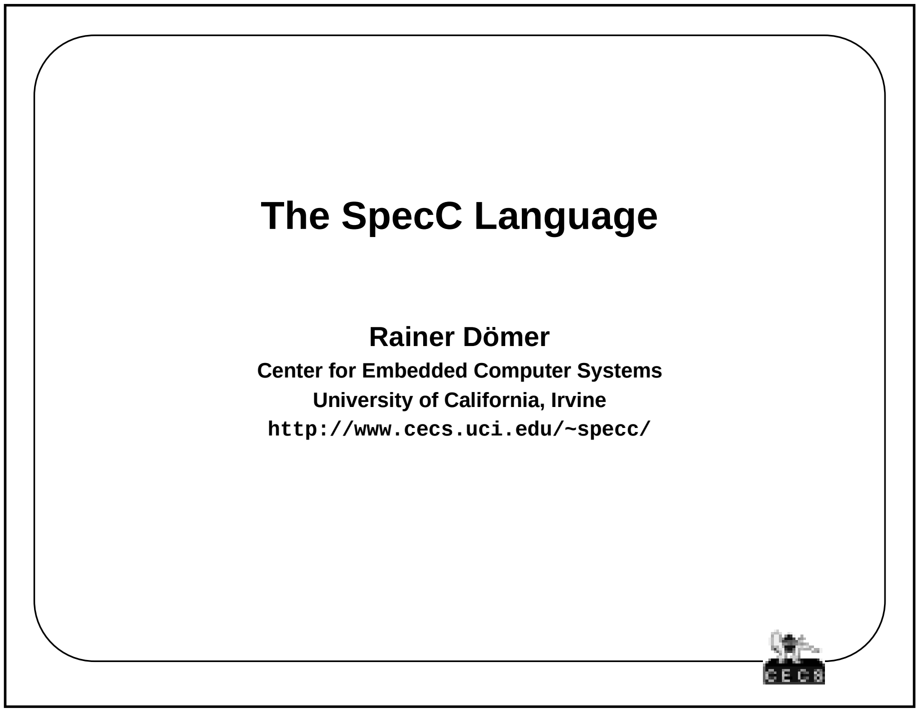
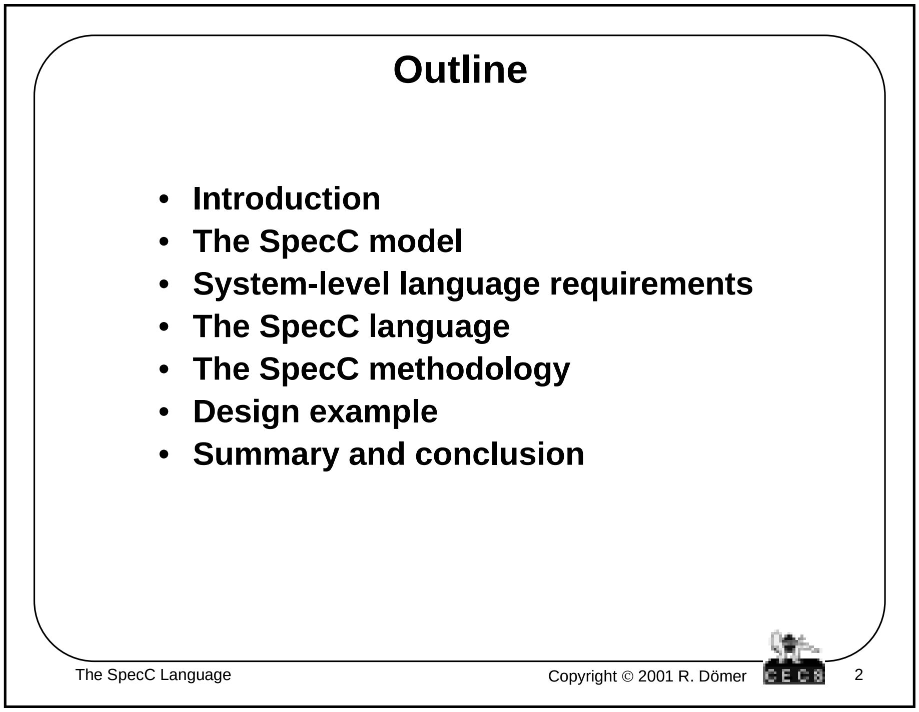


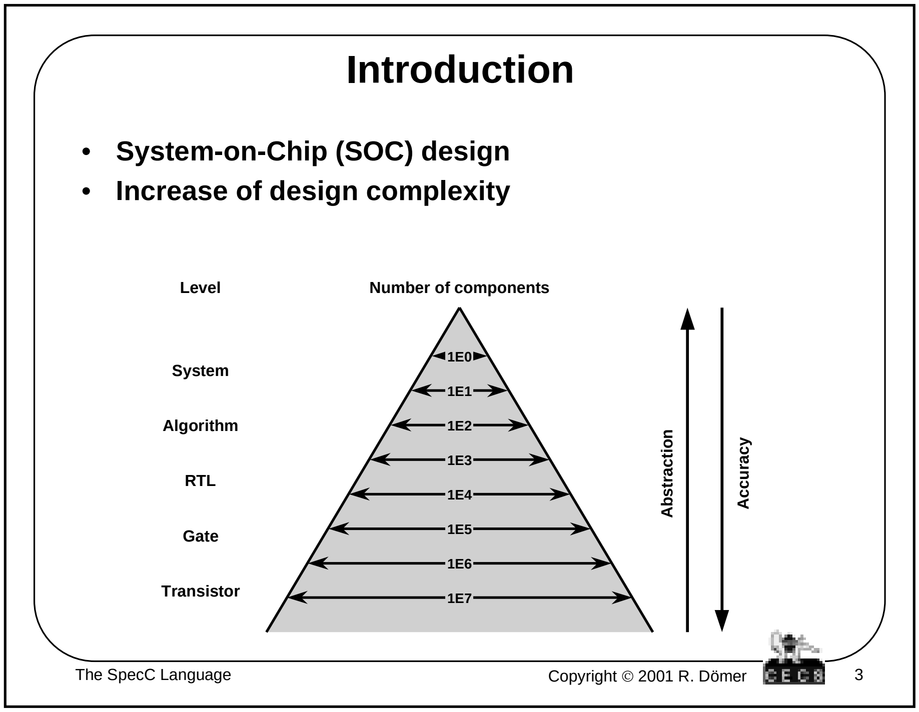
The SpecC Language

Rainer Dömer
Center for Embedded Computer Systems
University of California, Irvine
`http://www.cecs.uci.edu/~specc/`



Outline

- Introduction
- The SpecC model
- System-level language requirements
- The SpecC language
- The SpecC methodology
- Design example
- Summary and conclusion

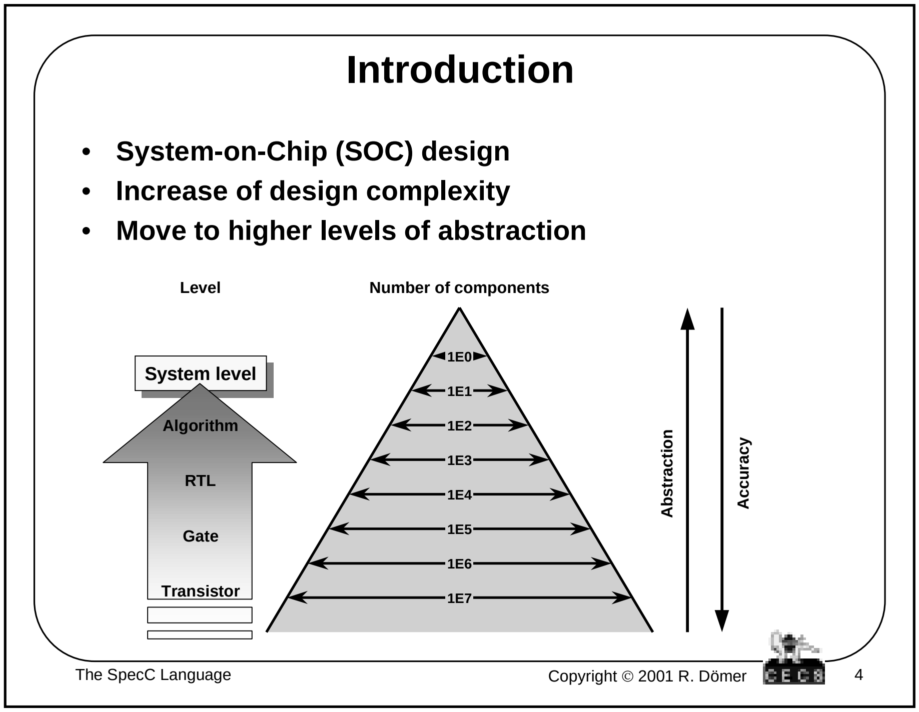


Introduction

- System-on-Chip (SOC) design
- Increase of design complexity

The SpecC Language

3

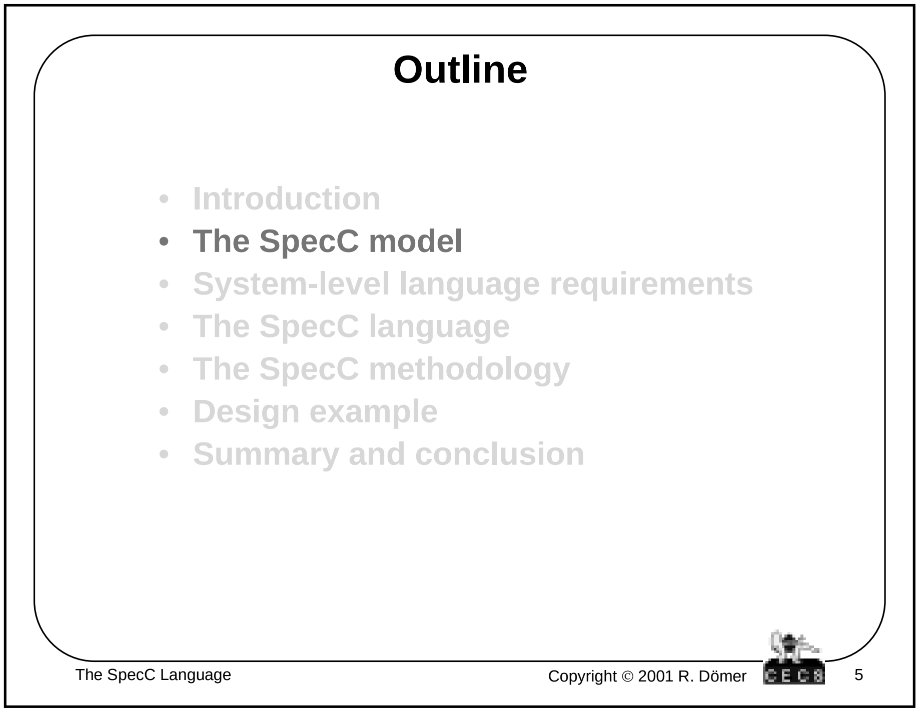


Introduction

- System-on-Chip (SOC) design
- Increase of design complexity
- Move to higher levels of abstraction

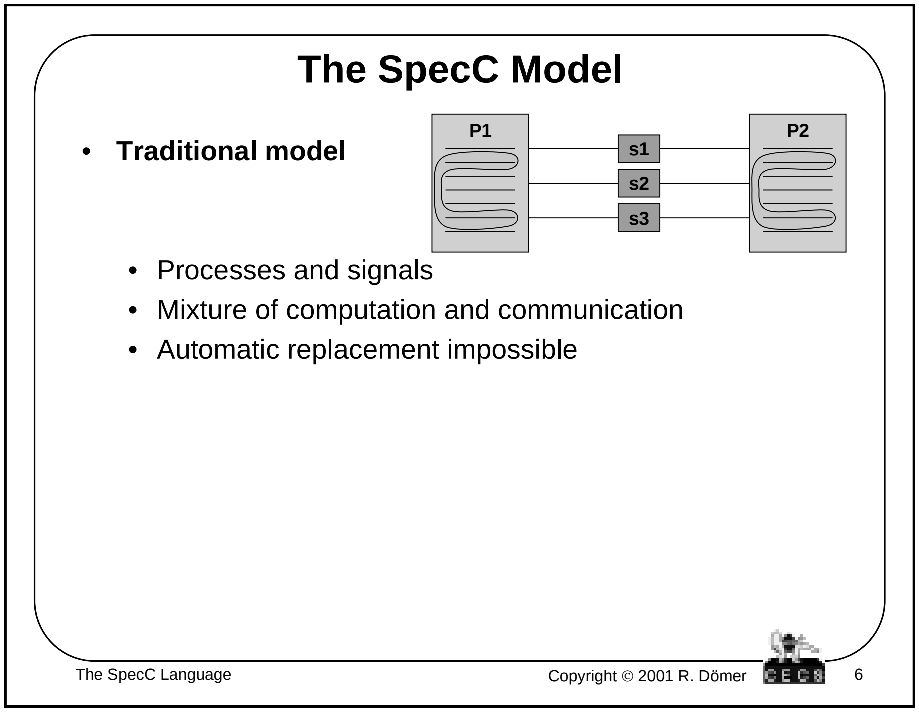
The SpecC Language

4



Outline

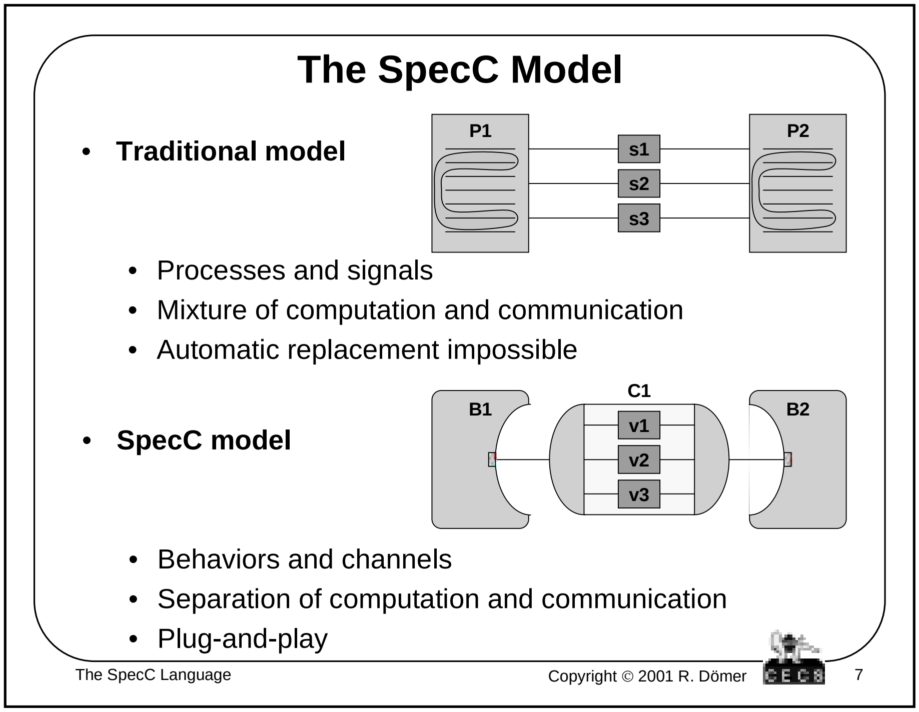
- Introduction
- The SpecC model
- System-level language requirements
- The SpecC language
- The SpecC methodology
- Design example
- Summary and conclusion



