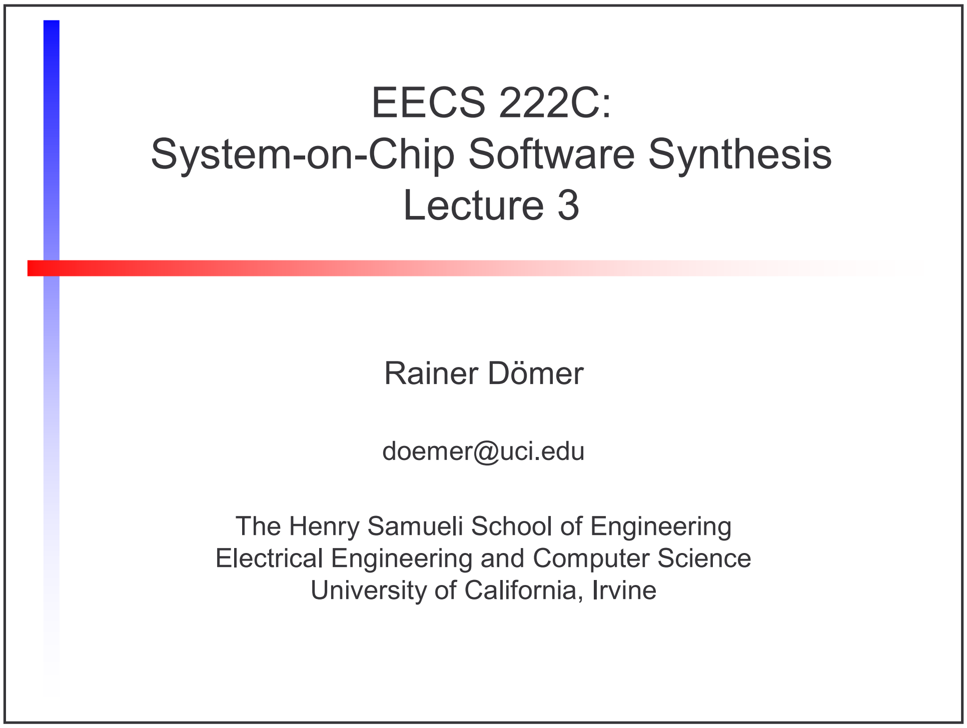
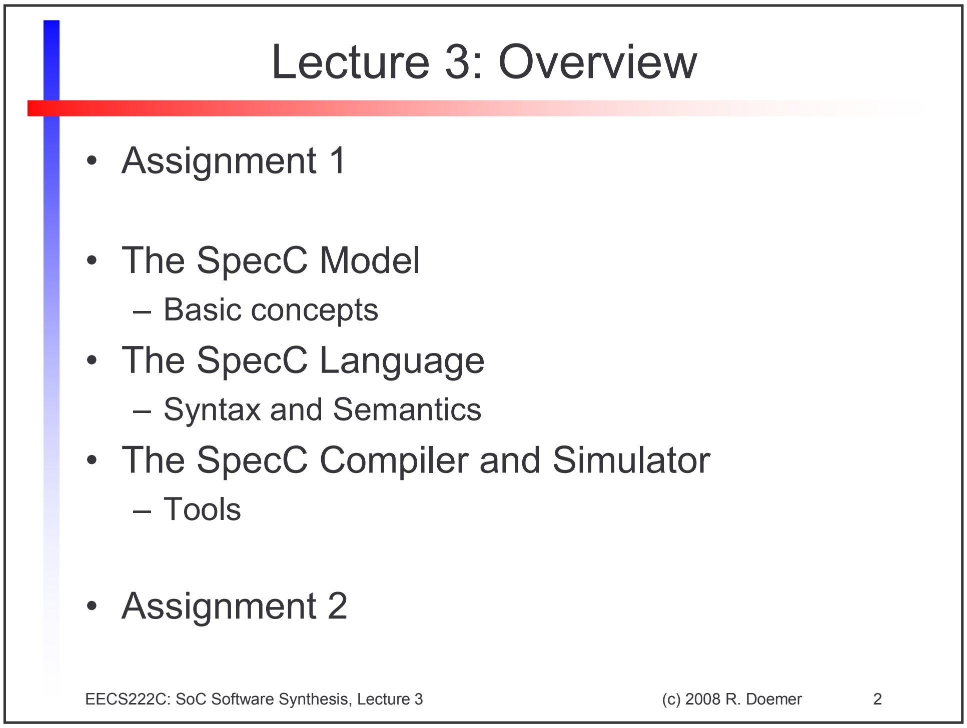


EECS 222C:
System-on-Chip Software Synthesis
Lecture 3

Rainer Dömer

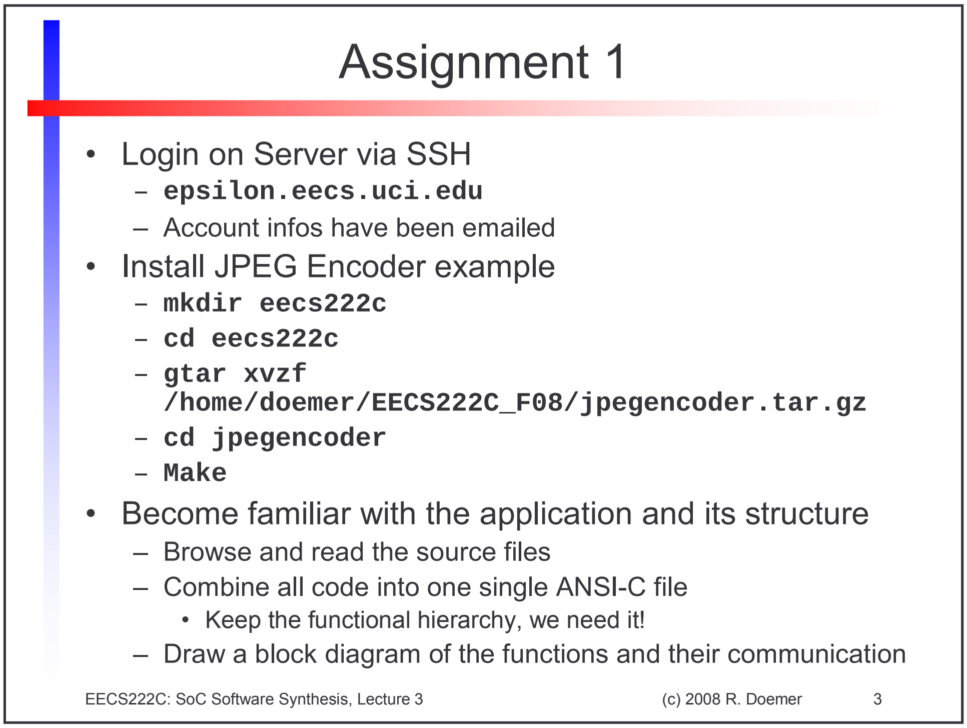
doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine



