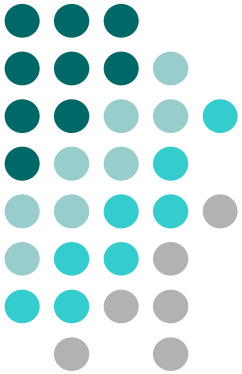
(a) Sequence of MBI Generation



(b) Bus System Generation

Outline

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

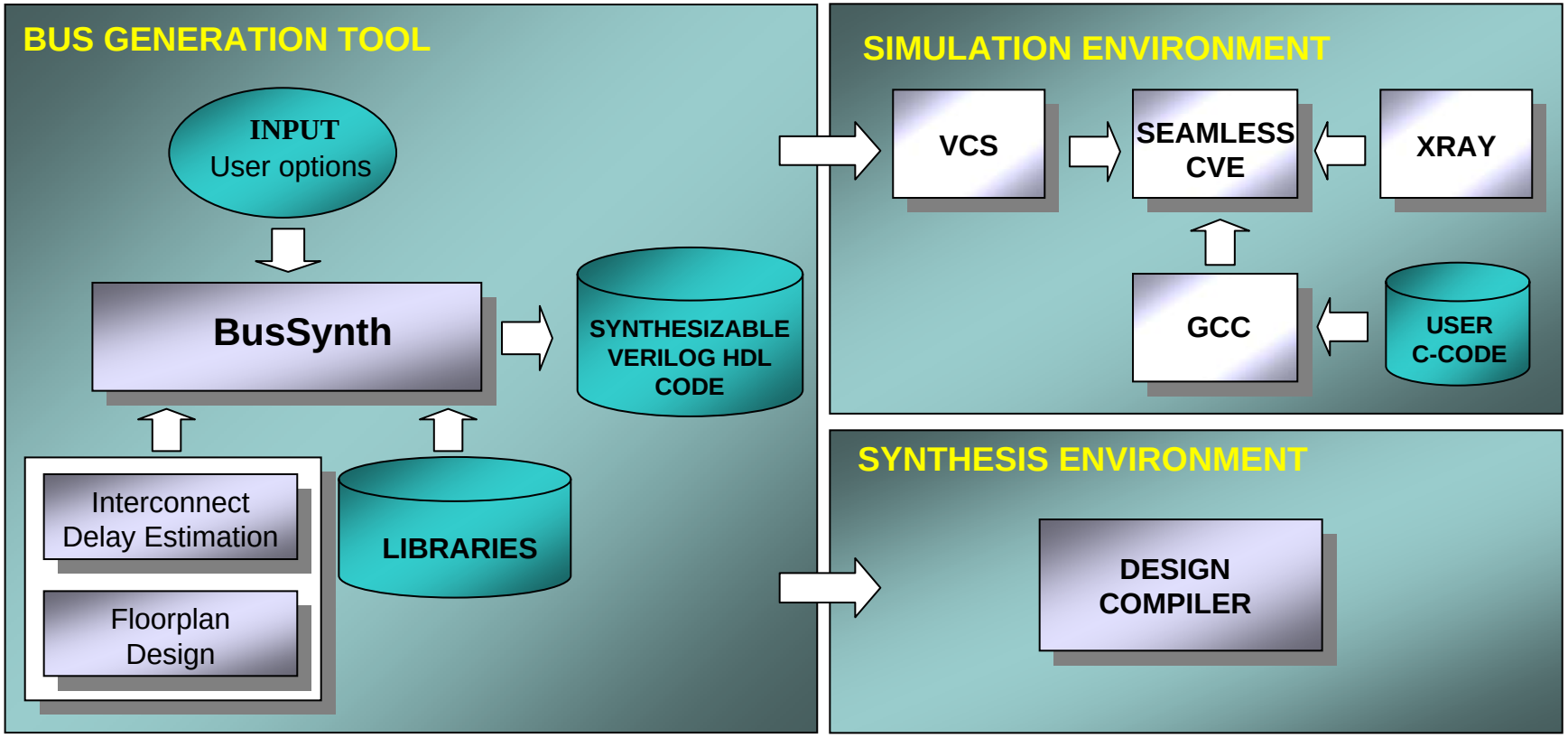


Georgia Institute of Technology

Application Examples

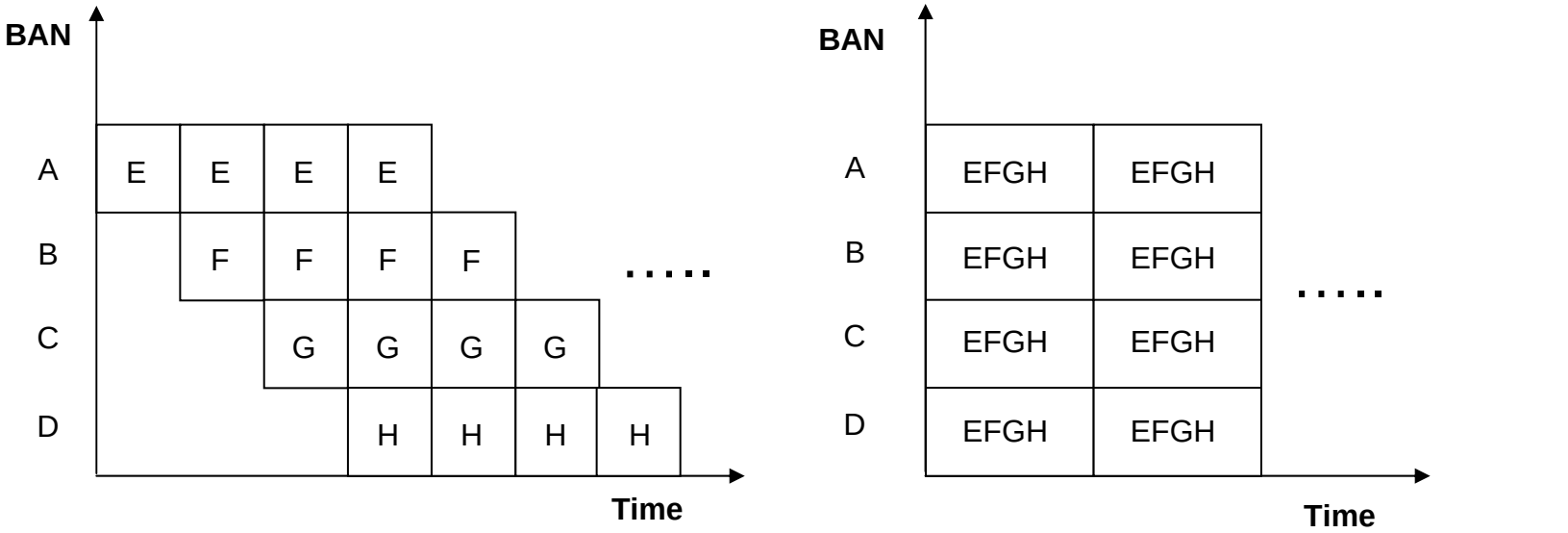
- OFDM transmitter
  - Wireless application
  - One packet: (2048+512)-complex samples
- MPEG2 decoder
  - A video stream decoder
- Database example
  - Multitask clients and server over PEs: total 41 tasks over four PEs
  - RTOS: Atalanta version 0.4

Experimental Setup



Note: VCS and Design Compiler from Synopsys, Seamless CVE and Xray from Mentor Graphics and GCC from GNU

Software Programming Style



(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

Case	Bus System	Application Throughput [Mbps]	Software Programming Style
1	BFBA	2.6504	PPA
2	GBAVI	2.1087	PPA
3	GBAVIII	4.5599	FPA
4		2.2567	PPA
5	HybridBA	4.5599	FPA
6		2.6504	PPA
7	SplitBA	5.1132	FPA
8	GGBA	4.3913	FPA
9		2.1880	PPA

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)

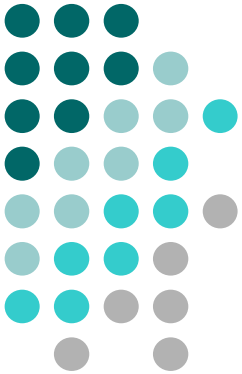
Case	Bus System	Application Throughput [Mbps]	Software Programming Style
10	BFBA	0.8594	FPA
11	GBAVI	0.8271	FPA
12	GBAVIII	1.1444	FPA
13	HybridBA	1.1650	FPA
14	CCBA	1.0083	FPA