The SpecC Model

- **Traditional model**

  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible



The SpecC Model

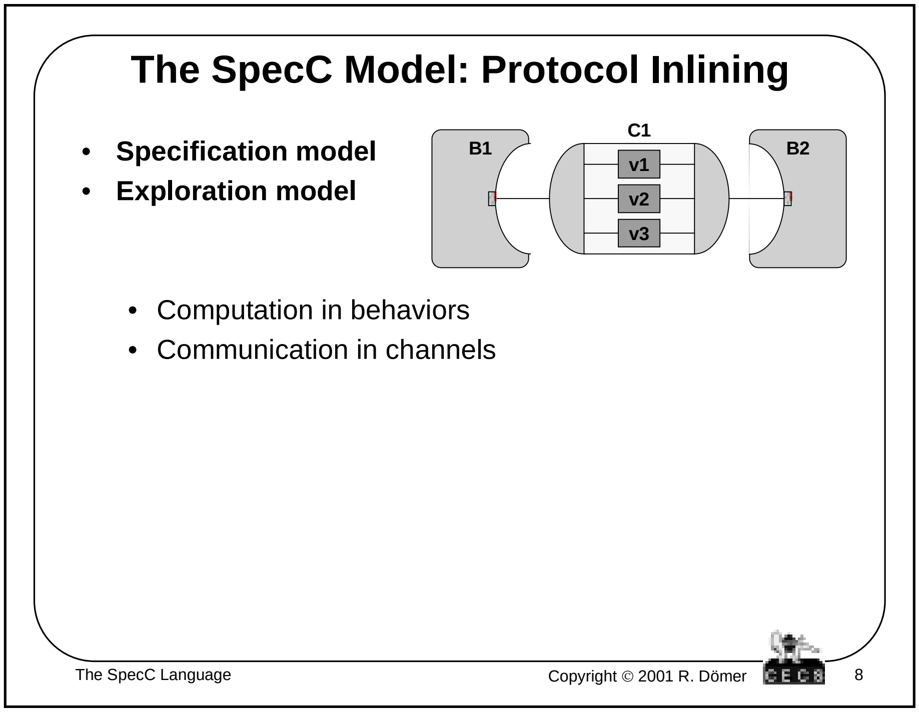
- **Traditional model**

  - Processes and signals
  - Mixture of computation and communication
  - Automatic replacement impossible

- **SpecC model**

  - Behaviors and channels
  - Separation of computation and communication
  - Plug-and-play