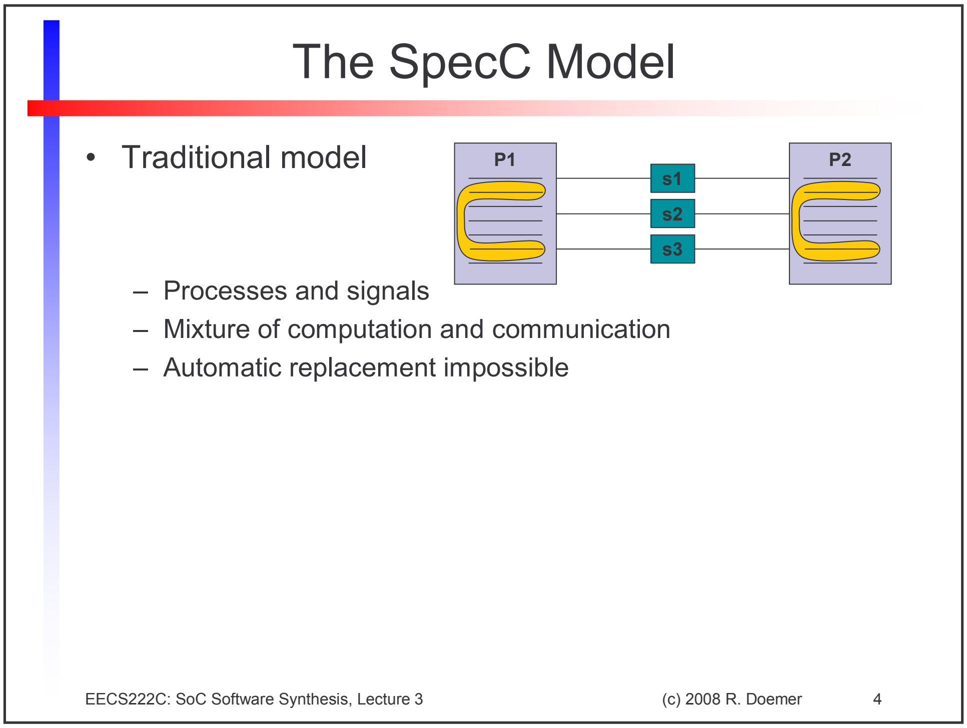
Lecture 3: Overview

- Assignment 1
- The SpecC Model
  - Basic concepts
- The SpecC Language
  - Syntax and Semantics
- The SpecC Compiler and Simulator
  - Tools
- Assignment 2



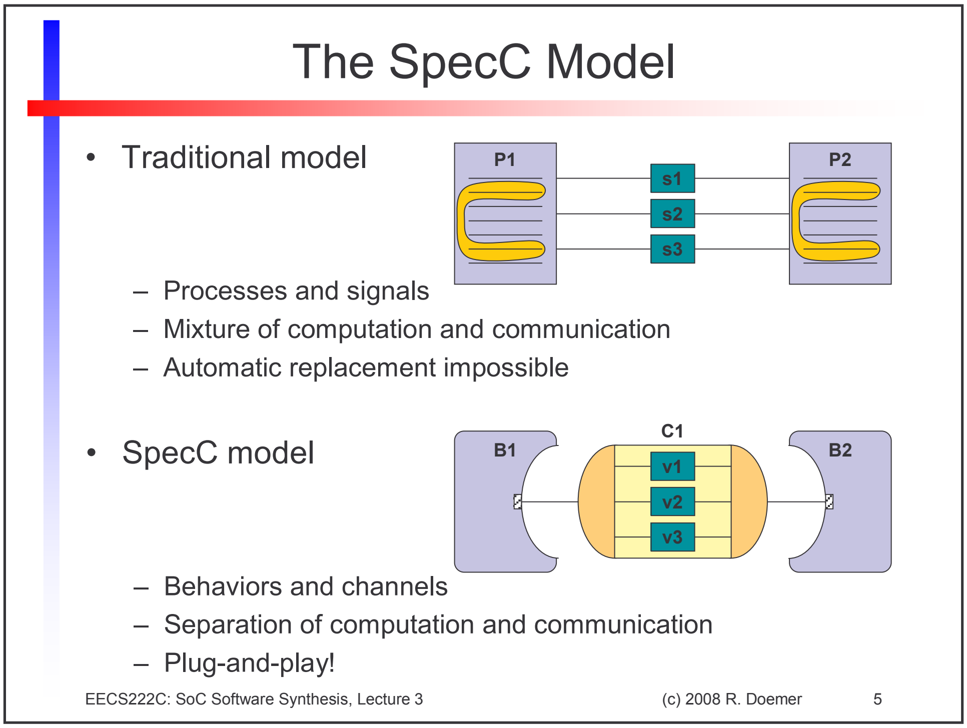
Assignment 1

- Login on Server via SSH
  - `epsilon.eecs.uci.edu`
  - Account infos have been emailed
- Install JPEG Encoder example
  - `mkdir eecs222c`
  - `cd eecs222c`
  - `gtar xvzf /home/doemer/EECS222C_F08/jpegencoder.tar.gz`
  - `cd jpegencoder`
  - `Make`
- Become familiar with the application and its structure
  - Browse and read the source files
  - Combine all code into one single ANSI-C file
    - Keep the functional hierarchy, we need it!
  - Draw a block diagram of the functions and their communication