Note: Picture size: 16 x 16

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

Case	Bus System	Execution Time [ns]	Software Programming Style
15	GGBA	2,241,100	FPA
16	SplitBA	1,317,804	FPA

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



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

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

(a) 300MHz Bus Clock

GGBA System	Execution Time [ns/packet]	Comparison I [increase in execution time]	Comparison II [decrease in execution time]
1. GGBA I (no interconnect delay)	1,825,751	0.0%	-
2. GGBA II (2, 2, 3 and 3 clock delays in each data path from PE 1 to PE 4)	2,323,670	27.3%	27.4%
3. GGBA III (3 clock delays in all data paths)	3,198,620	75.2%	0.0%

(b) 200MHz Bus Clock

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

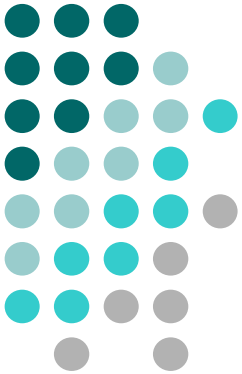
(c) 100MHz Bus Clock

Generation Time and Logic Area (no wires)

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

Bus System	1 processor		8 processors		16 processors		24 processors	
	Time [ms]	Gate count	Time [ms]	Gate count	Time [ms]	Gate count	Time [ms]	Gate counts
BFBA	509	800	534	6,401	546	12,793	578	19,188
GBAVI	417	872	432	6,899	457	13,751	506	21,256
GBAVIII	513	2,070	542	14,746	563	30,798	590	48,395
HybridBA	763	2,973	859	21,869	928	44,847	983	69,697
SplitBA	N/A	N/A	413	4,297	440	8,605	491	16,110

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



Conclusions

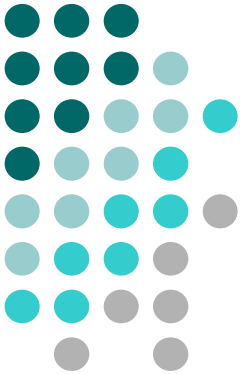
- SoC bus system design aid
  - Expert guide to design an SoC bus system
- Automated bus generation tool: BusSynth
  - Solution: how to easily and quickly design a multi-processor SoC bus system
  - User option based tool that generates diverse custom bus systems
  - Synthesizable Verilog HDL output
- 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

- K. Ryu and V. Mooney, “Automated Bus Generation for Multiprocessor SoC Design,” to appear in *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems* (TCAD’04), 2004.
- K. Ryu, A. Talpasanu, V. Mooney and J. Davis, “Interconnect Delay Aware RTL Verilog Bus Architecture Generation for an SoC,” to appear in *Proceeding of IEEE Asia-Pacific Conference on Advanced System Integrated Circuits* (AP-ASIC’04), August 2004.
- K. Ryu and V. Mooney, “Automated Bus Generation for Multiprocessor SoC Design,” in *Proceedings of the Design, Automation and Test in Europe* (DATE'03), pp. 282-287, March 2003.
- K. Ryu and V. Mooney, “Automated Bus Generation for Multiprocessor SoC Design,” [Online]. Available: http://www.cc.gatech.edu/tech_reports, Georgia Institute of Technology, Atlanta, GA, Technical Report GIT-CC-02-64, December 2002.
- K. Ryu, E. Shin and V. Mooney, "A Comparison of Five Different Multiprocessor SoC Bus Architectures," in *Proceedings of the EUROMICRO Symposium on Digital Systems Design* (EUROMICRO'01), pp. 202-209, September 2001.
- J. Lee, K. Ryu and V. Mooney, "A Framework for Automatic Generation of Configuration Files for a Custom Hardware/Software RTOS," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms* (ERSA'02), pp. 31-37, June 2002.

Poster Presentation and Demonstration

- K. Ryu and V. Mooney, “Automated Bus Generation for Multiprocessor SoC design,” Ph.D. Forum at the 40th Design Automation Conference (DAC’03), June 2003.
- K. Ryu, E. Shin, J. Lee and V. Mooney, “A Framework for Automatic Generation of Bus Systems and a Hw/Sw RTOS for Multiprocessor SoC,” University Booth at the 39th Design Automation Conference (DAC’02), June 2002.



Georgia Institute of Technology

Thank you

