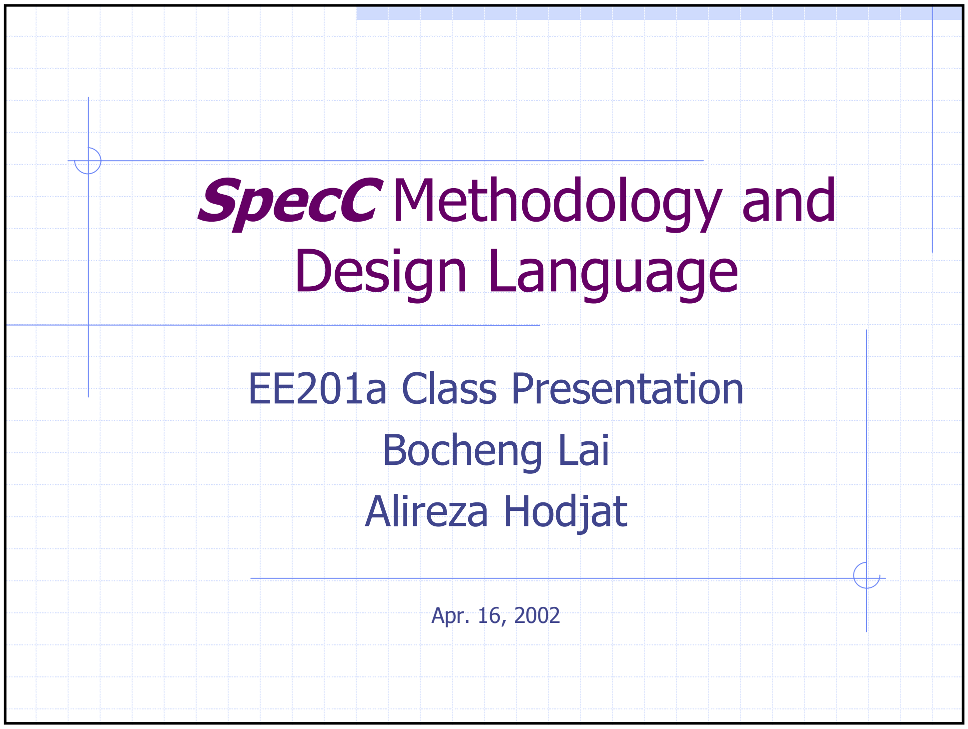


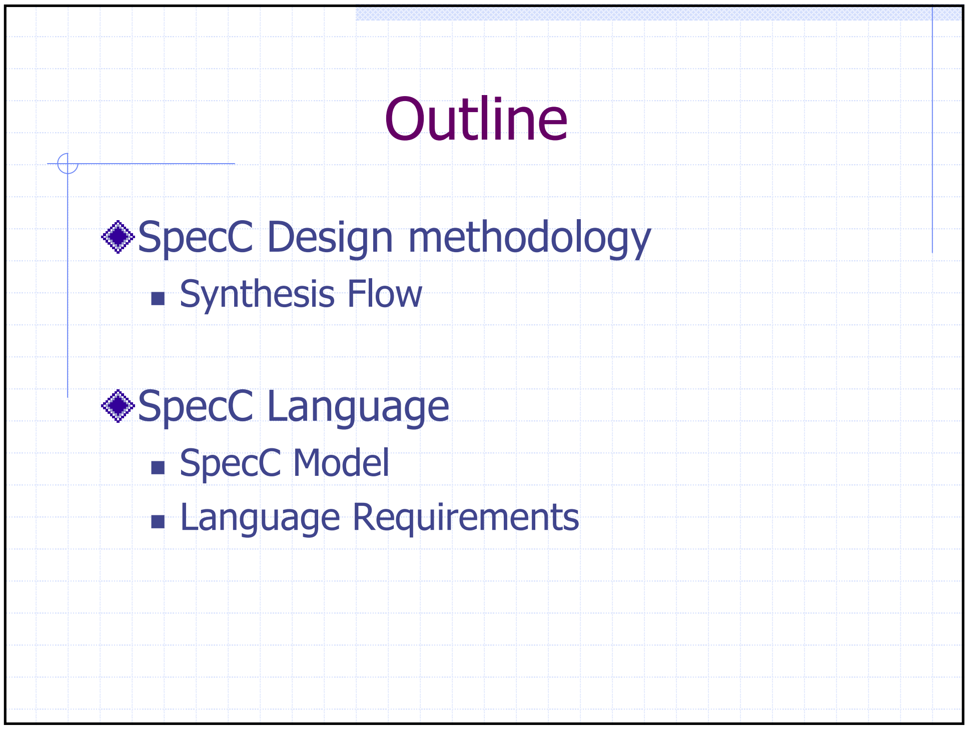
# *SpecC* Methodology and Design Language