The SpecC Model

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



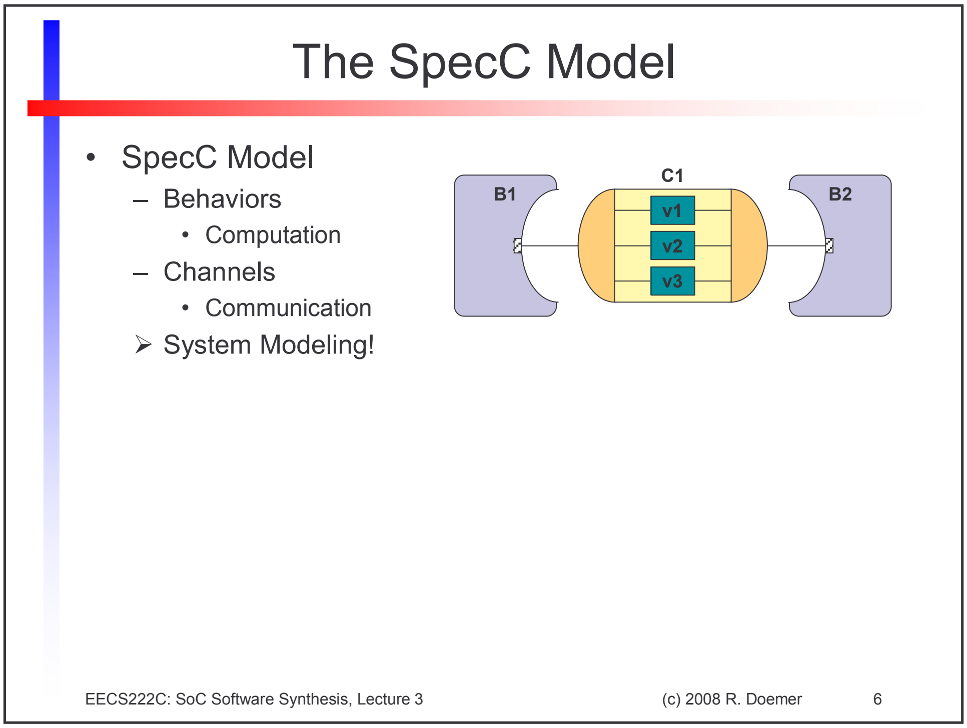
The SpecC Model

- 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!

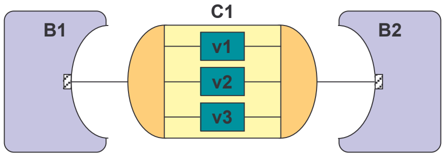


The SpecC Model

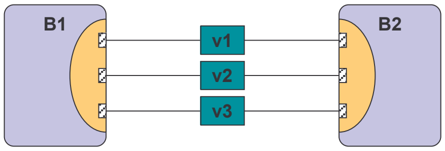
- SpecC Model
  - Behaviors
    - Computation
  - Channels
    - Communication
  - ➢ System Modeling!

The SpecC Model

- SpecC Model
  - Behaviors
    - Computation
  - Channels
    - Communication
  - ➢ System Modeling!



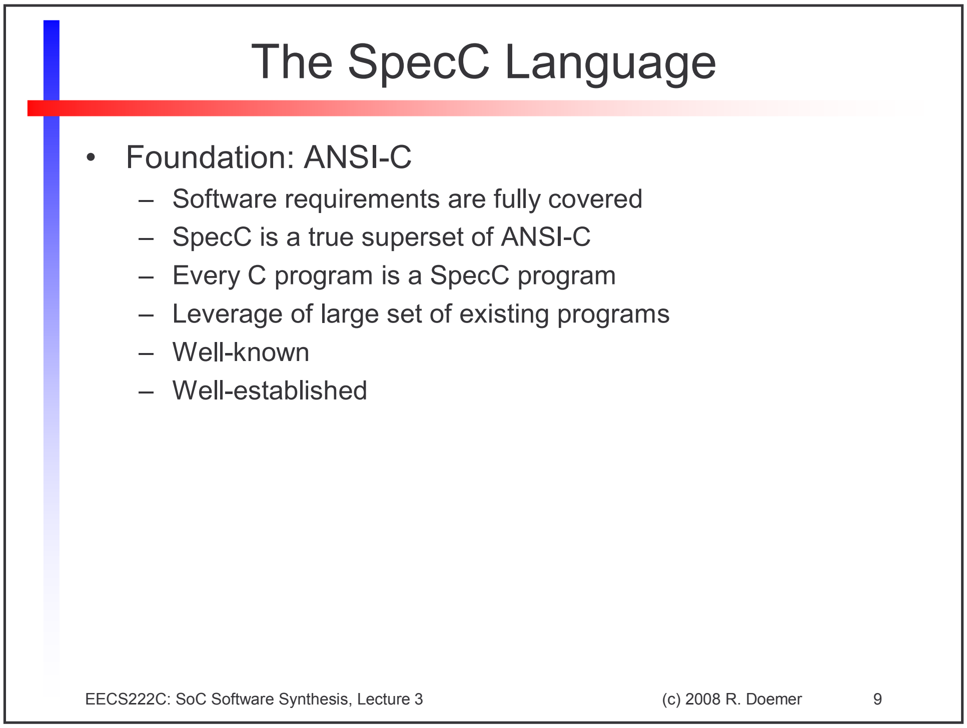
- Implementation through *Protocol Inlining*
  - Channel disappears
  - Communication is inlined into behaviors
  - Signals are exposed



➢ Model is converted to traditional model for implementation!