The SpecC Language Copyright © 2001 R. Dömer 7

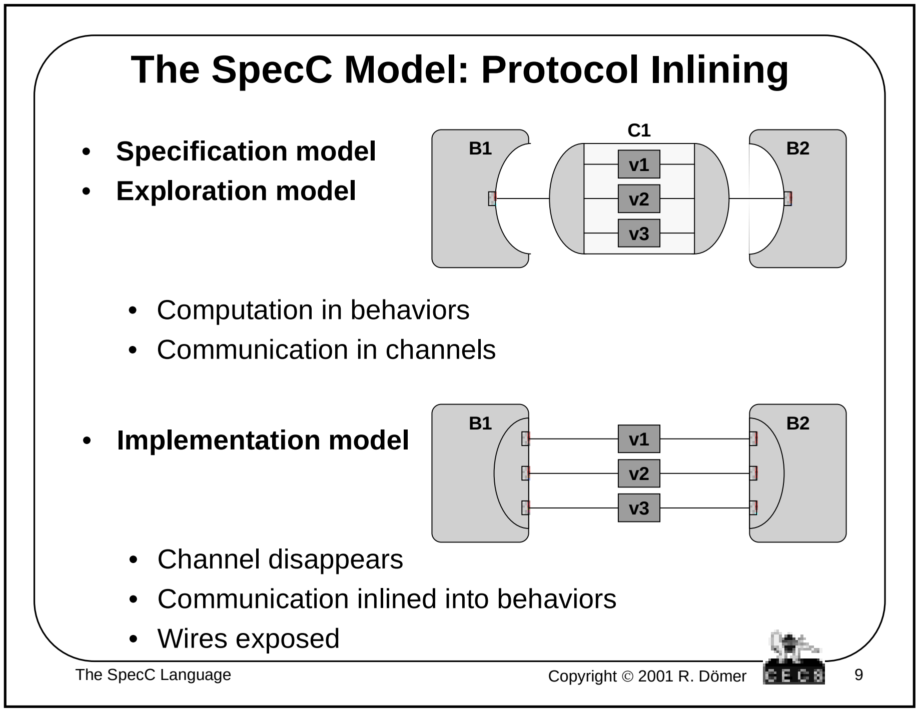


The SpecC Model: Protocol Inlining

- **Specification model**
- **Exploration model**

  - Computation in behaviors
  - Communication in channels

The SpecC Language Copyright © 2001 R. Dömer 8



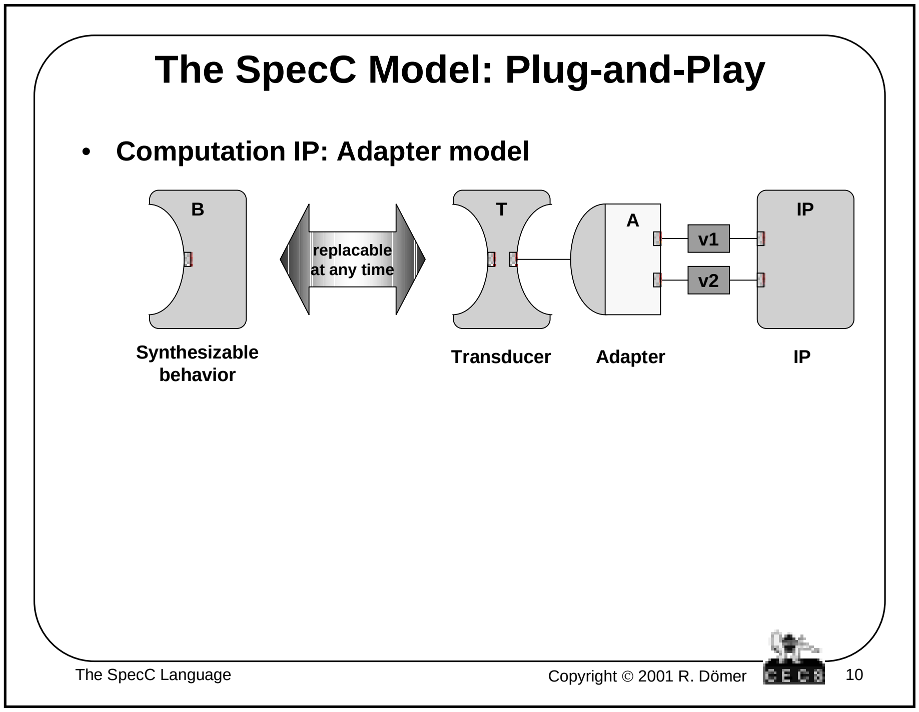
The SpecC Model: Protocol Inlining

- **Specification model**
- **Exploration model**

  - Computation in behaviors
  - Communication in channels

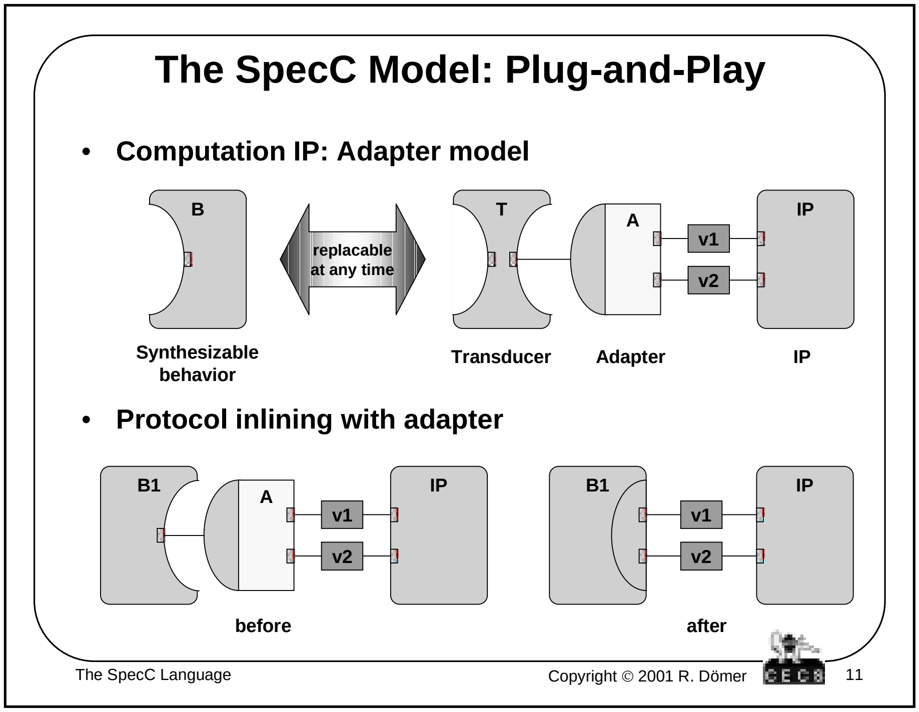
- **Implementation model**

  - Channel disappears
  - Communication inlined into behaviors
  - Wires exposed



The SpecC Model: Plug-and-Play

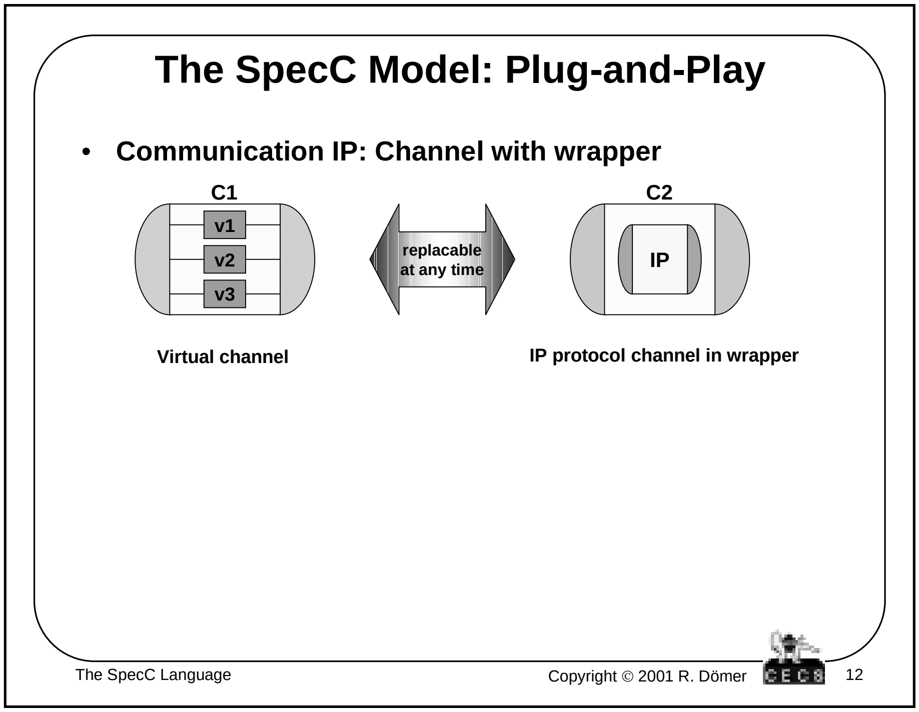
- **Computation IP: Adapter model**



The SpecC Model: Plug-and-Play

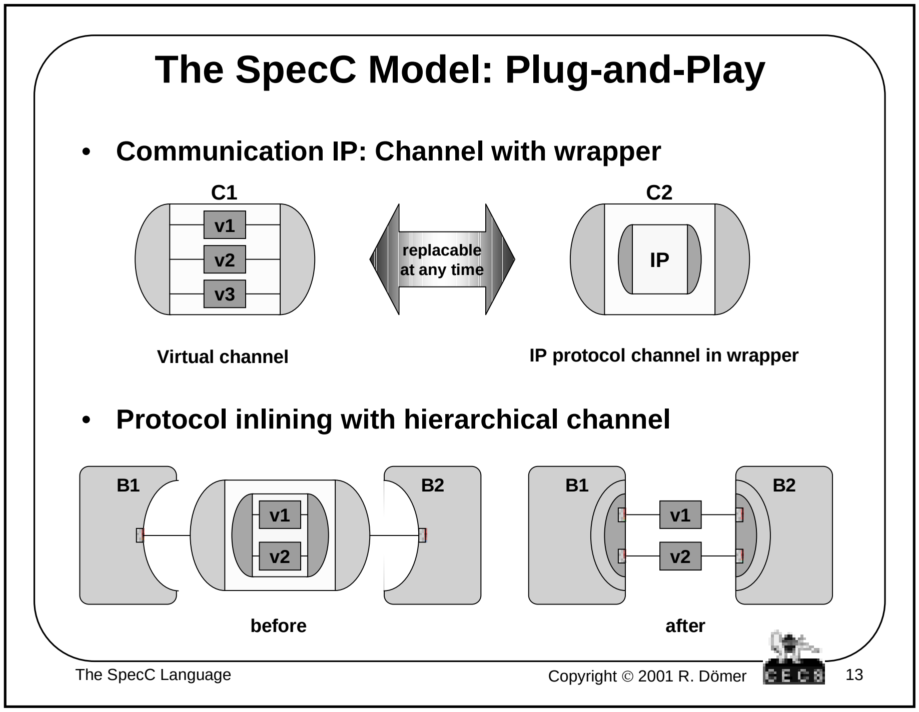
- Computation IP: Adapter model

- Protocol inlining with adapter



The SpecC Model: Plug-and-Play

- Communication IP: Channel with wrapper



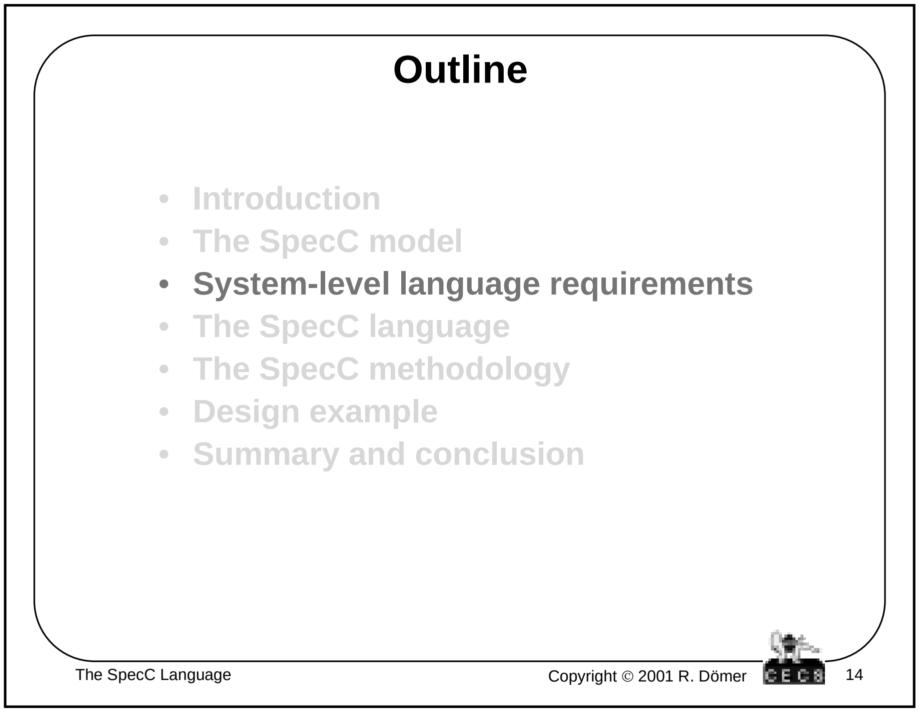
The SpecC Model: Plug-and-Play

- Communication IP: Channel with wrapper

- Protocol inlining with hierarchical channel

The SpecC Language

13

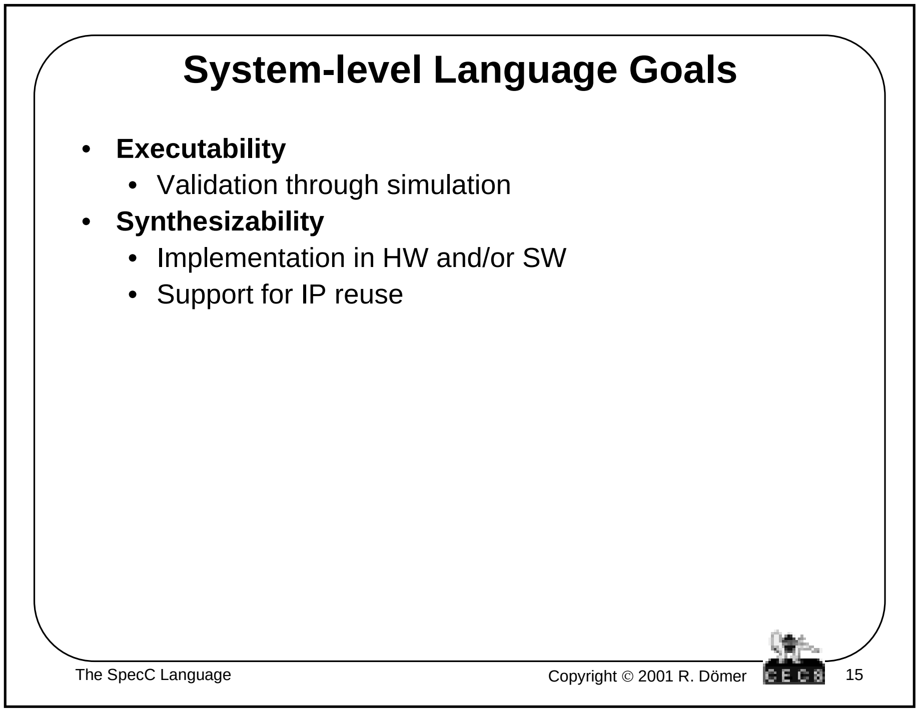


Outline

- Introduction
- The SpecC model
- System-level language requirements
- The SpecC language
- The SpecC methodology
- Design example
- Summary and conclusion

The SpecC Language

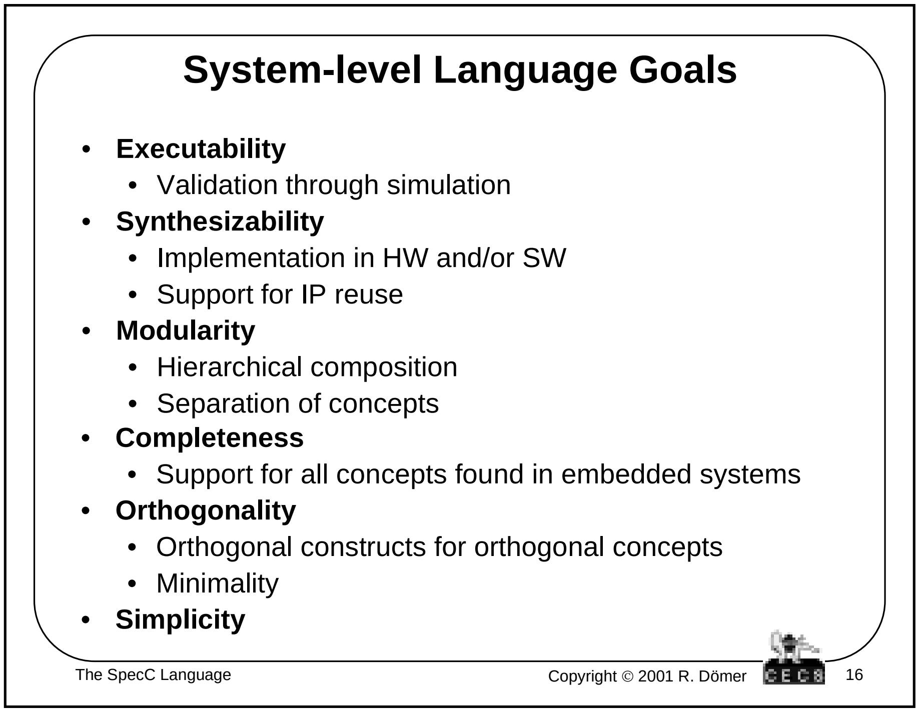
14



System-level Language Goals

- **Executability**
  - Validation through simulation
- **Synthesizability**
  - Implementation in HW and/or SW
  - Support for IP reuse

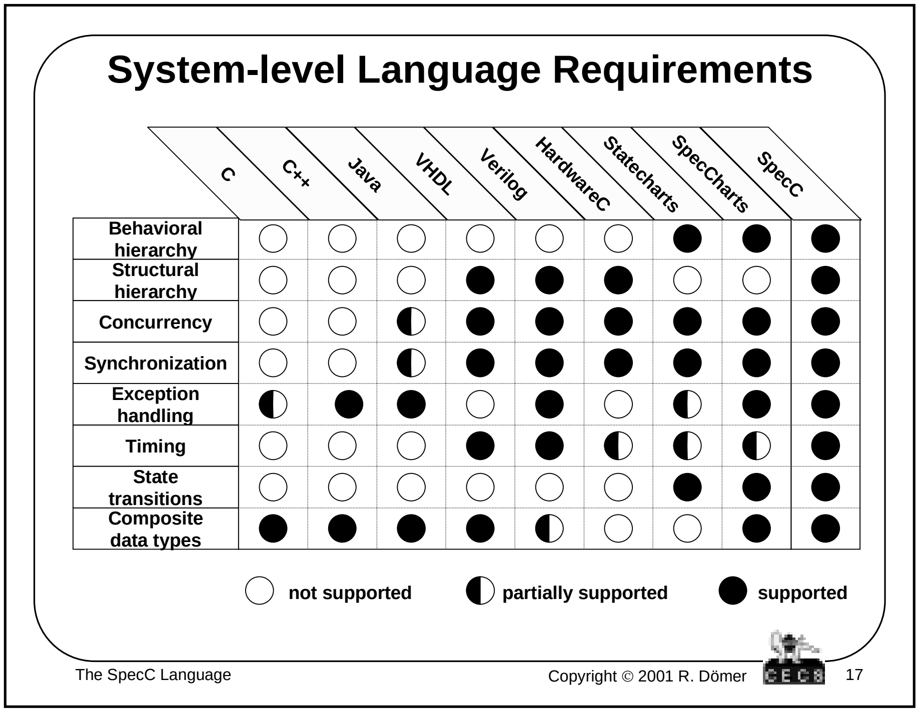
The SpecC Language Copyright © 2001 R. Dömer 15



System-level Language Goals

- **Executability**
  - Validation through simulation
- **Synthesizability**
  - Implementation in HW and/or SW
  - Support for IP reuse
- **Modularity**
  - Hierarchical composition
  - Separation of concepts
- **Completeness**
  - Support for all concepts found in embedded systems
- **Orthogonality**
  - Orthogonal constructs for orthogonal concepts
  - Minimality
- **Simplicity**

The SpecC Language Copyright © 2001 R. Dömer 16

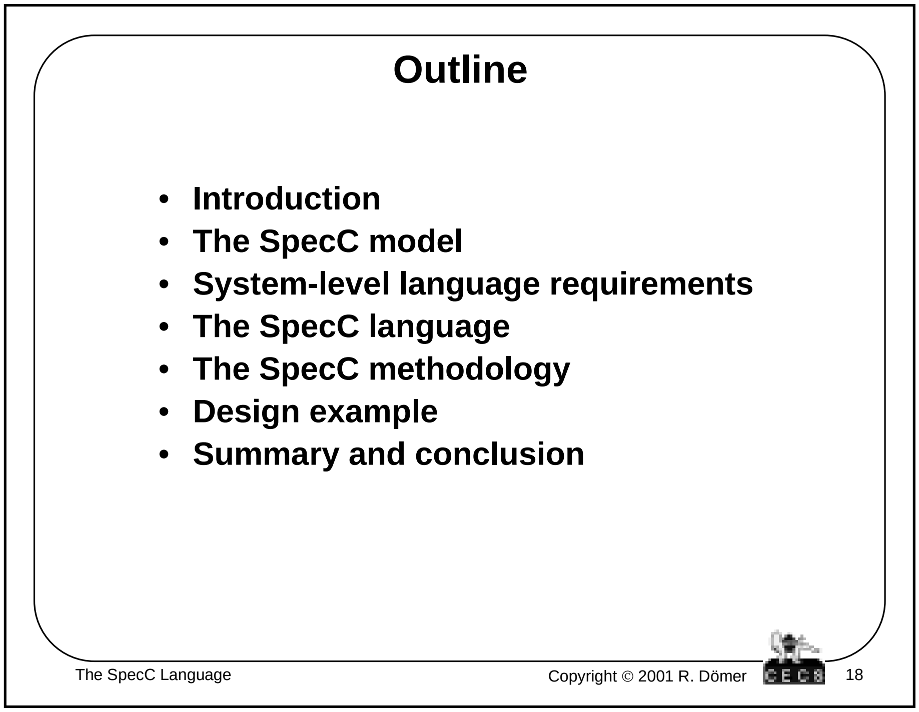


System-level Language Requirements

	C	C++	Java	VHDL	Verilog	HardwareC	Statecharts	SpecCharts	SpecC
Behavioral hierarchy	○	○	○	○	○	○	●	●	●
Structural hierarchy	○	○	○	●	●	●	○	○	●
Concurrency	○	○	◐	●	●	●	●	●	●
Synchronization	○	○	◐	●	●	●	●	●	●
Exception handling	◐	●	●	○	●	○	◐	●	●
Timing	○	○	○	●	●	◐	◐	◐	●
State transitions	○	○	○	○	○	○	●	●	●
Composite data types	●	●	●	●	◐	○	○	●	●

○ not supported ◐ partially supported ● supported

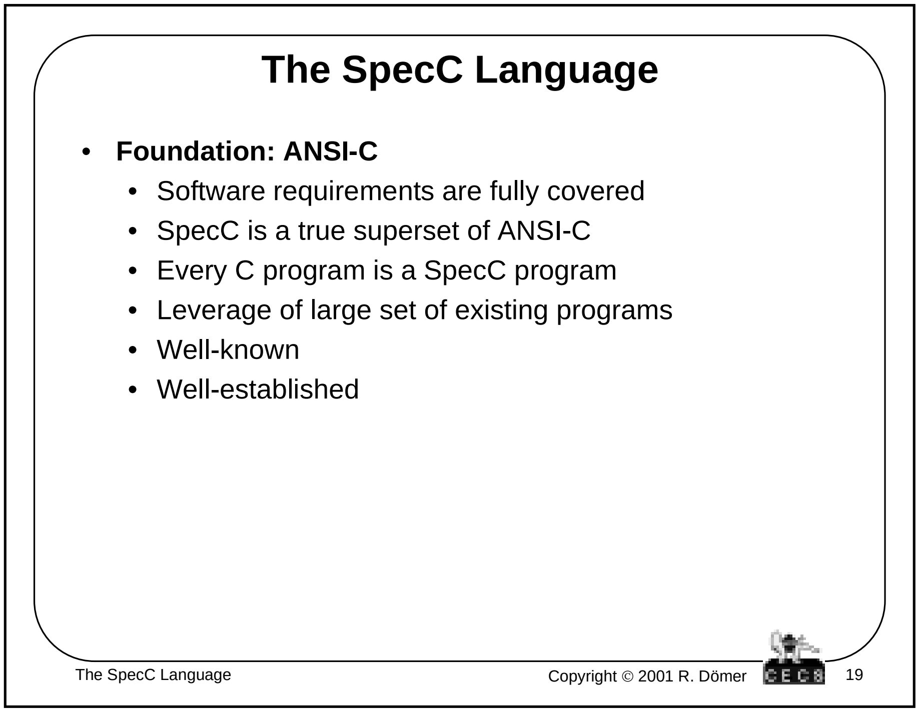
The SpecC Language Copyright © 2001 R. Dömer 17



Outline

- Introduction
- The SpecC model
- System-level language requirements
- The SpecC language
- The SpecC methodology
- Design example
- Summary and conclusion