EE201a Class Presentation

Bocheng Lai

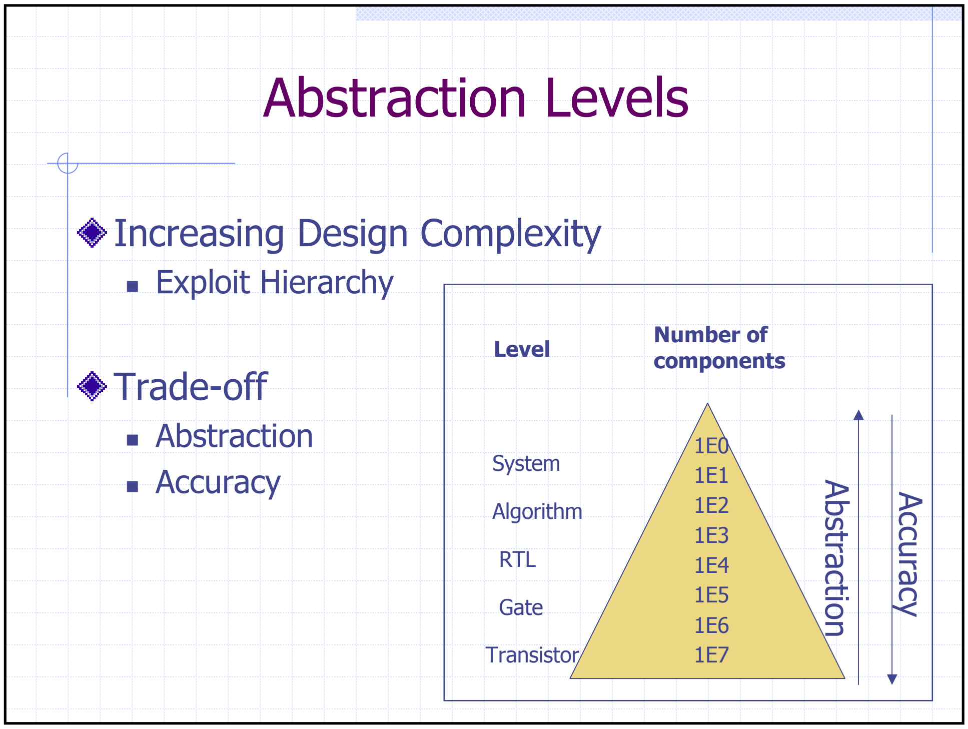
Alireza Hodjat

Apr. 16, 2002



# Outline

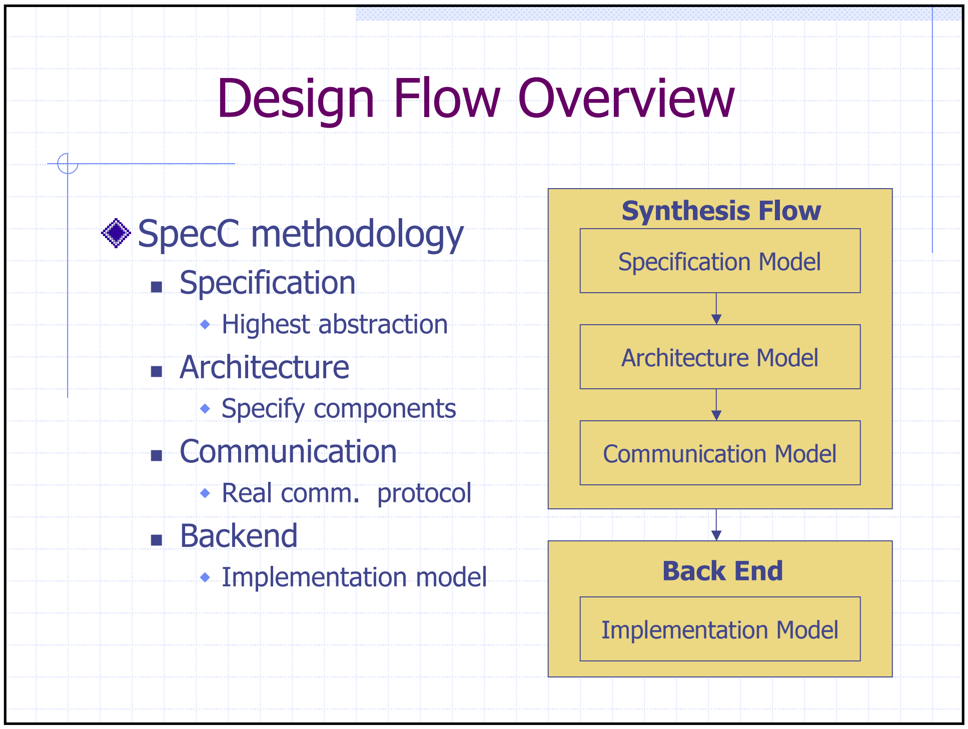
- SpecC Design methodology
  - Synthesis Flow
- SpecC Language
  - SpecC Model
  - Language Requirements



# Abstraction Levels

- Increasing Design Complexity
  - Exploit Hierarchy
- Trade-off
  - Abstraction
  - Accuracy

| Level | Number of components |
|---|---|
|  | 1E0 |
| System | 1E1 |
| Algorithm | 1E2 |
|  | 1E3 |
| RTL | 1E4 |
|  | 1E5 |
| Gate | 1E6 |
| Transistor | 1E7 |

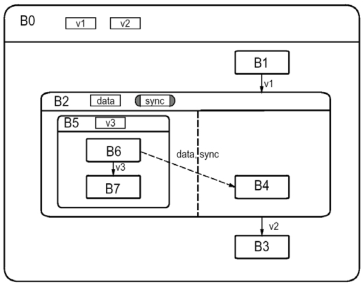


# Design Flow Overview

- SpecC methodology
  - Specification
    - Highest abstraction
  - Architecture
    - Specify components
  - Communication
    - Real comm.  protocol
  - Backend
    - Implementation model