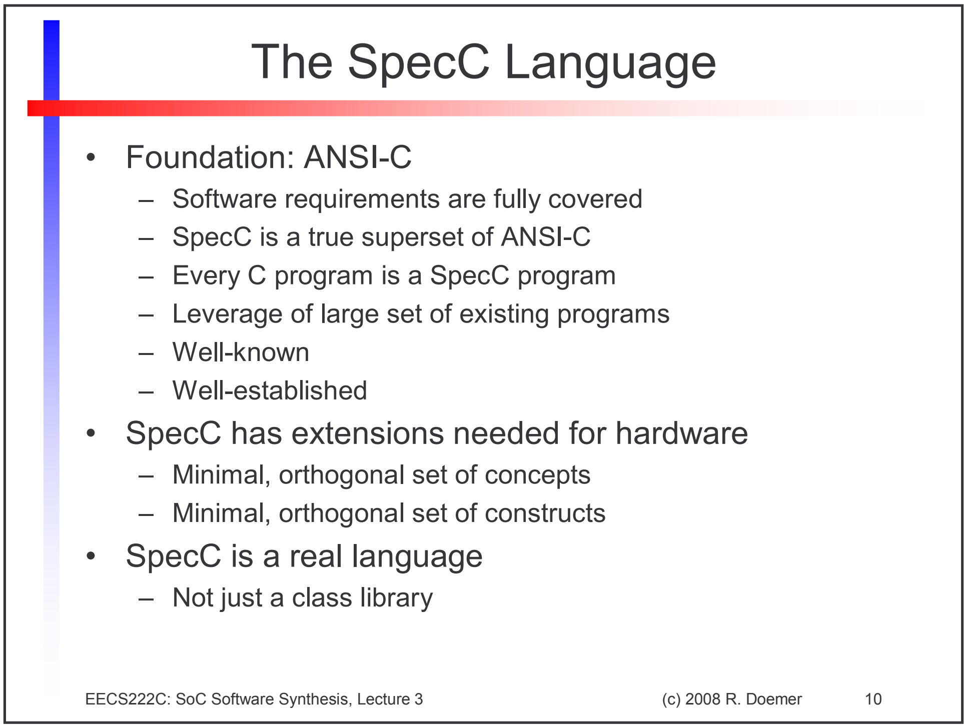
The SpecC Language

- Overview
  - Foundation
  - Types
  - Structural and behavioral hierarchy
  - Concurrency
  - State transitions
  - Exception handling
  - Communication
  - Synchronization
  - Timing
  - (RTL)



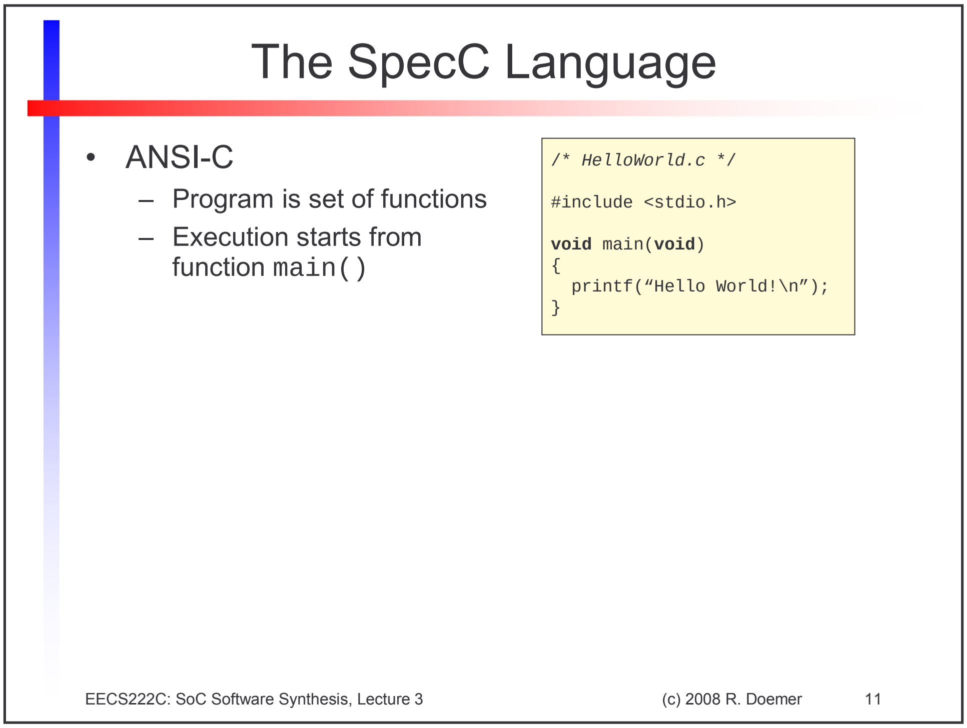
The SpecC Language

- 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

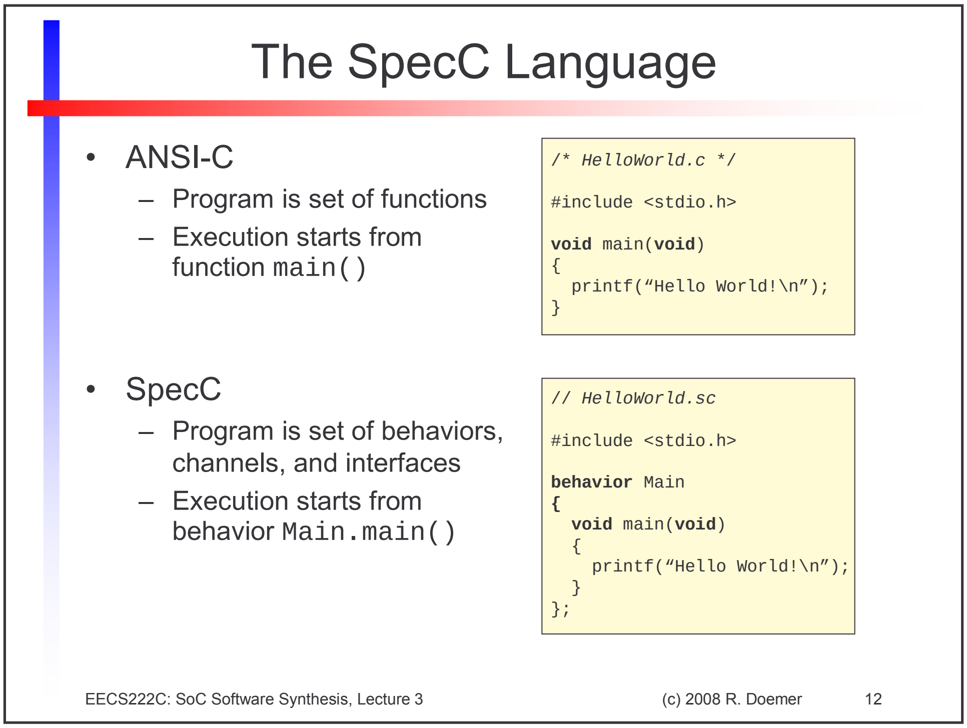
- Foundation: ANSI-C
  - Software requirements are fully covered
  - SpecC is a true superset of ANSI-C
  - Every C program is a SpecC program
  - Leverage of large set of existing programs
  - Well-known
  - Well-established
- SpecC has extensions needed for hardware
  - Minimal, orthogonal set of concepts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