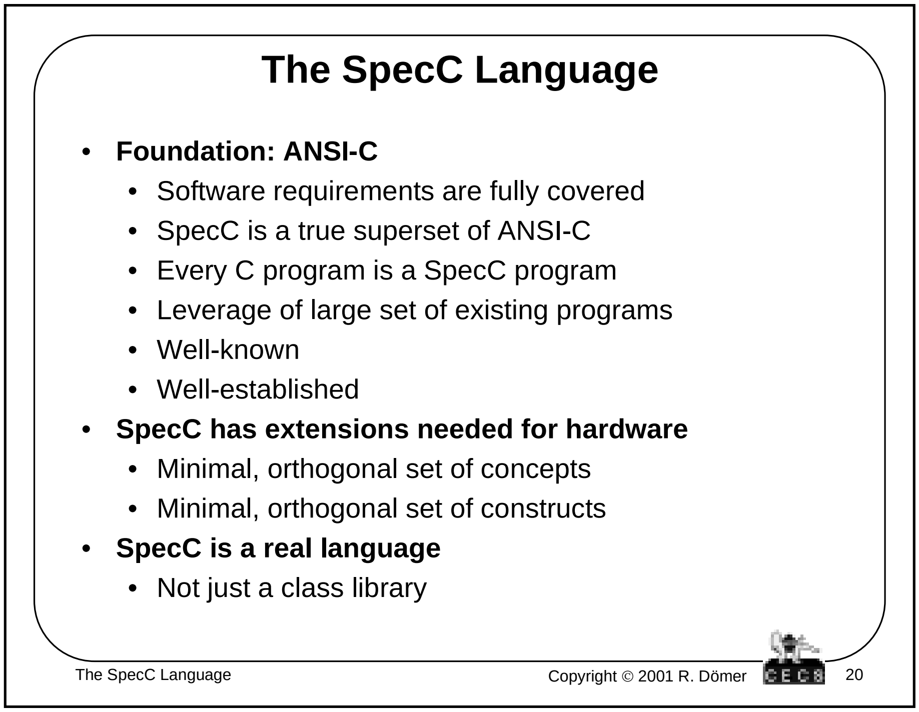
The SpecC Language Copyright © 2001 R. Dömer 18



The SpecC Language

- **Foundation: ANSI-C**
  - Software requirements are fully covered
  - SpecC is a true superset of ANSI-C
  - Every C program is a SpecC program
  - Leverage of large set of existing programs
  - Well-known
  - Well-established

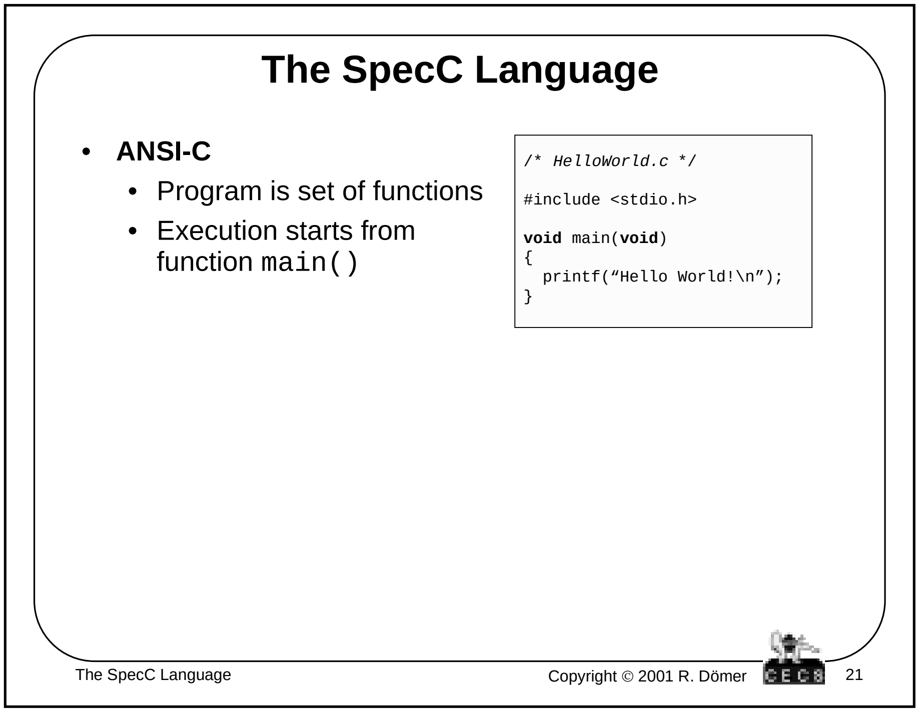
The SpecC Language Copyright © 2001 R. Dömer 19



The SpecC Language

- **Foundation: ANSI-C**
  - Software requirements are fully covered
  - SpecC is a true superset of ANSI-C
  - Every C program is a SpecC program
  - Leverage of large set of existing programs
  - Well-known
  - Well-established
- **SpecC has extensions needed for hardware**
  - Minimal, orthogonal set of concepts
  - Minimal, orthogonal set of constructs
- **SpecC is a real language**
  - Not just a class library

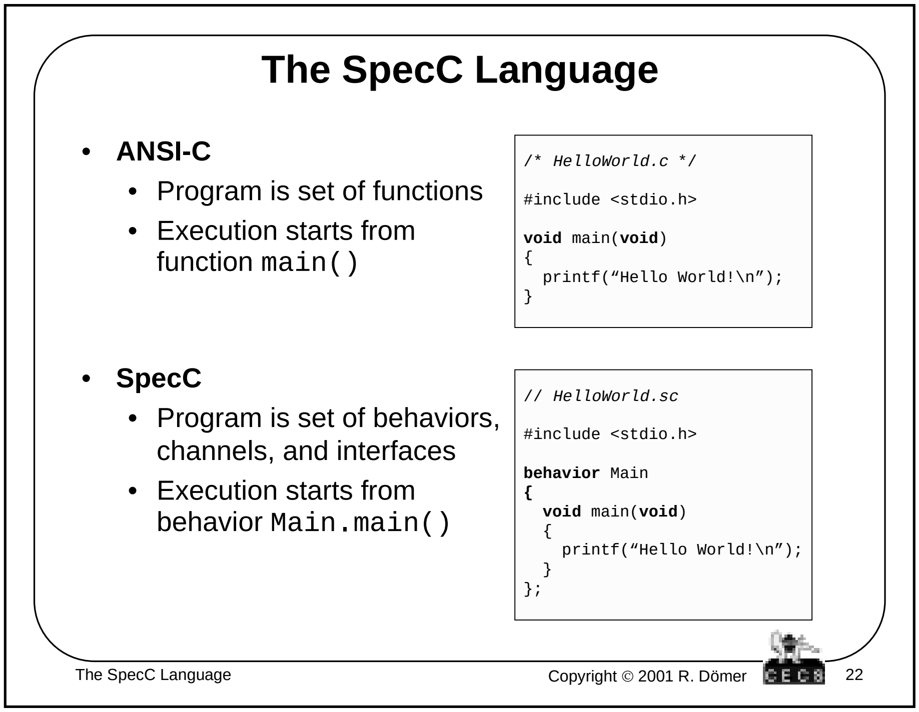
The SpecC Language Copyright © 2001 R. Dömer 20



The SpecC Language

- **ANSI-C**
  - Program is set of functions
  - Execution starts from function `main()`

```
/* HelloWorld.c */

#include <stdio.h>

void main(void)
{
  printf(“Hello World!\n”);
}
```



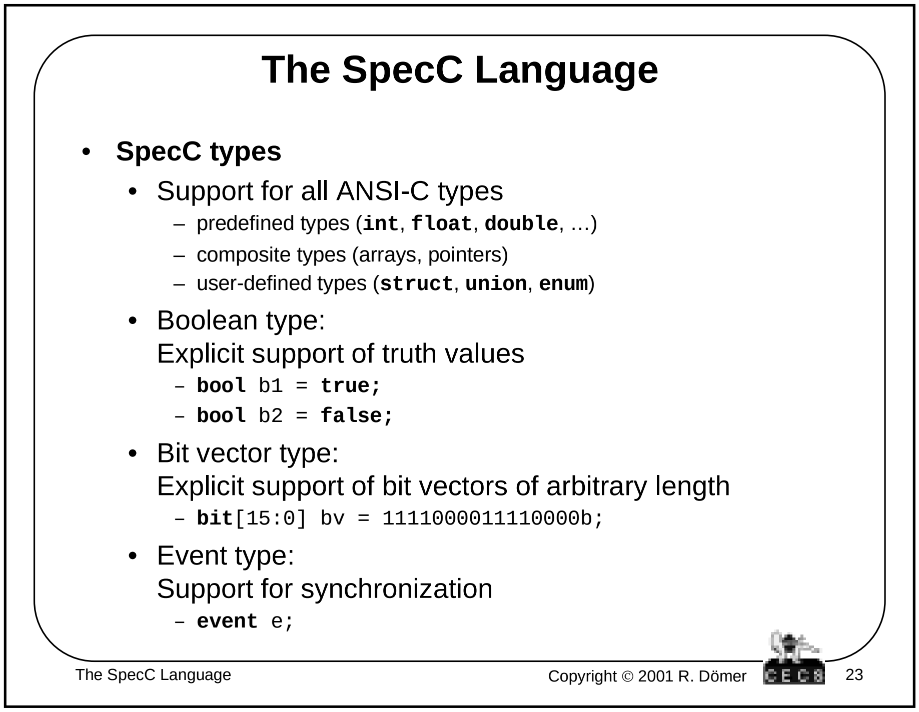
The SpecC Language

- **ANSI-C**
  - Program is set of functions
  - Execution starts from function `main()`

```
/* HelloWorld.c */

#include <stdio.h>

void main(void)
{
  printf(“Hello World!\n”);
}
```

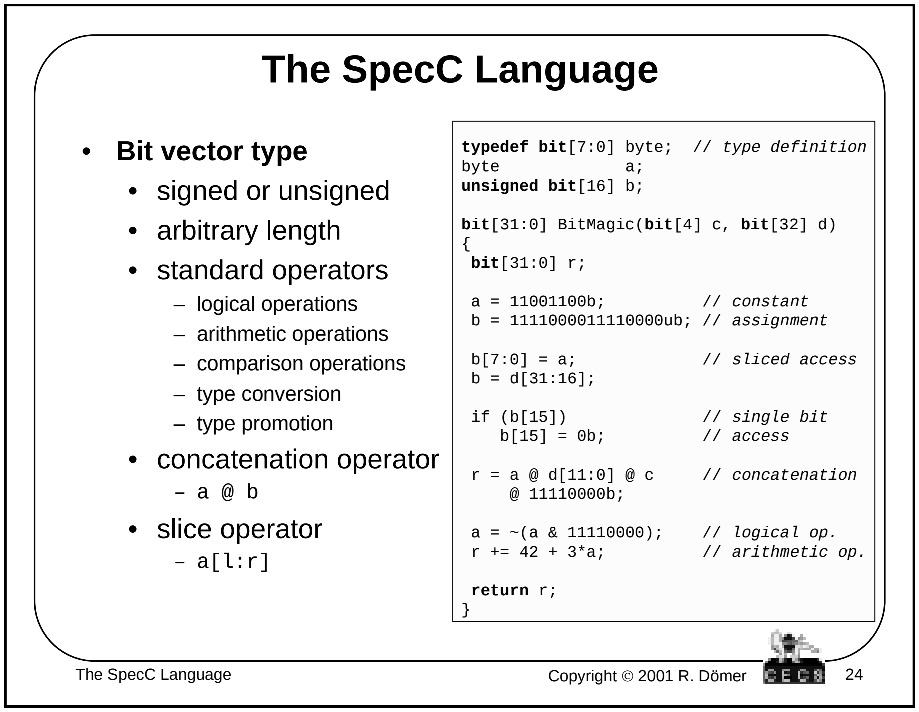
- **SpecC**
  - Program is set of behaviors, channels, and interfaces
  - Execution starts from behavior `Main.main()`

```
// HelloWorld.sc

#include <stdio.h>

behavior Main
{
  void main(void)
  {
    printf(“Hello World!\n”);
  }
};
```



The SpecC Language

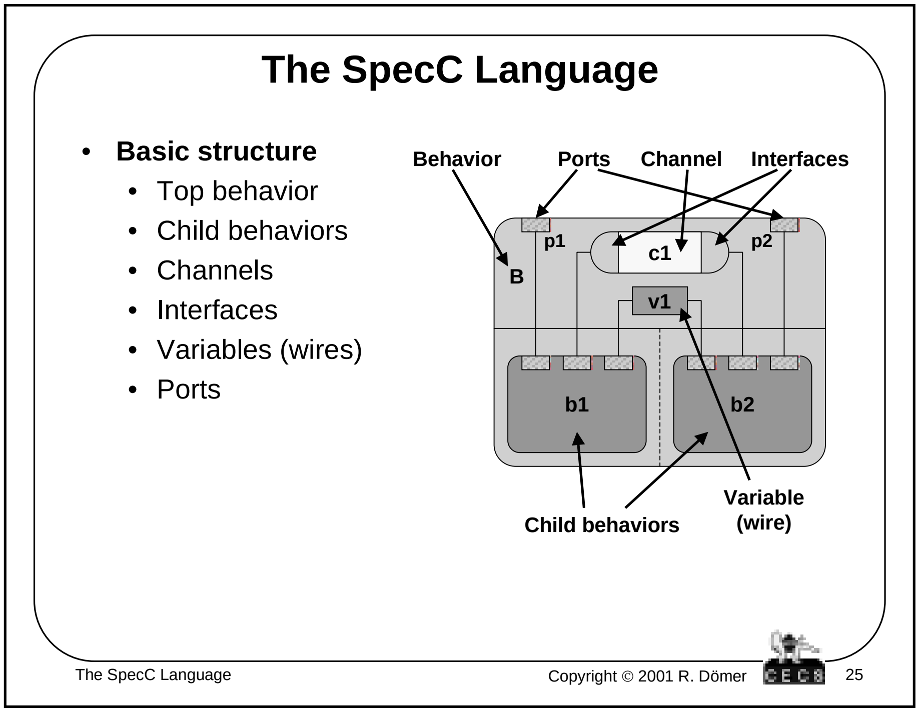
- **SpecC types**
  - Support for all ANSI-C types
    - predefined types (**`int`**, **`float`**, **`double`**, …)
    - composite types (arrays, pointers)
    - user-defined types (**`struct`**, **`union`**, **`enum`**)
  - Boolean type:
    Explicit support of truth values
    - **`bool`** `b1 =` **`true;`**
    - **`bool`** `b2 =` **`false;`**
  - Bit vector type:
    Explicit support of bit vectors of arbitrary length
    - **`bit`**`[15:0] bv = 1111000011110000b;`
  - Event type:
    Support for synchronization
    - **`event`** `e;`