# Specification Capture

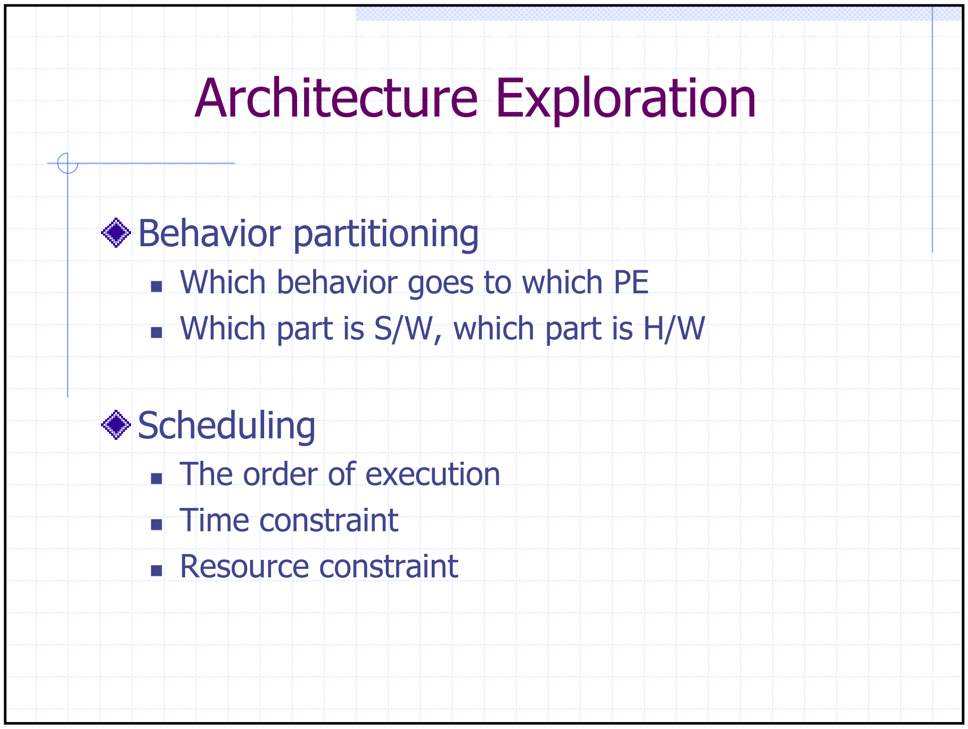
- Highest Level of Abstraction
  - Pure functionality, no structure or timing
- Guidelines
  - Separate communication and computation
  - Expose parallelism
  - Use hierarchy
  - Choose proper granularity
  - Identify system states



# Architecture Exploration

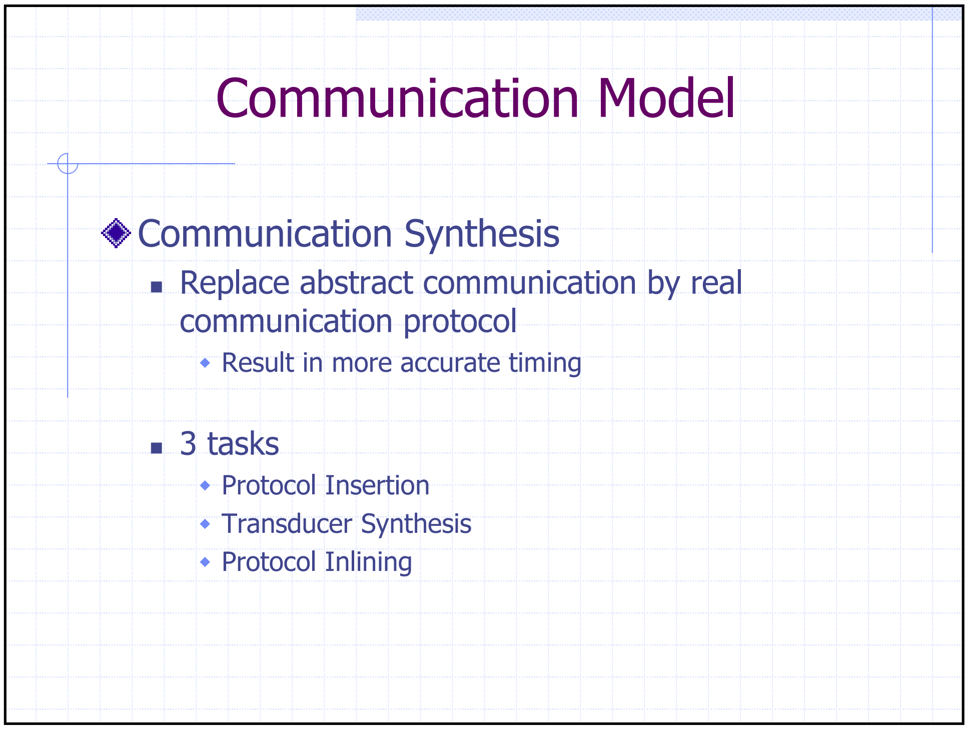
- Architecture exploration
  - Allocation
  - Behavior partitioning
  - Scheduling

- Allocation
  - Select components from library (3 types)
    - PE, memories, busses
    - Pre-designed IP components can be selected



# Architecture Exploration

- Behavior partitioning
  - Which behavior goes to which PE
  - Which part is S/W, which part is H/W
- Scheduling
  - The order of execution
  - Time constraint
  - Resource constraint



# Communication Model

- Communication Synthesis
  - Replace abstract communication by real communication protocol
    - Result in more accurate timing
  - 3 tasks
    - Protocol Insertion
    - Transducer Synthesis
    - Protocol Inlining

# Communication Model

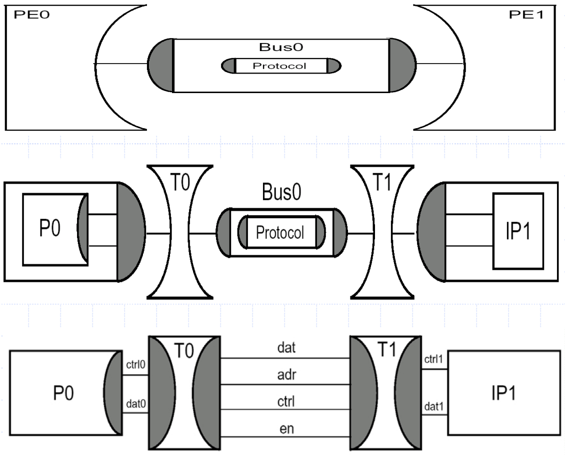
- ◈ Protocol Insertion
  - Replace virtual busses with real busses and protocol
- ◈ Transducer Synthesis
  - Bridge the gap between different protocols
- ◈ Protocol Inlining
  - Methods in channels are moved into the connected behaviors
  - Exposes wires of the system busses