The SpecC Language

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

```
/* HelloWorld.c */

#include <stdio.h>

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



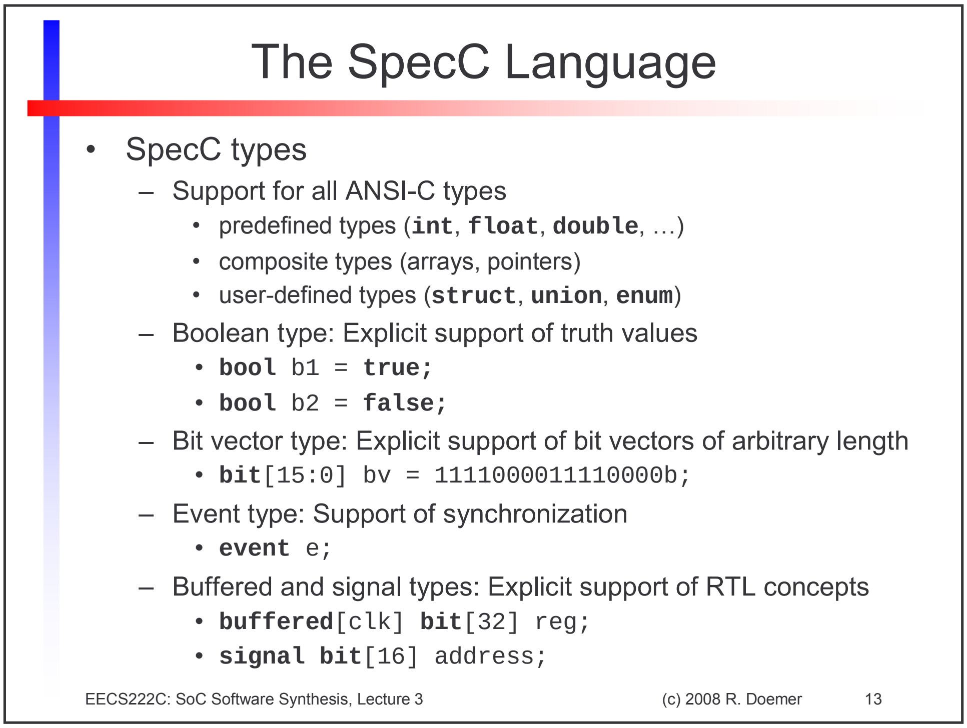
The SpecC Language

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

```
/* HelloWorld.c */

#include <stdio.h>

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

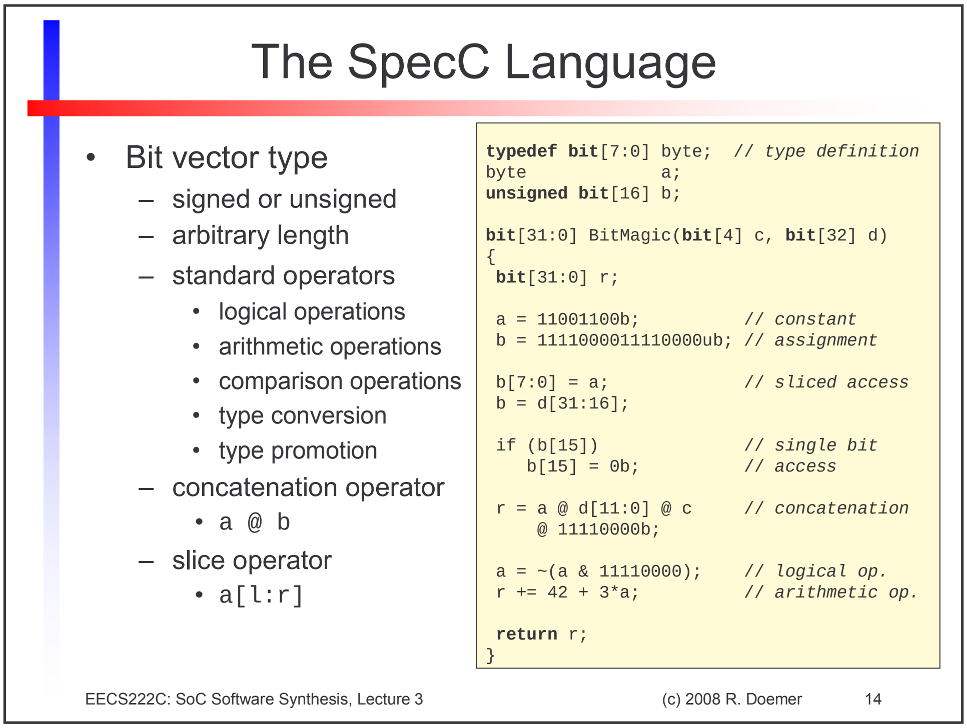
- SpecC
  - Program is set of behaviors, channels, and interfaces
  - Execution starts from behavior `Main.main()`

```
// HelloWorld.sc

#include <stdio.h>

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



The SpecC Language

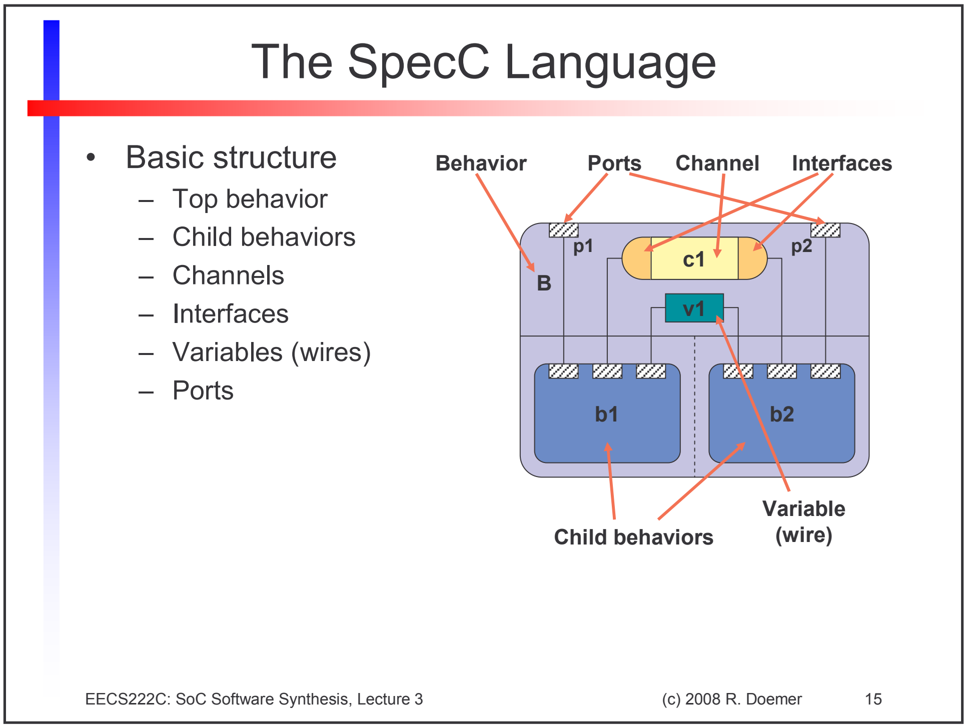
- SpecC types
  - Support for all ANSI-C types
    - predefined types (**`int`**, **`float`**, **`double`**, …)
    - composite types (arrays, pointers)
    - user-defined types (**`struct`**, **`union`**, **`enum`**)
  - Boolean type: Explicit support of truth values
    - `bool b1 = true;`
    - `bool b2 = false;`
  - Bit vector type: Explicit support of bit vectors of arbitrary length
    - `bit[15:0] bv = 1111000011110000b;`
  - Event type: Support of synchronization
    - `event e;`
  - Buffered and signal types: Explicit support of RTL concepts
    - `buffered[clk] bit[32] reg;`
    - `signal bit[16] address;`



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;
}
```