The SpecC Language

- **Bit vector type**
  - signed or unsigned
  - arbitrary length
  - standard operators
    - logical operations
    - arithmetic operations
    - comparison operations
    - type conversion
    - type promotion
  - concatenation operator
    - `a @ b`
  - slice operator
    - `a[l:r]`

```
typedef bit[7:0] byte;  // type definition
byte             a;
unsigned bit[16] b;

bit[31:0] BitMagic(bit[4] c, bit[32] d)
{
 bit[31:0] r;

 a = 11001100b;          // constant
 b = 1111000011110000ub; // assignment

 b[7:0] = a;             // sliced access
 b = d[31:16];

 if (b[15])              // single bit
    b[15] = 0b;          // access

 r = a @ d[11:0] @ c     // concatenation
     @ 11110000b;

 a = ~(a & 11110000);    // logical op.
 r += 42 + 3*a;          // arithmetic op.

 return r;
}
```



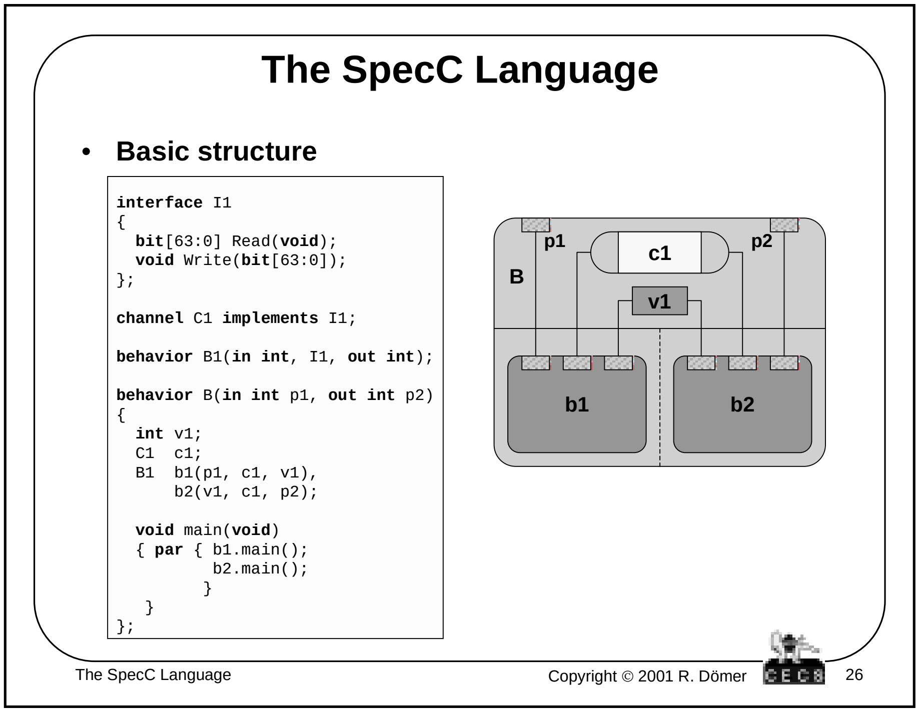
The SpecC Language

- **Basic structure**
  - Top behavior
  - Child behaviors
  - Channels
  - Interfaces
  - Variables (wires)
  - Ports

The SpecC Language

Copyright © 2001 R. Dömer

25



The SpecC Language

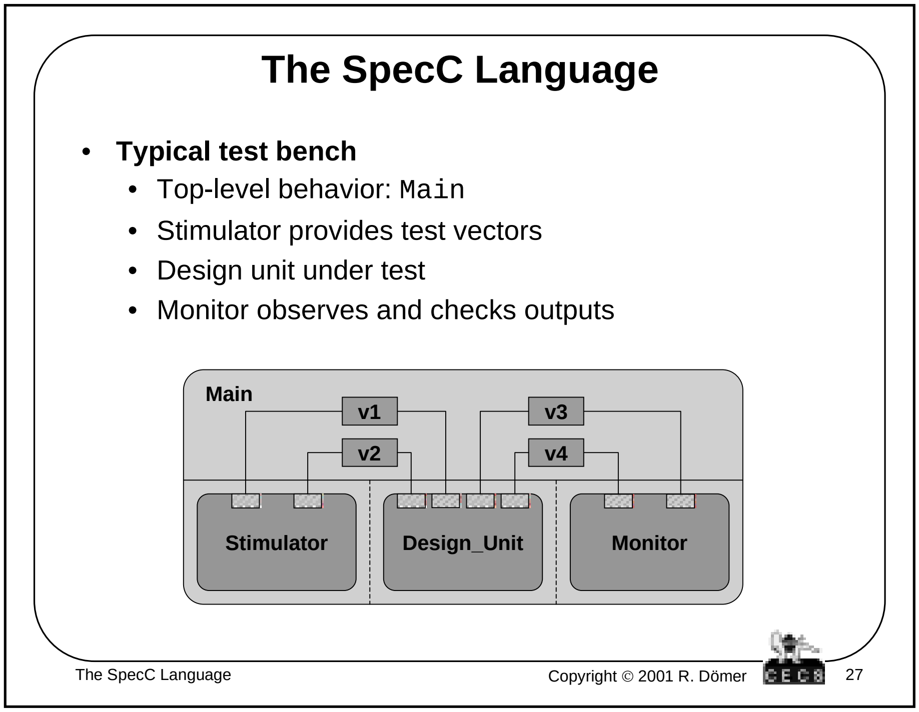
- **Basic structure**

```
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();
        }
   }
};
```

The SpecC Language

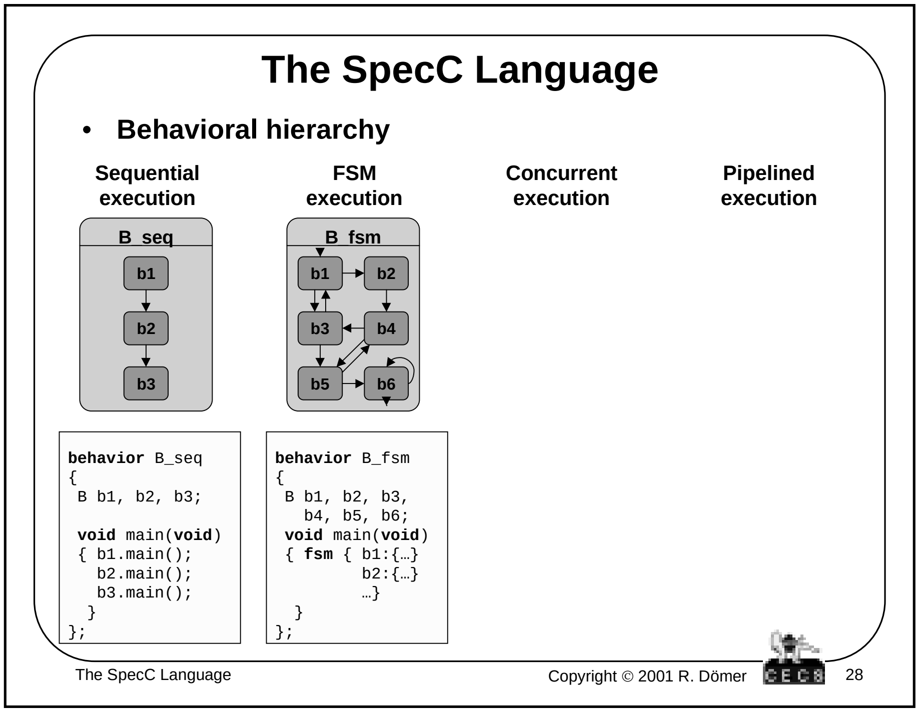
Copyright © 2001 R. Dömer

26



The SpecC Language

- **Typical test bench**
  - Top-level behavior: `Main`
  - Stimulator provides test vectors
  - Design unit under test
  - Monitor observes and checks outputs



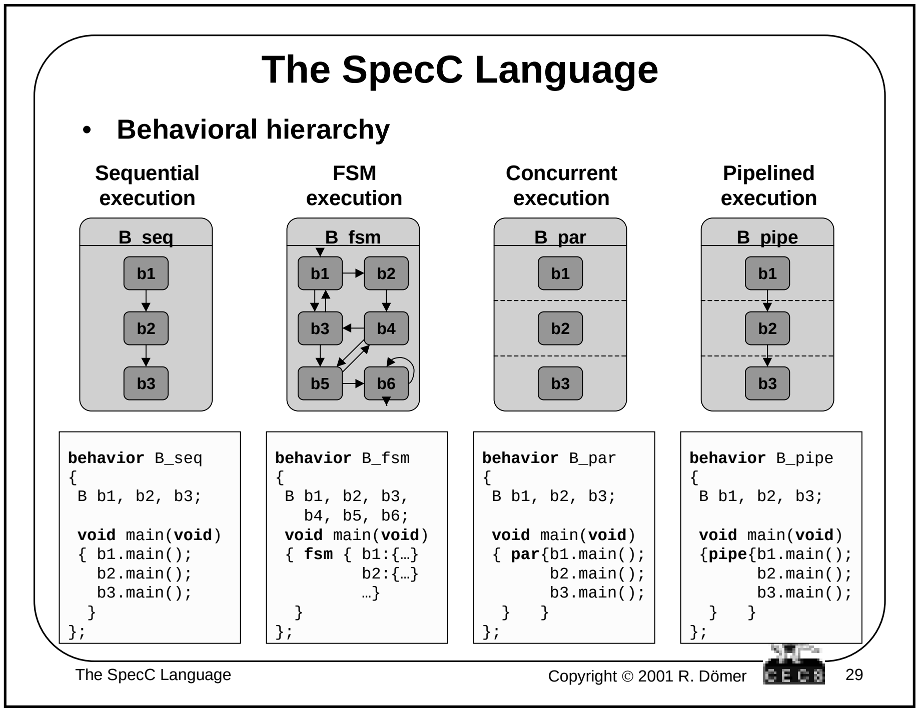
The SpecC Language

- **Behavioral hierarchy**

Sequential execution **FSM execution** **Concurrent execution** **Pipelined execution**

```
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:{…}
         …}
  }
};
```



The SpecC Language

- **Behavioral hierarchy**

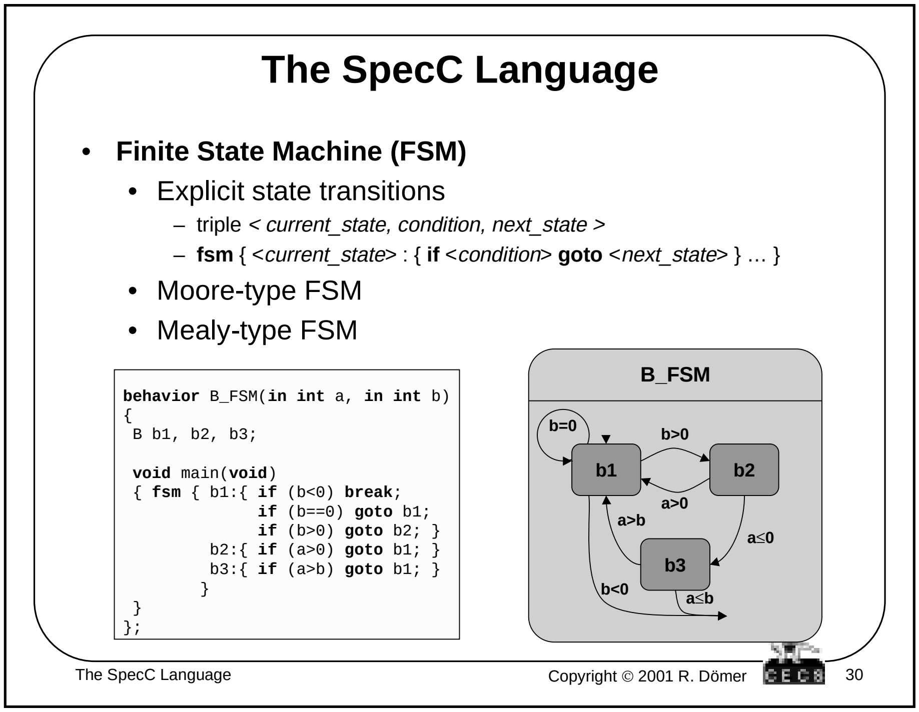
Sequential execution
FSM execution
Concurrent execution
Pipelined execution
B_seq
b1
b2
b3
B_fsm
b1
b2
b3
b4
b5
b6
B_par
b1
b2
b3
B_pipe
b1
b2
b3

```
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();
  }   }
};
```

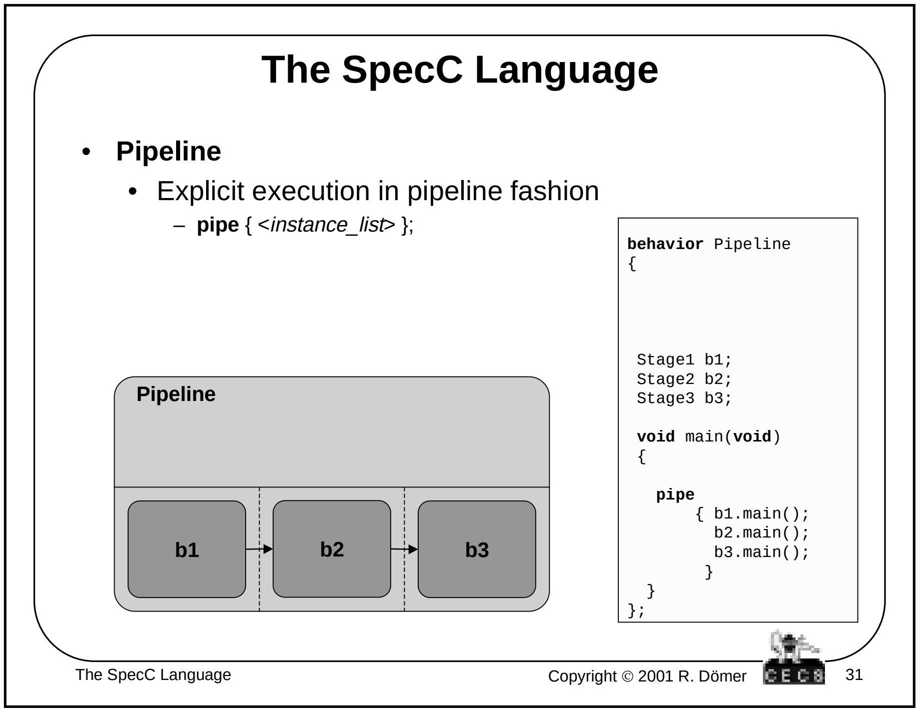


The SpecC Language

- **Finite State Machine (FSM)**
  - Explicit state transitions
    - triple *< current_state, condition, next_state >*
    - **fsm** { <*current_state*> : { **if** <*condition*> **goto** <*next_state*> } … }
  - Moore-type FSM
  - Mealy-type FSM

```
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; }
        }
 }
};
```