# Communication Model

◈Protocol Insertion

◈Transducer

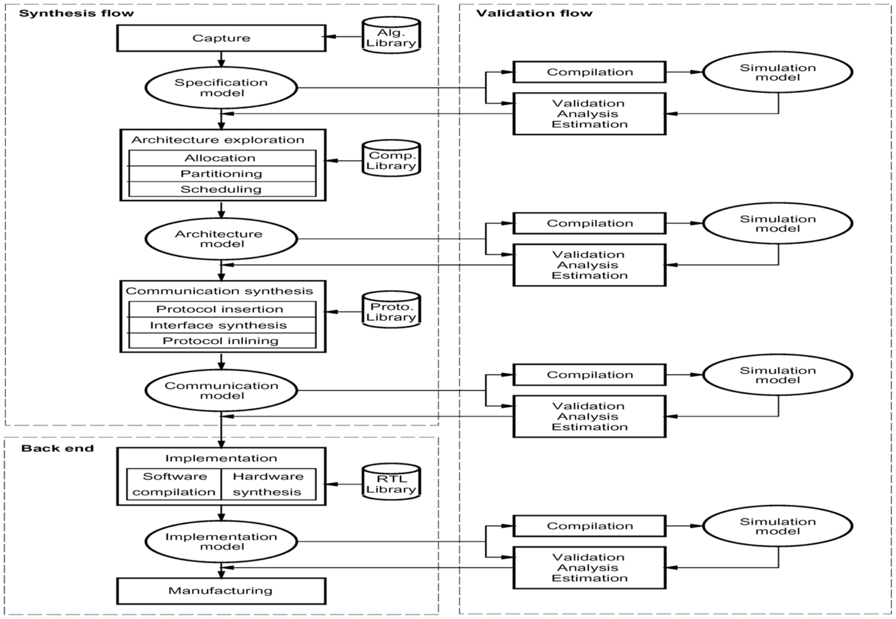
◈Protocol Inlining



# Backend

- Implementation model
  - Lowest level of abstraction
    - Compile the software part for each processor
    - Generate RTL description for hardware
  - Cycle accurate timing for the entire system
  - It is ready for manufacturing!!

# Summary



Synthesis flow

Capture

Alg. Library

Specification model

Architecture exploration

Allocation

Partitioning

Scheduling

Comp. Library

Architecture model

Communication synthesis

Protocol insertion

Interface synthesis

Protocol inlining

Proto. Library

Communication model

Back end

Implementation

Software compilation

Hardware synthesis

RTL Library

Implementation model

Manufacturing

Validation flow

Compilation

Simulation model

Validation Analysis Estimation

Compilation

Simulation model

Validation Analysis Estimation

Compilation

Simulation model

Validation Analysis Estimation

Compilation

Simulation model

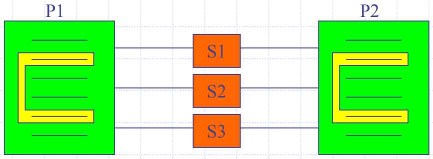
Validation Analysis Estimation

# Outline

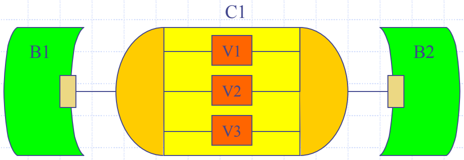
- SpecC Design methodology
  - Synthesis Flow
- SpecC Language
  - SpecC Model
  - Language Requirements

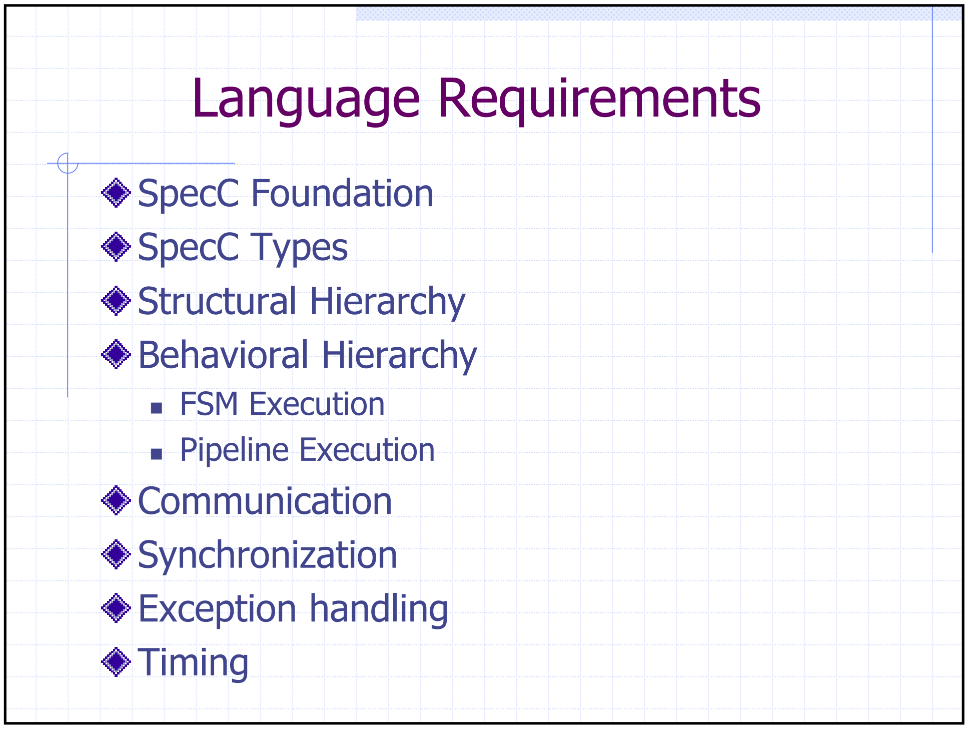
# The SpecC Model

- Traditional Model
  - VHDL, Verilog
  - Intermixed computation and communication
  - Not identified by tools
    - New communication
    - New computation