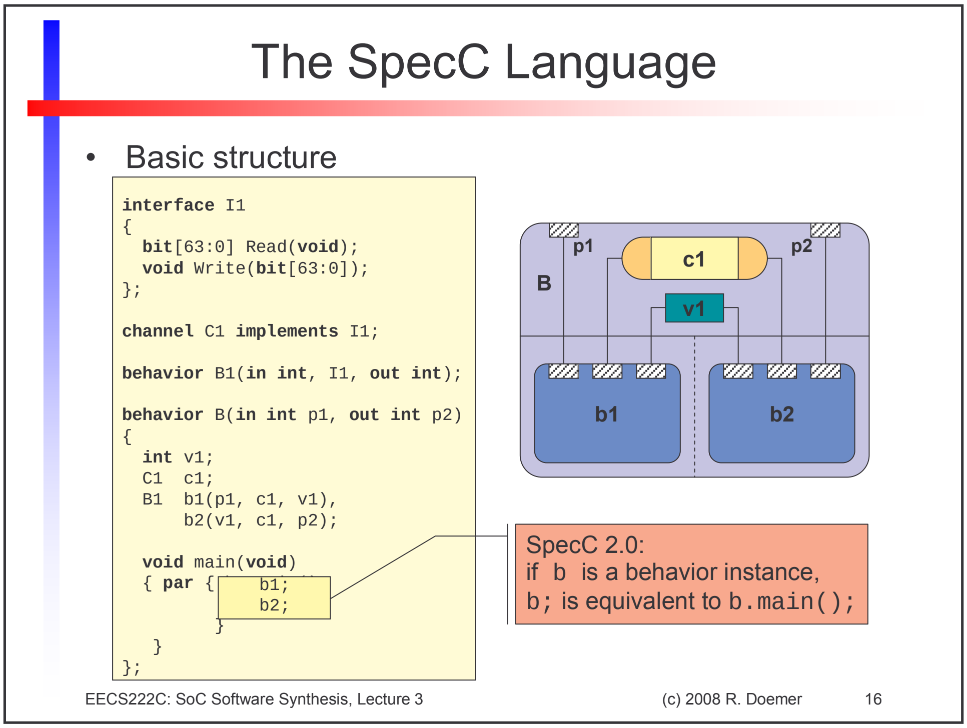


The SpecC Language

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

Behavior
Ports
Channel
Interfaces
B
p1
c1
p2
v1
b1
b2
Child behaviors
Variable
(wire)

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 15



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;
           b2;
         }
  }
};
```

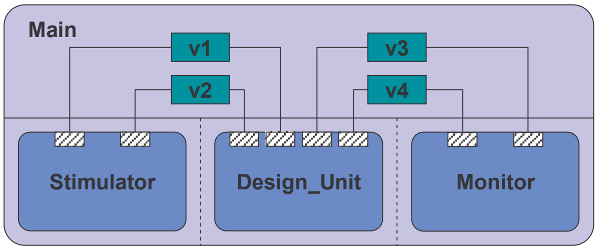
B
p1
c1
p2
v1
b1
b2

SpecC 2.0:
if b is a behavior instance,
b; is equivalent to b.main();

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 16

The SpecC Language

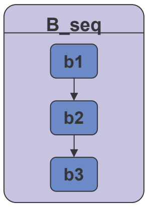
- Typical test bench
  - Top-level behavior: `Main`
  - Stimulator provides test vectors
  - Design unit under test
  - Monitor observes and checks outputs



The SpecC Language

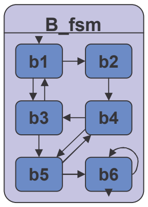
- Behavioral hierarchy

Sequential execution