B_FSM
b=0
b>0
b1
b2
a>0
a>b
b3
a≤0
b<0
a≤b



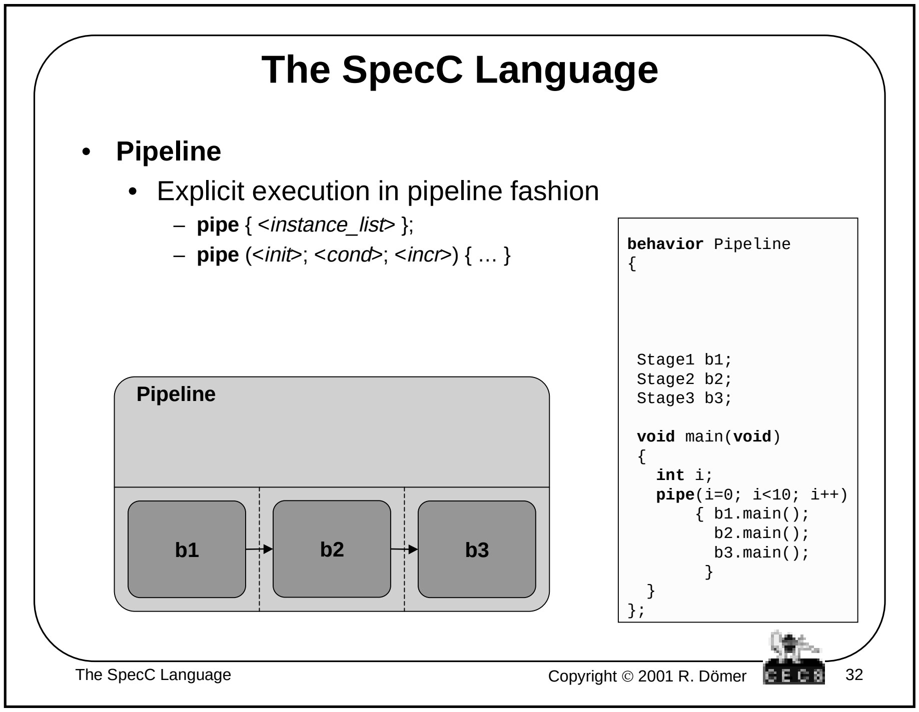
The SpecC Language

- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };

Pipeline
b1
b2
b3

```
behavior Pipeline
{



 Stage1 b1;
 Stage2 b2;
 Stage3 b3;

 void main(void)
 {

   pipe
       { b1.main();
         b2.main();
         b3.main();
        }
  }
};
```

The SpecC Language Copyright © 2001 R. Dömer 31



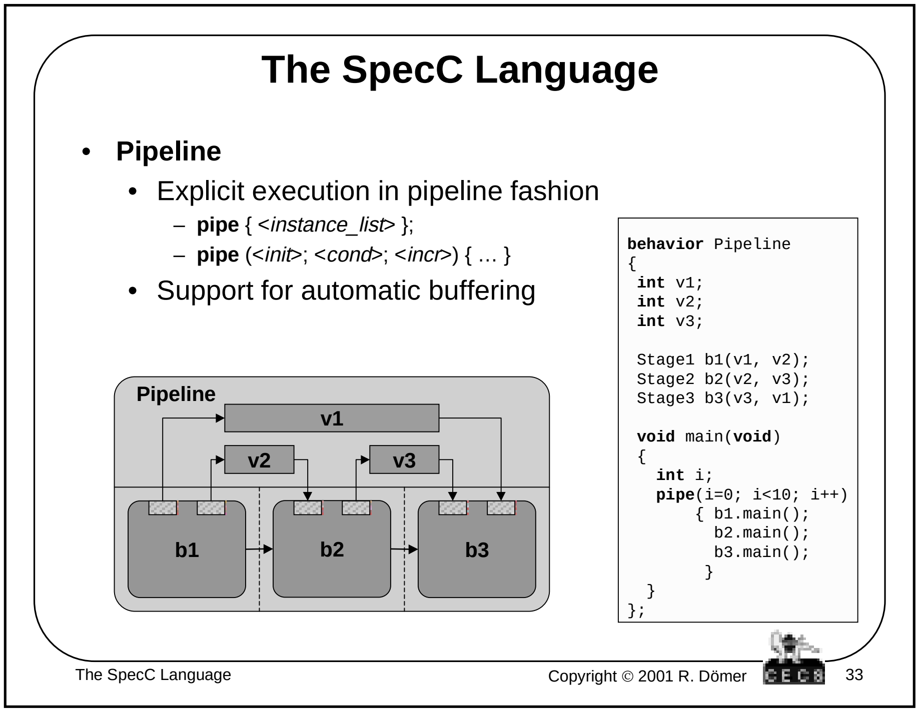
The SpecC Language

- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }

Pipeline
b1
b2
b3

```
behavior Pipeline
{



 Stage1 b1;
 Stage2 b2;
 Stage3 b3;

 void main(void)
 {
   int i;
   pipe(i=0; i<10; i++)
       { b1.main();
         b2.main();
         b3.main();
        }
  }
};
```

The SpecC Language Copyright © 2001 R. Dömer 32


Pipeline
v1
v2
v3
b1
b2
b3

The SpecC Language

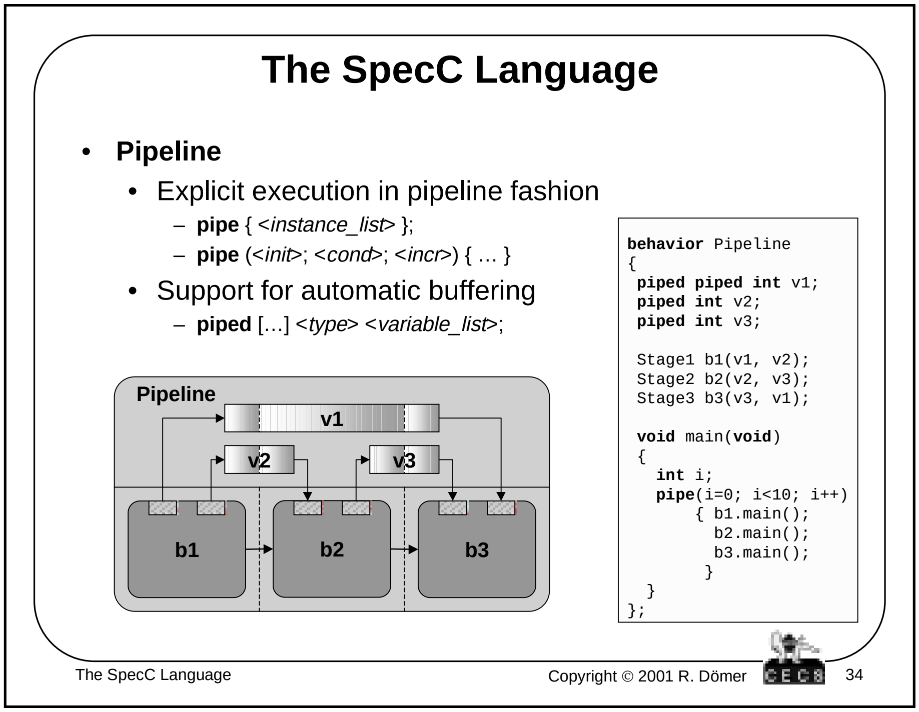
- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }
  - Support for automatic buffering

```
behavior Pipeline
{
 int v1;
 int v2;
 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();
        }
  }
};
```

The SpecC Language

Copyright © 2001 R. Dömer

33


Pipeline
v1
v2
v3
b1
b2
b3

The SpecC Language

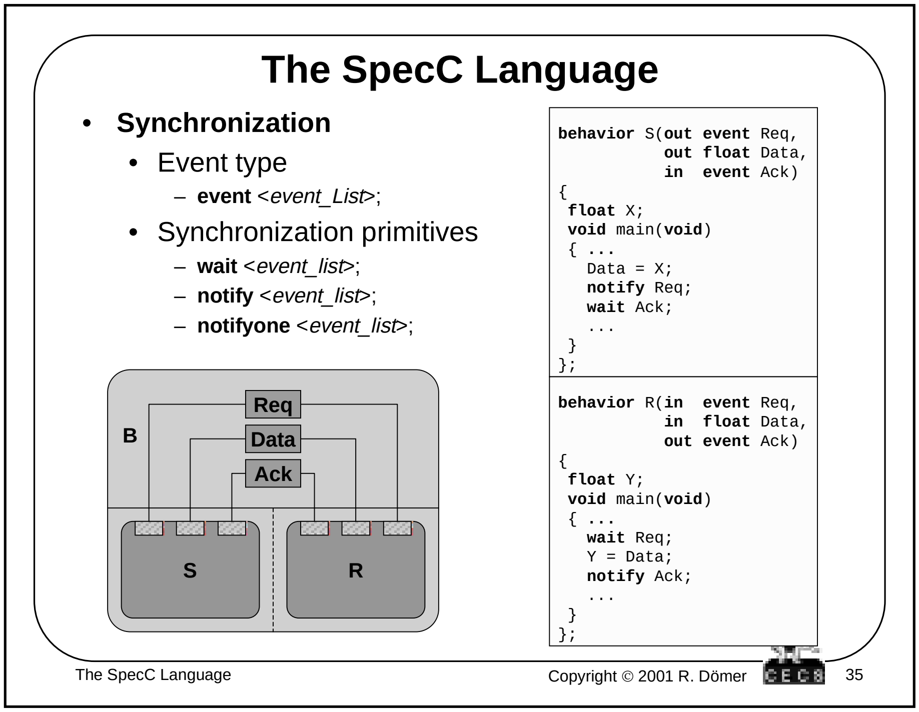
- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }
  - Support for automatic buffering
    - **piped** […] <*type*> <*variable_list*>;

```
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();
        }
  }
};
```

The SpecC Language

Copyright © 2001 R. Dömer

34



The SpecC Language

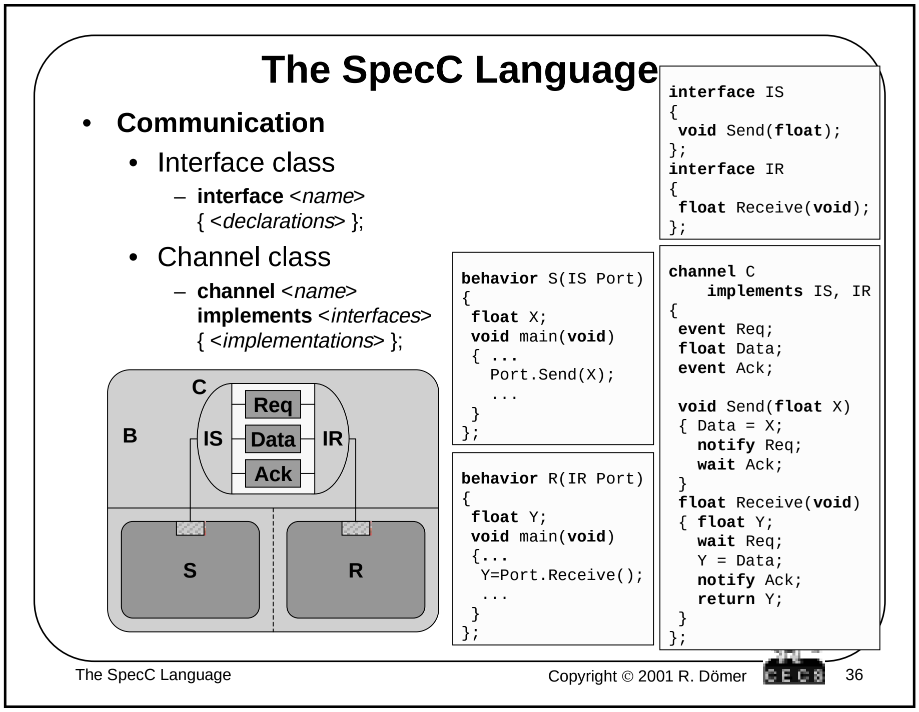
- **Synchronization**
  - Event type
    - **event** <*event_List*>;
  - Synchronization primitives
    - **wait** <*event_list*>;
    - **notify** <*event_list*>;
    - **notifyone** <*event_list*>;

B
Req
Data
Ack
S
R

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

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

The SpecC Language Copyright © 2001 R. Dömer 35



The SpecC Language

- **Communication**
  - Interface class
    - **interface** <*name*>
      { <*declarations*> };
  - Channel class
    - **channel** <*name*>
      implements <*interfaces*>
      { <*implementations*> };

B
C
IS
IR
Req
Data
Ack
S
R

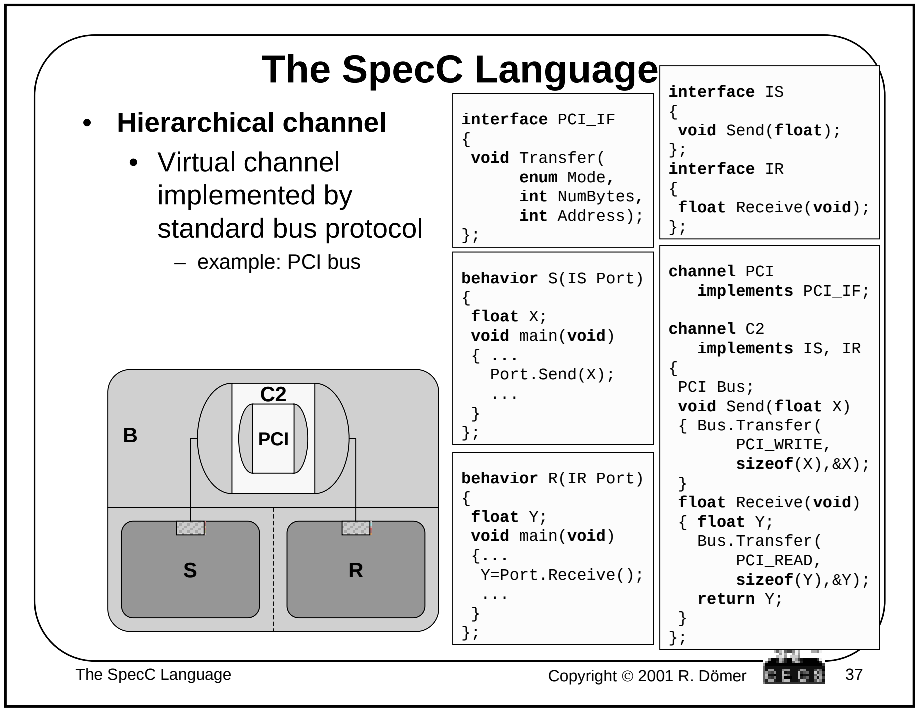
```
interface IS
{
 void Send(float);
};
interface IR
{
 float Receive(void);
};
```

```
behavior S(IS Port)
{
 float X;
 void main(void)
 { ...
   Port.Send(X);
   ...
 }
};
```

```
behavior R(IR Port)
{
 float Y;
 void main(void)
 {...
  Y=Port.Receive();
  ...
 }
};
```

```
channel C
    implements IS, IR
{
 event Req;
 float Data;
 event Ack;

 void Send(float X)
 { Data = X;
   notify Req;
   wait Ack;
 }
 float Receive(void)
 { float Y;
   wait Req;
   Y = Data;
   notify Ack;
   return Y;
 }
};
```