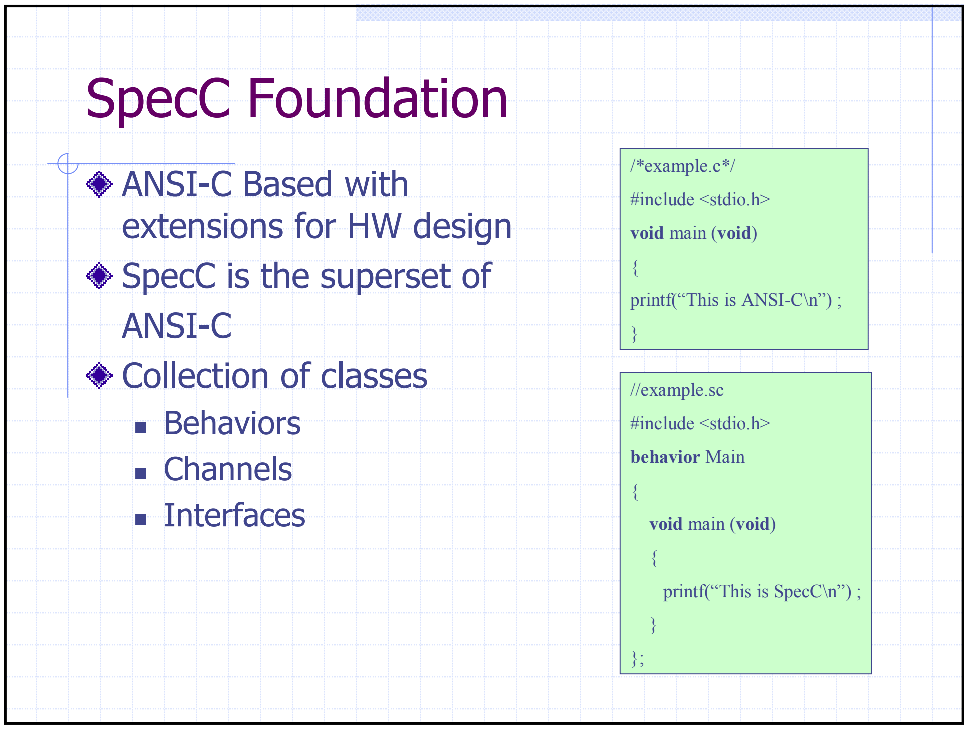
- SpecC Model
  - Separated computation and communication
  - Behaviors and Channels
  - Plug-and -Play
    - Computation
    - Communication





# Language Requirements

- SpecC Foundation
- SpecC Types
- Structural Hierarchy
- Behavioral Hierarchy
  - FSM Execution
  - Pipeline Execution
- Communication
- Synchronization
- Exception handling
- Timing

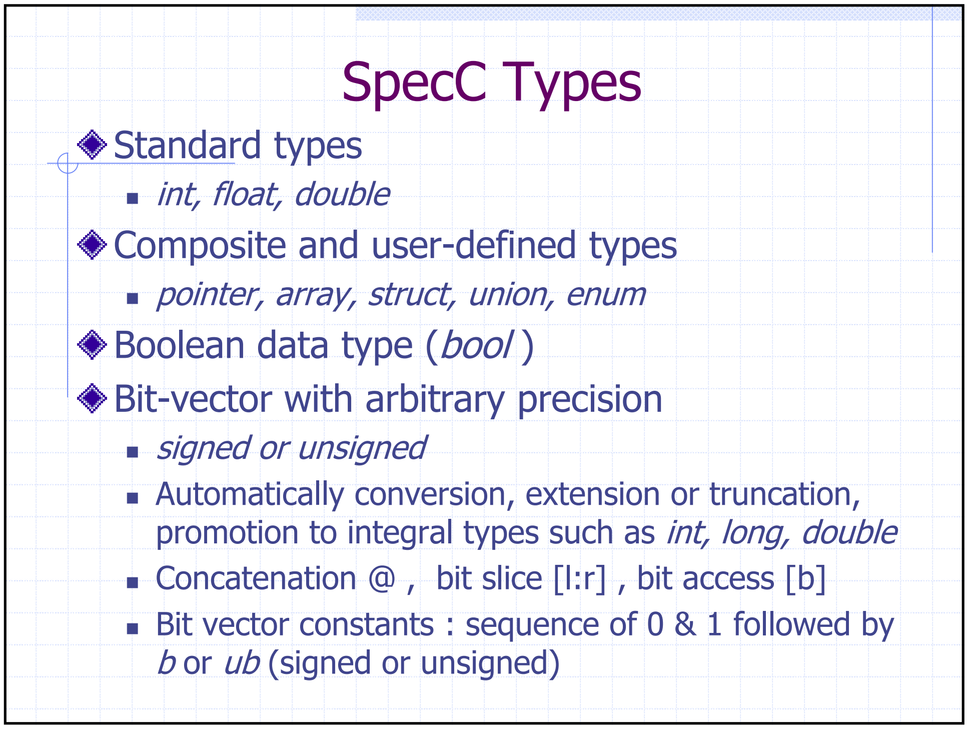


# SpecC Foundation

- ANSI-C Based with extensions for HW design
- SpecC is the superset of ANSI-C
- Collection of classes
  - Behaviors
  - Channels
  - Interfaces