```
behavior B_seq
{
 B b1, b2, b3;

 void main(void)
 {  b1;
    b2;
    b3;
  }
};
```

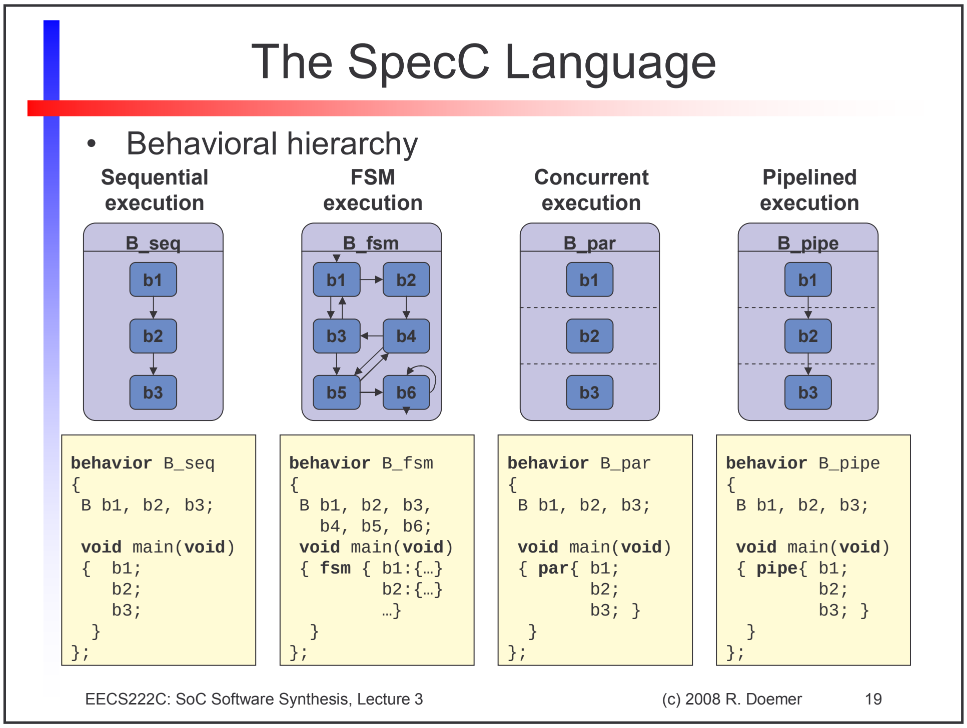
FSM execution



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

Concurrent execution

Pipelined execution



The SpecC Language

- Behavioral hierarchy

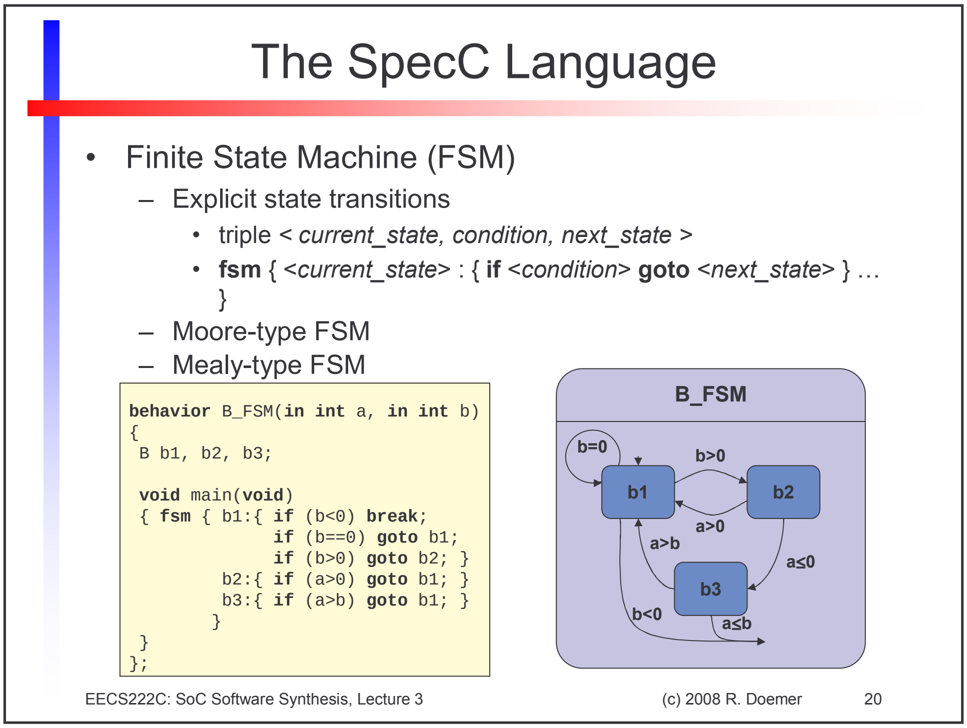
```
behavior B_seq
{
 B b1, b2, b3;

 void main(void)
 {  b1;
    b2;
    b3;
  }
};
```

```
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;
        b2;
        b3; }
  }
};
```

```
behavior B_pipe
{
 B b1, b2, b3;