The SpecC Language Copyright © 2001 R. Dömer 36



The SpecC Language

- **Hierarchical channel**
  - Virtual channel implemented by standard bus protocol
    - example: PCI bus

```
interface PCI_IF
{
 void Transfer(
      enum Mode,
      int NumBytes,
      int Address);
};
```

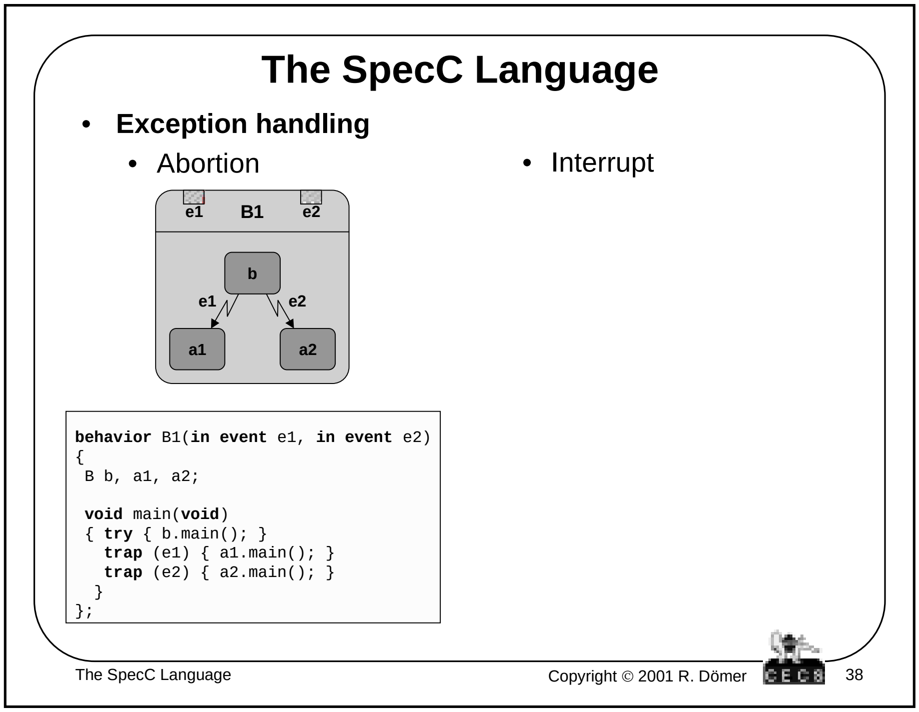
```
behavior S(IS Port)
{
 float X;
 void main(void)
 { ...
   Port.Send(X);
   ...
 }
};
```

```
behavior R(IR Port)
{
 float Y;
 void main(void)
 {...
  Y=Port.Receive();
  ...
 }
};
```

```
interface IS
{
 void Send(float);
};
interface IR
{
 float Receive(void);
};
```

```
channel PCI
   implements PCI_IF;

channel C2
   implements IS, IR
{
 PCI Bus;
 void Send(float X)
 { Bus.Transfer(
       PCI_WRITE,
       sizeof(X),&X);
 }
 float Receive(void)
 { float Y;
   Bus.Transfer(
       PCI_READ,
       sizeof(Y),&Y);
   return Y;
 }
};
```

The SpecC Language Copyright © 2001 R. Dömer 37



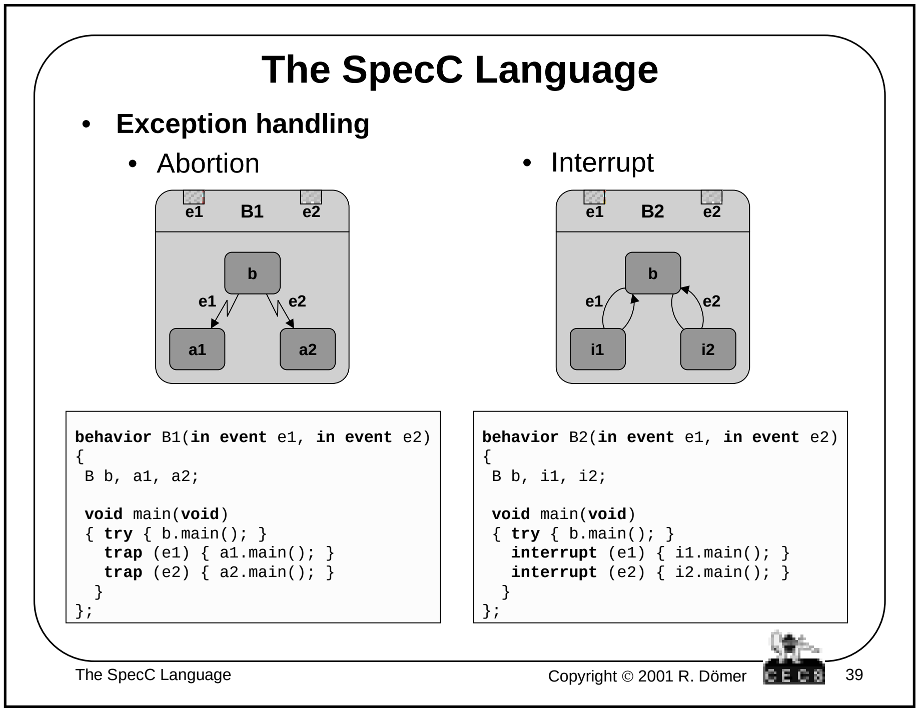
The SpecC Language

- **Exception handling**
  - Abortion
  - Interrupt

e1
B1
e2
b
e1
e2
a1
a2

```
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(); }
  }
};
```

The SpecC Language Copyright © 2001 R. Dömer 38



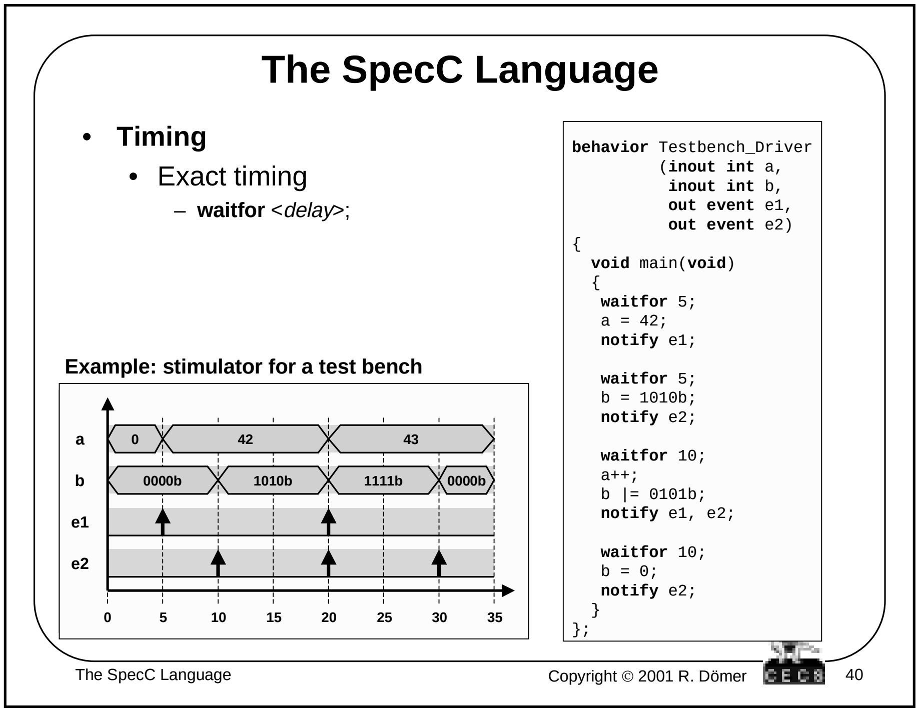
The SpecC Language

- **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, i1, i2;

 void main(void)
 { try { b.main(); }
   interrupt (e1) { i1.main(); }
   interrupt (e2) { i2.main(); }
  }
};
```

The SpecC Language Copyright © 2001 R. Dömer 39



The SpecC Language

- **Timing**
  - Exact timing
    - – **waitfor** <*delay*>;

Example: stimulator for a test bench

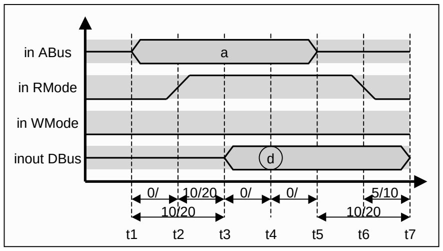
```
behavior Testbench_Driver
         (inout int a,
          inout int b,
          out event e1,
          out event e2)
{
  void main(void)
  {
   waitfor 5;
   a = 42;
   notify e1;

   waitfor 5;
   b = 1010b;
   notify e2;

   waitfor 10;
   a++;
   b |= 0101b;
   notify e1, e2;

   waitfor 10;
   b = 0;
   notify e2;
  }
};
```

The SpecC Language Copyright © 2001 R. Dömer 40

The SpecC Language

- **Timing**
  - Exact timing
    - **waitfor** <*delay*>;
  - Timing constraints
    - **do** { <*actions*> }
      timing {<*constraints*>}

Example: SRAM read protocol



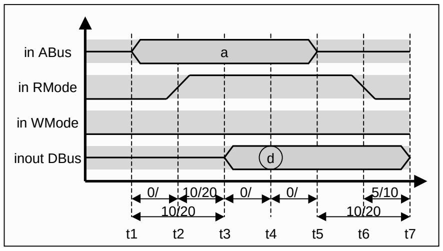
Specification

```
bit[7:0] Read_SRAM(bit[15:0] a)
{
 bit[7:0] d;

 do { t1: {ABus = a;  }
      t2: {RMode = 1;
           WMode = 0; }
      t3: {           }
      t4: {d = Dbus;  }
      t5: {ABus = 0;  }
      t6: {RMode = 0;
           WMode = 0; }
      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);
}
```

The SpecC Language

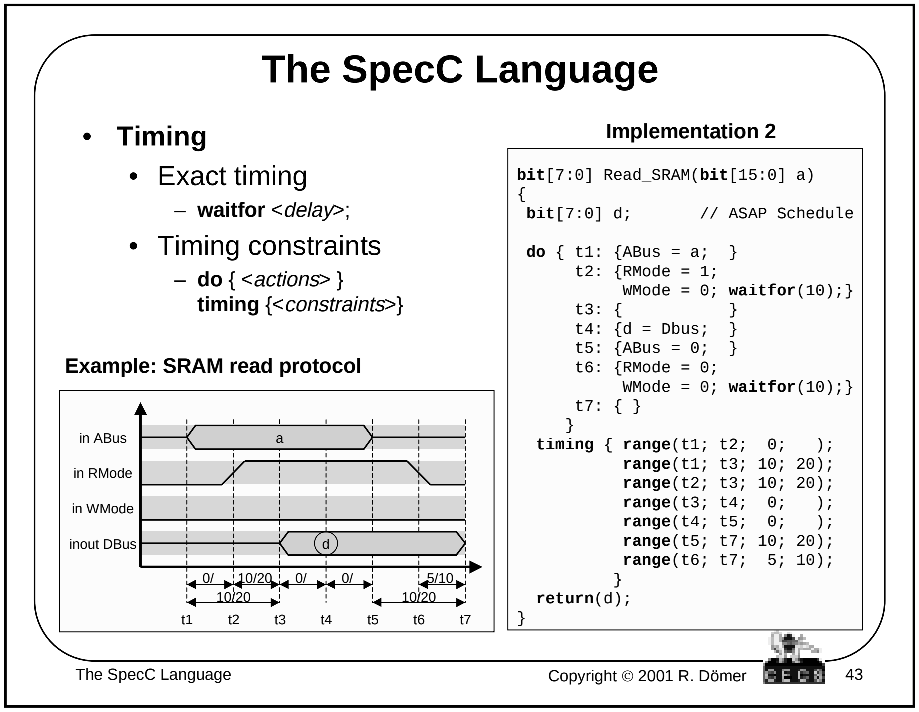
- **Timing**
  - Exact timing
    - **waitfor** <*delay*>;
  - Timing constraints
    - **do** { <*actions*> }
      timing {<*constraints*>}

Example: SRAM read protocol



Implementation 1

```
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);
}
```



The SpecC Language

- **Timing**
  - Exact timing
    - – **waitfor** <*delay*>;
  - Timing constraints
    - – **do** { <*actions*> }
      timing {<*constraints*>}

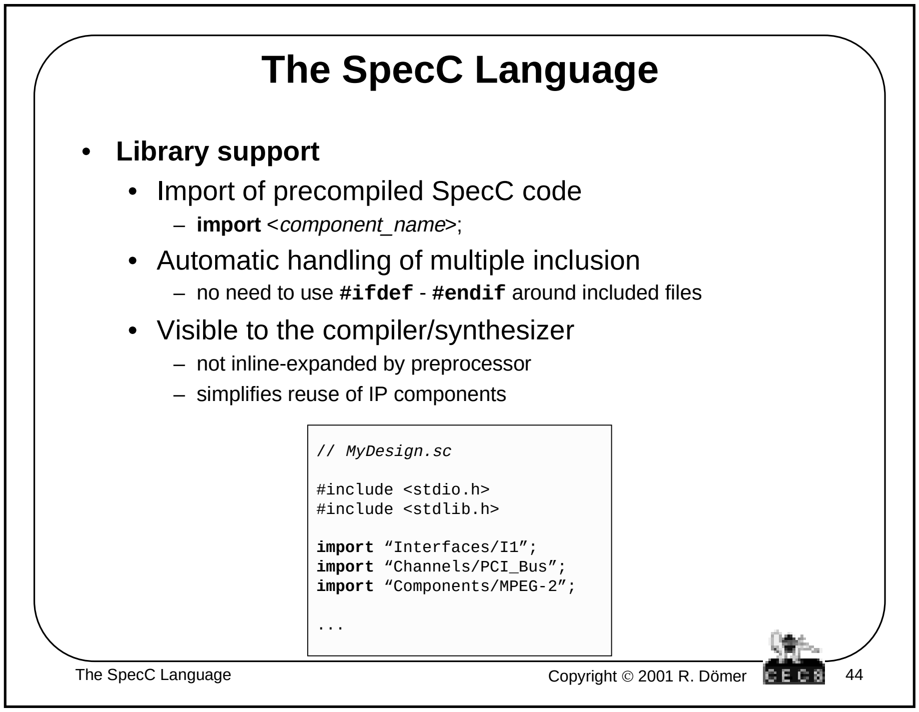
Example: SRAM read protocol

in ABus
in RMode
in WMode
inout DBus
a
d
0/
10/20
0/
0/
5/10
10/20
10/20
t1
t2
t3
t4
t5
t6
t7

Implementation 2