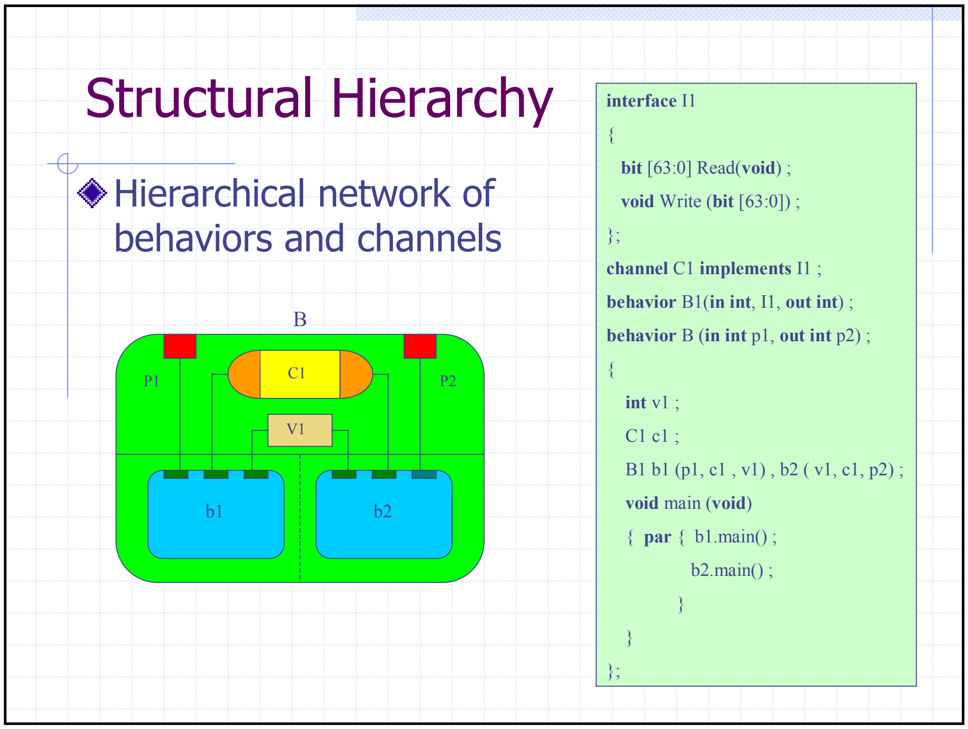
```
/*example.c*/
#include <stdio.h>
void main (void)
{
printf(“This is ANSI-C\n”) ;
}
```

```
//example.sc
#include <stdio.h>
behavior Main
{
   void main (void)
   {
     printf(“This is SpecC\n”) ;
   }
};
```



# SpecC Types

- Standard types
  - *int, float, double*
- Composite and user-defined types
  - *pointer, array, struct, union, enum*
- Boolean data type (*bool* )
- Bit-vector with arbitrary precision
  - *signed or unsigned*
  - Automatically conversion, extension or truncation, promotion to integral types such as *int, long, double*
  - Concatenation @ ,  bit slice [l:r] , bit access [b]
  - Bit vector constants : sequence of 0 & 1 followed by *b* or *ub* (signed or unsigned)

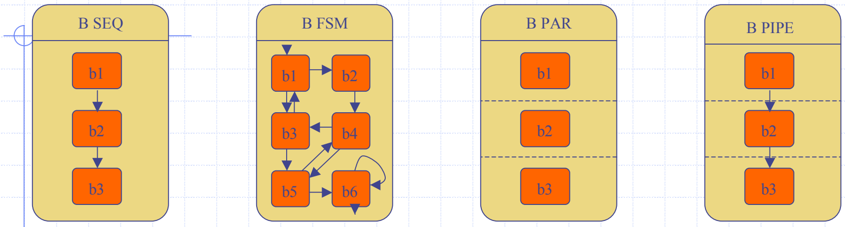


# Structural Hierarchy

- Hierarchical network of behaviors and channels

```
interface I1
{
  bit [63:0] Read(void) ;
  void Write (bit [63:0]) ;
};
channel C1 implements I1 ;
behavior B1(in int, I1, out int) ;
behavior B (in int p1, out int p2) ;
{
   int v1 ;
   C1 c1 ;
   B1 b1 (p1, c1 , v1) , b2 ( v1, c1, p2) ;
   void main (void)
   {  par  {  b1.main() ;
              b2.main() ;
           }
   }
};
```

# Behavioral Hierarchy



```
Behavior B_seq
{
   B b1, b2, b3 ;
   void main (void)
   {  b1.main( ) ;
      b2.main( ) ;
      b3.main( ) ;
   }
};
```

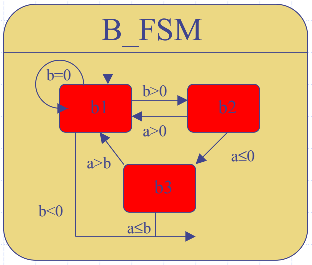
```
Behavior B_fsm
{
   B b1,b2,b3,b4,b5,b6;
   void main (void)
   {  fsm ( b1: (…)
             b2: (…)
             …)
   }
};
```

```
Behavior B_par
{
   B b1 ,b2 ,b3 ;
   void main (void)
   {  par { b1.main( ) ;
            b2.main( ) ;
            b3.main( ) ;
   }     }
};
```

```
Behavior B_pipe
{
   B b1, b2, b3 ;
   void main (void)
   {  pipe { b1.main( ) ;
             b2.main( ) ;
             b3.main( ) ;
   }      }
};
```

# Finite State Machine Execution

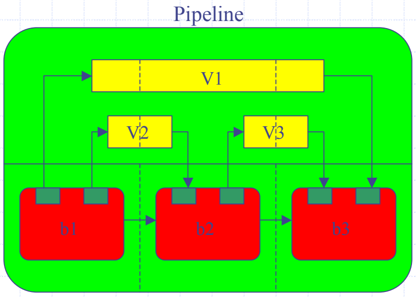
- Both Mealy and Moore FSMs
- State transition definition :
  - current state, condition, next state



```
behavior B_FSM (in int a, in int b)
{
   B b1, b2, b3 ;
   void main (void)
   {  fsm  { b1: { if (b<0) break ;
                   if (b==0) goto b1 ;
                   if (b>0) goto b2 ;  }
             b2: { if (a>0) goto b1 ;  }
             b3: { if (a>b) goto b1 ;  }
           }
   }
};
```

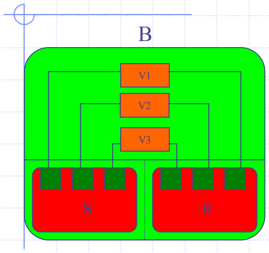
# Pipeline Execution

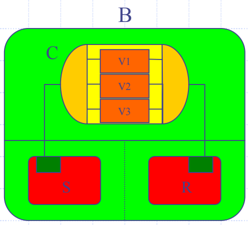
- Automatic communication buffering
    - FIFO instead of standard variables
    - Special care with *pipe* storage class

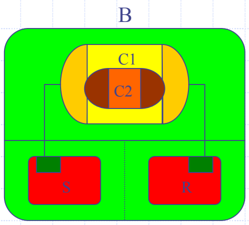


```
behavior Pipeline
{
  piped piped int v1 ;
  piped int v2 ;
  piped int v3 ;
  Stage1 b1 (v1, v2) ;
  Stage2 b2 (v2, v3) ;
  Stage3 b3 (v3, v1) ;
  void main (void)
  {
     int i ;
     pipe (i=0 ; i<10; i++)
        {  b1.main( ) ;
           b2.main( ) ;
           b3.main( ) ;
        }
  }
} ;
```

# Communication







- Variables :
    - Shared memory communication
    - comm. wires

- Virtual channel :
    - Message passing communication
    - leaf channels

- Hierarchical channels :
    - Communication protocol stack

# Synchronization

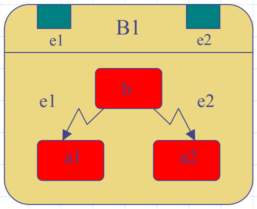
```
behavior S (out event Req,
            out float Data,
            in event Ack)
{
  float X;
  void main (void)
  { …
    Data = X ;
    notify Req ;
    wait Ack ;
    …
  }
};
```

- *event*
  - Basic type of synchronization
- *wait*
- *notify*
- *notifyone*

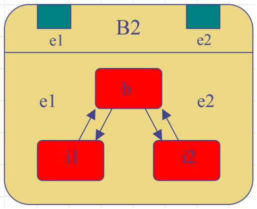
```
behavior R (in event Req,
            in float Data,
            out event Ack)
{
  float Y;
  void main (void)
  { …
    wait Req ;
    Y = Data ;
    notify Ack ;
    …
  }
};
```

# Exception Handling

Abortion

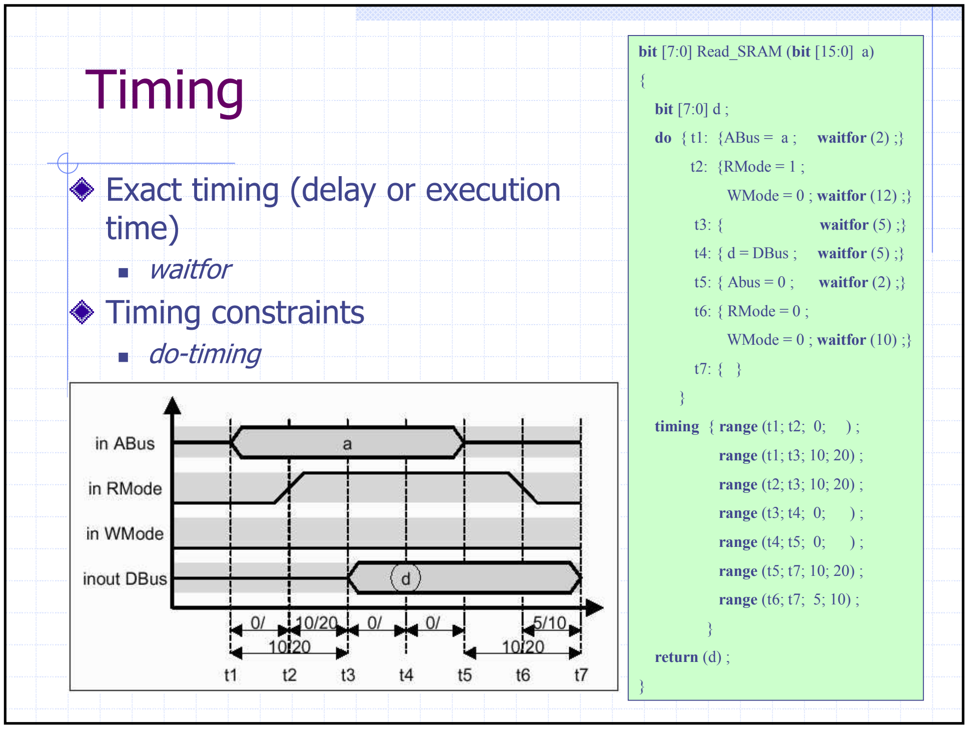


Interrupt



```
behavior B1 (in event e1, in event e2)
{
  B b, a1, a2 ;
  void main (void)
  {
    try  { b.main( ) ; }
    trap (e1)  { a1.main( ) ; }
    trap (e2)  { a2.main( ) ; }
  }
} ;
```