```
bit[7:0] Read_SRAM(bit[15:0] a)
{
 bit[7:0] d;       // ASAP Schedule

 do { t1: {ABus = a;  }
      t2: {RMode = 1;
           WMode = 0; waitfor(10);}
      t3: {           }
      t4: {d = Dbus;  }
      t5: {ABus = 0;  }
      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);
}
```

The SpecC Language   Copyright © 2001 R. Dömer   43

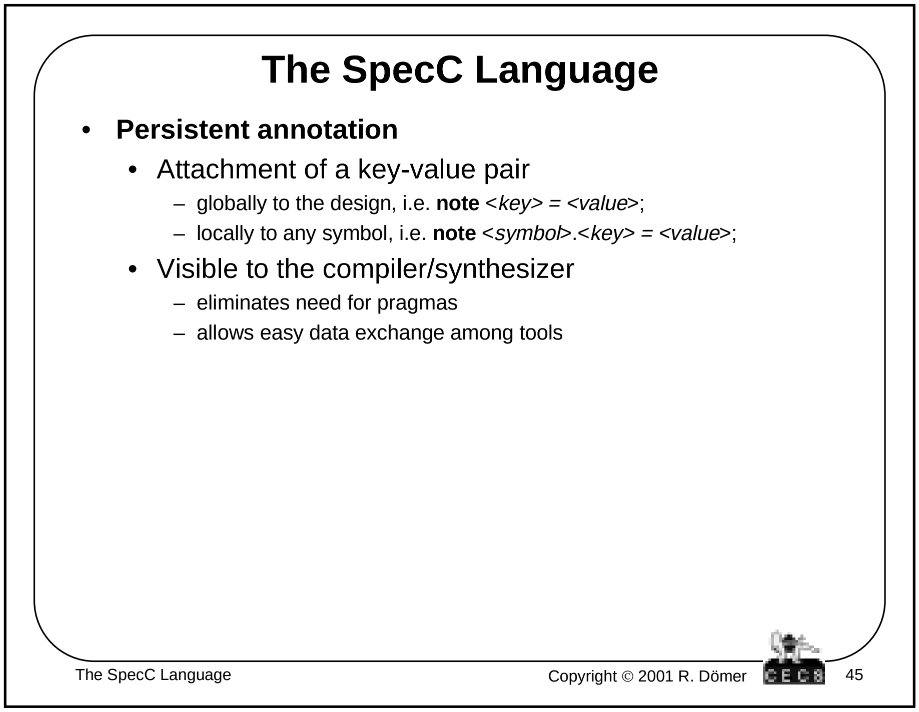


The SpecC Language

- **Library support**
  - Import of precompiled SpecC code
    - – **import** <*component_name*>;
  - Automatic handling of multiple inclusion
    - – no need to use **`#ifdef`** - **`#endif`** around included files
  - Visible to the compiler/synthesizer
    - – not inline-expanded by preprocessor
    - – simplifies reuse of IP components

```
// MyDesign.sc

#include <stdio.h>
#include <stdlib.h>

import “Interfaces/I1”;
import “Channels/PCI_Bus”;
import “Components/MPEG-2”;

...
```

The SpecC Language   Copyright © 2001 R. Dömer   44

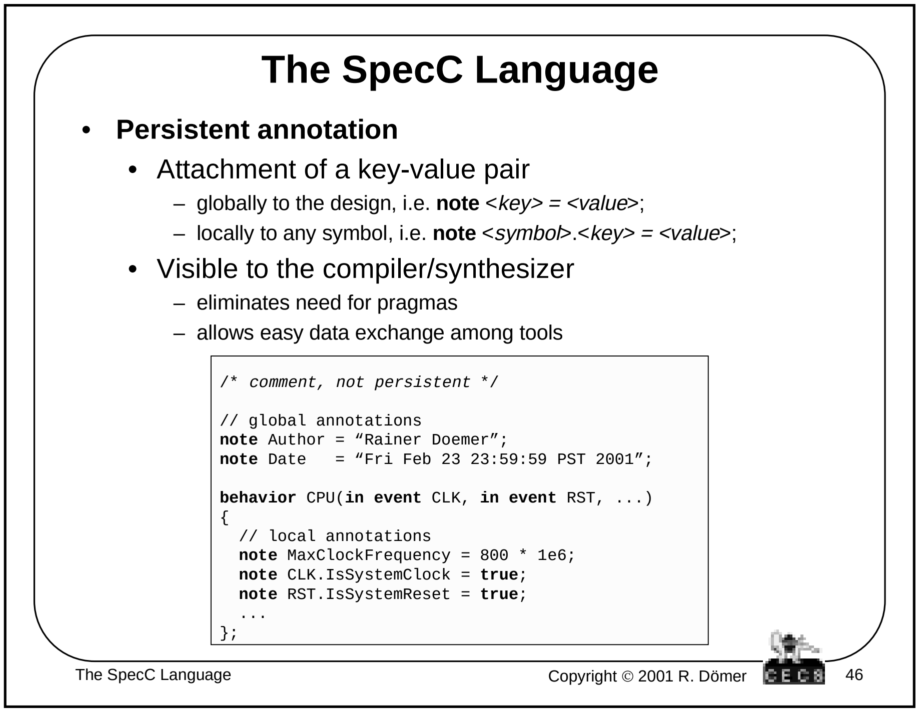


The SpecC Language

- **Persistent annotation**
  - Attachment of a key-value pair
    - globally to the design, i.e. **note** <*key*> = <*value*>;
    - locally to any symbol, i.e. **note** <*symbol*>.<*key*> = <*value*>;
  - Visible to the compiler/synthesizer
    - eliminates need for pragmas
    - allows easy data exchange among tools

The SpecC Language

Copyright © 2001 R. Dömer 45



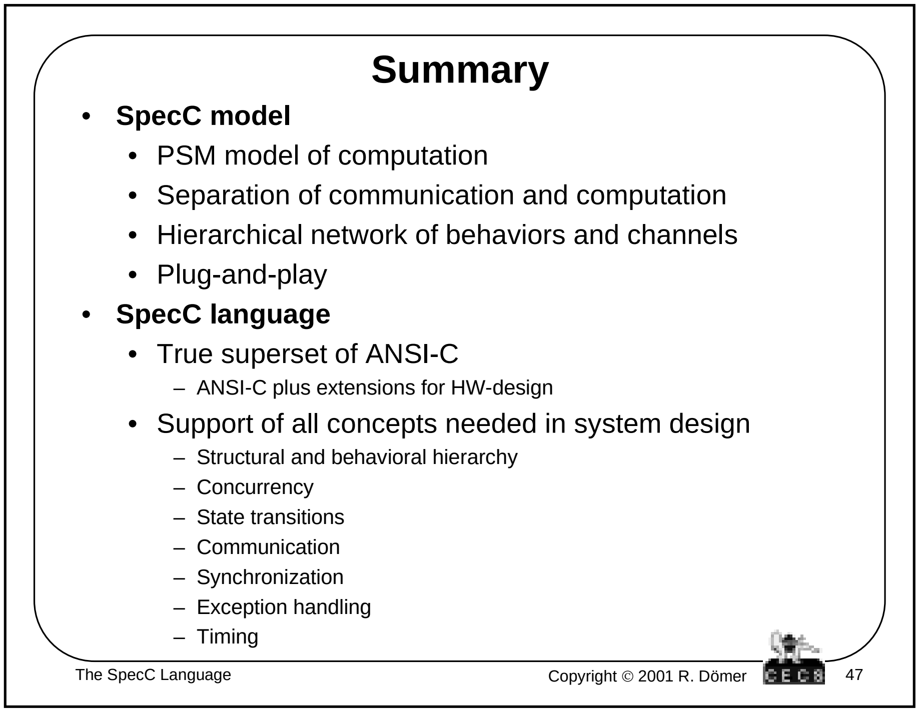
The SpecC Language

- **Persistent annotation**
  - Attachment of a key-value pair
    - globally to the design, i.e. **note** <*key*> = <*value*>;
    - locally to any symbol, i.e. **note** <*symbol*>.<*key*> = <*value*>;
  - Visible to the compiler/synthesizer
    - eliminates need for pragmas
    - allows easy data exchange among tools

```
/* comment, not persistent */

// global annotations
note Author = “Rainer Doemer”;
note Date   = “Fri Feb 23 23:59:59 PST 2001”;

behavior CPU(in event CLK, in event RST, ...)
{
  // local annotations
  note MaxClockFrequency = 800 * 1e6;
  note CLK.IsSystemClock = true;
  note RST.IsSystemReset = true;
  ...
};
```

The SpecC Language

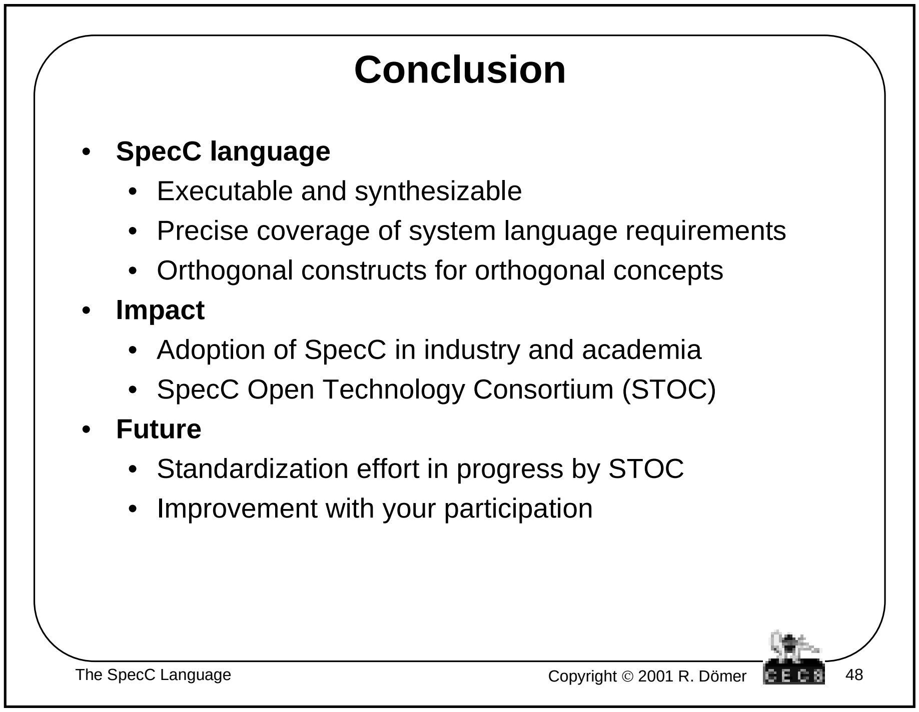
Copyright © 2001 R. Dömer 46



Summary

- **SpecC model**
  - PSM model of computation
  - Separation of communication and computation
  - Hierarchical network of behaviors and channels
  - Plug-and-play
- **SpecC language**
  - True superset of ANSI-C
    - ANSI-C plus extensions for HW-design
  - Support of all concepts needed in system design
    - Structural and behavioral hierarchy
    - Concurrency
    - State transitions
    - Communication
    - Synchronization
    - Exception handling
    - Timing

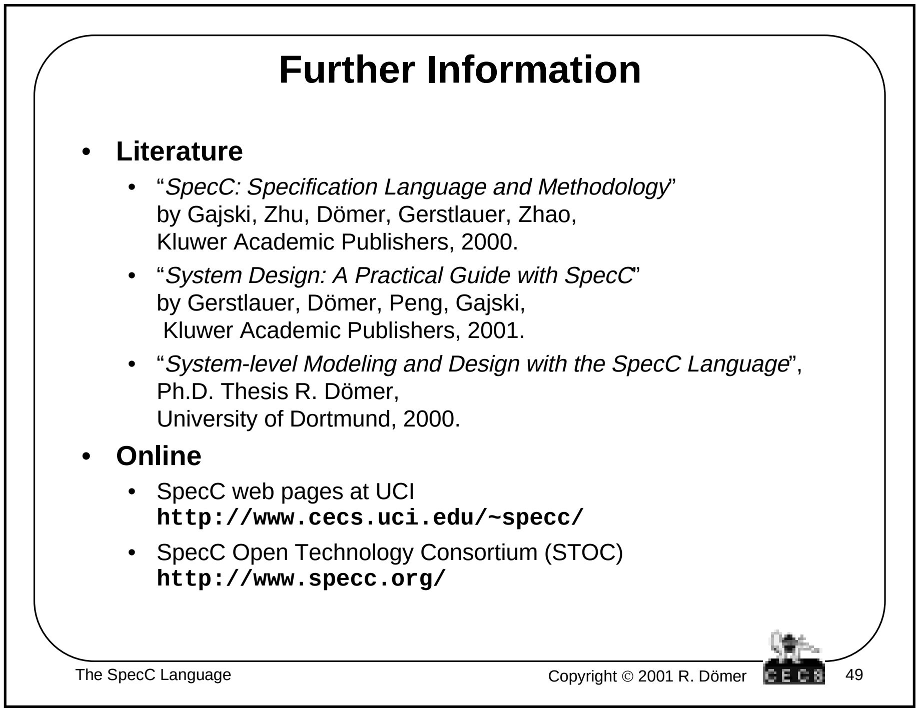
The SpecC Language Copyright © 2001 R. Dömer CECS 47



Conclusion

- **SpecC language**
  - Executable and synthesizable
  - Precise coverage of system language requirements
  - Orthogonal constructs for orthogonal concepts
- **Impact**
  - Adoption of SpecC in industry and academia
  - SpecC Open Technology Consortium (STOC)
- **Future**
  - Standardization effort in progress by STOC
  - Improvement with your participation

The SpecC Language Copyright © 2001 R. Dömer CECS 48



Further Information

- **Literature**
  - "*SpecC: Specification Language and Methodology*"
    by Gajski, Zhu, Dömer, Gerstlauer, Zhao,
    Kluwer Academic Publishers, 2000.
  - "*System Design: A Practical Guide with SpecC*"
    by Gerstlauer, Dömer, Peng, Gajski,
    Kluwer Academic Publishers, 2001.
  - "*System-level Modeling and Design with the SpecC Language*",
    Ph.D. Thesis R. Dömer,
    University of Dortmund, 2000.
- **Online**
  - SpecC web pages at UCI
    `http://www.cecs.uci.edu/~specc/`
  - SpecC Open Technology Consortium (STOC)
    `http://www.specc.org/`

The SpecC Language

Copyright © 2001 R. Dömer

49