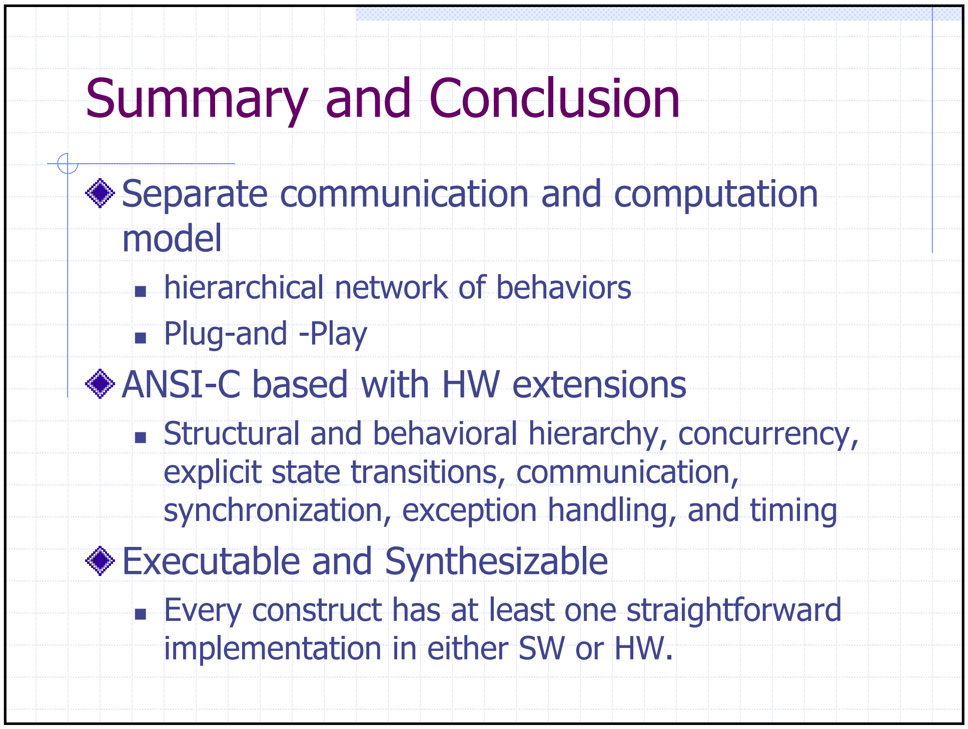
```
behavior B2 (in event e1, in event e2)
{
  B b, a1, a2 ;
  void main (void)
  {
    try  { b.main( ) ; }
    trap (e1)  { i1.main( ) ; }
    trap (e2)  { i2.main( ) ; }
  }
} ;
```



# Timing

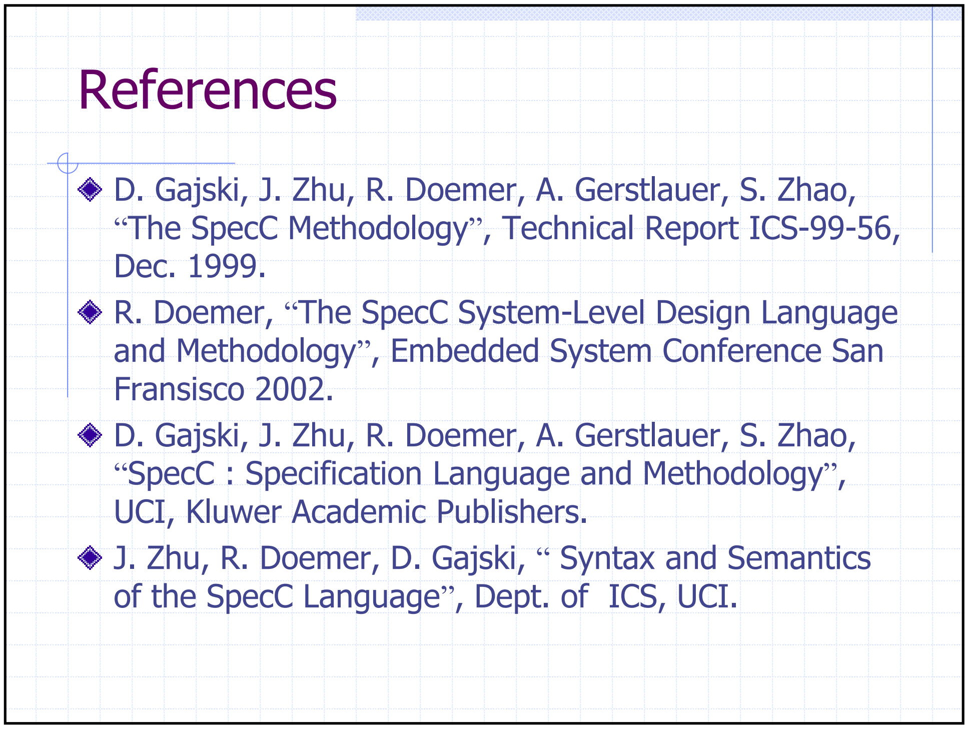
- Exact timing (delay or execution time)
  - *waitfor*
- Timing constraints
  - *do-timing*

```
bit [7:0] Read_SRAM (bit [15:0]  a)
{
   bit [7:0] d ;
   do  { t1:  {ABus =  a ;     waitfor (2) ;}
          t2:  {RMode = 1 ;
                   WMode = 0 ; waitfor (12) ;}
          t3: {                     waitfor (5) ;}
          t4: { d = DBus ;      waitfor (5) ;}
          t5: { Abus = 0 ;      waitfor (2) ;}
          t6: { RMode = 0 ;
                   WMode = 0 ; waitfor (10) ;}
          t7: {   }
        }
   timing  { range (t1; t2;  0;     ) ;
                range (t1; t3; 10; 20) ;
                range (t2; t3; 10; 20) ;
                range (t3; t4;  0;     ) ;
                range (t4; t5;  0;     ) ;
                range (t5; t7; 10; 20) ;
                range (t6; t7;  5; 10) ;
              }
   return (d) ;
}
```



# Summary and Conclusion

- Separate communication and computation model
  - hierarchical network of behaviors
  - Plug-and -Play
- ANSI-C based with HW extensions
  - Structural and behavioral hierarchy, concurrency, explicit state transitions, communication, synchronization, exception handling, and timing
- Executable and Synthesizable
  - Every construct has at least one straightforward implementation in either SW or HW.



# References

- D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, S. Zhao, “The SpecC Methodology”, Technical Report ICS-99-56, Dec. 1999.
- R. Doemer, “The SpecC System-Level Design Language and Methodology”, Embedded System Conference San Fransisco 2002.
- D. Gajski, J. Zhu, R. Doemer, A. Gerstlauer, S. Zhao, “SpecC : Specification Language and Methodology”, UCI, Kluwer Academic Publishers.
- J. Zhu, R. Doemer, D. Gajski, “ Syntax and Semantics of the SpecC Language”, Dept. of  ICS, UCI.