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

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 19

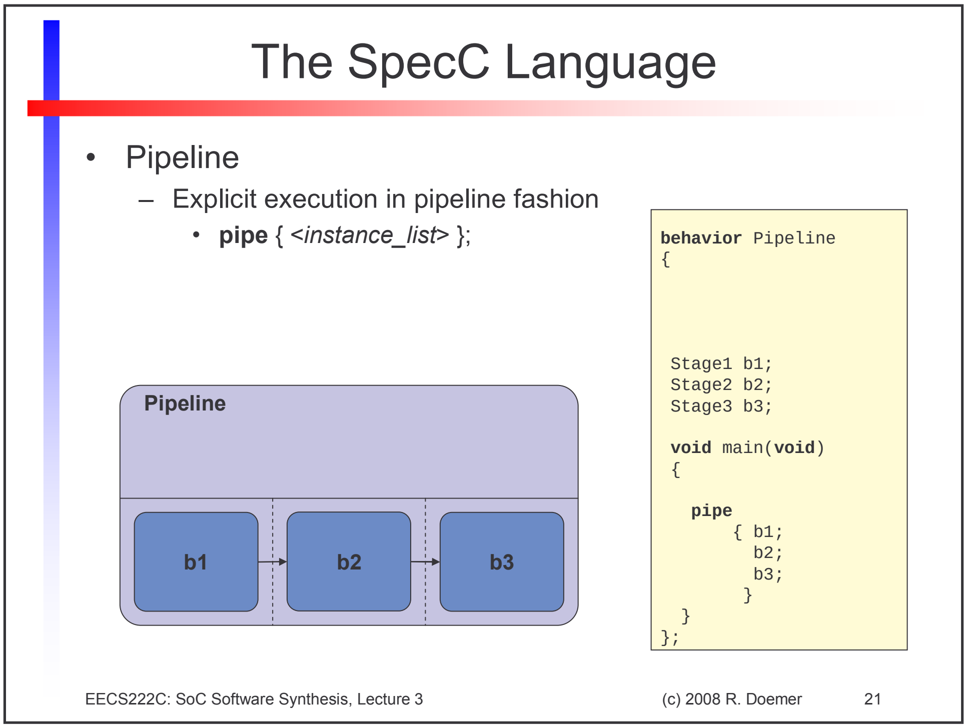


The SpecC Language

- 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; }
        }
 }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 20



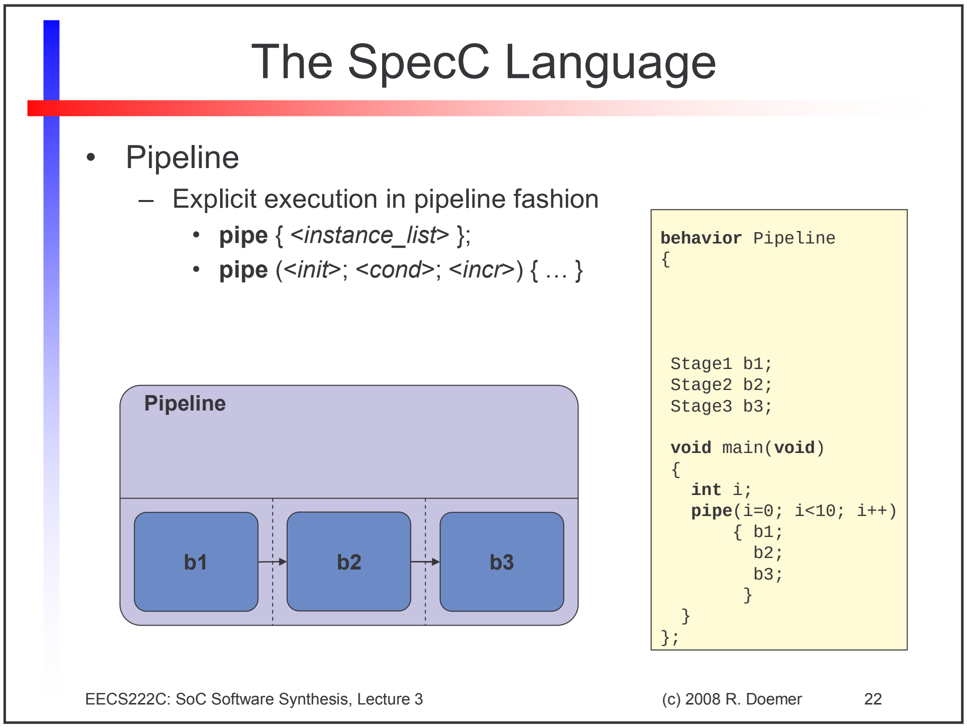
The SpecC Language

- 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;
         b2;
         b3;
        }
  }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 21



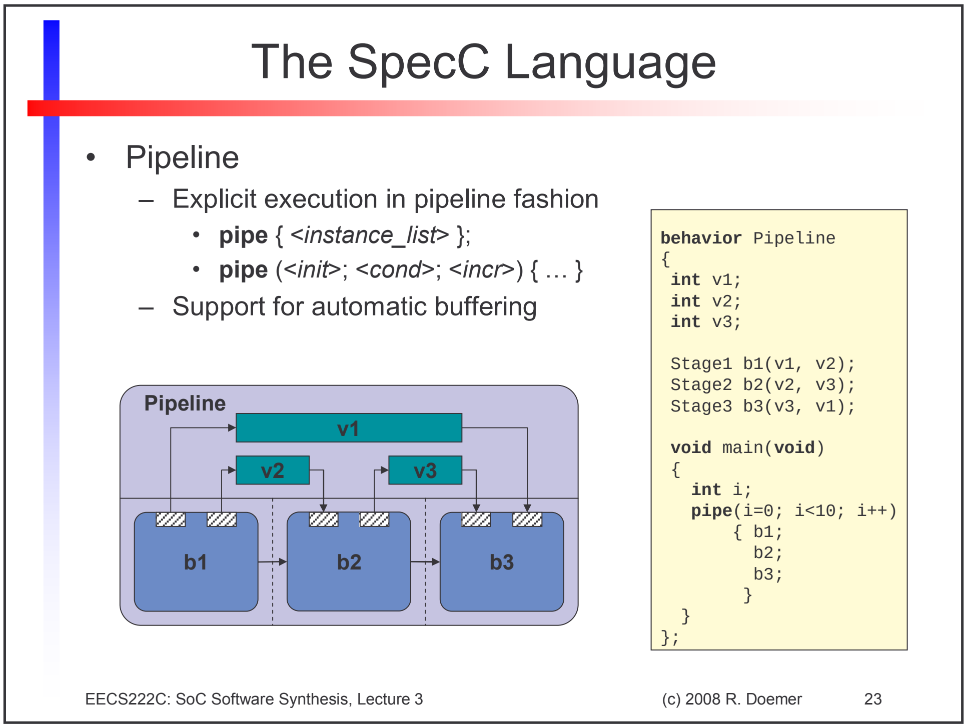
The SpecC Language

- 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;
         b2;
         b3;
        }
  }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 22



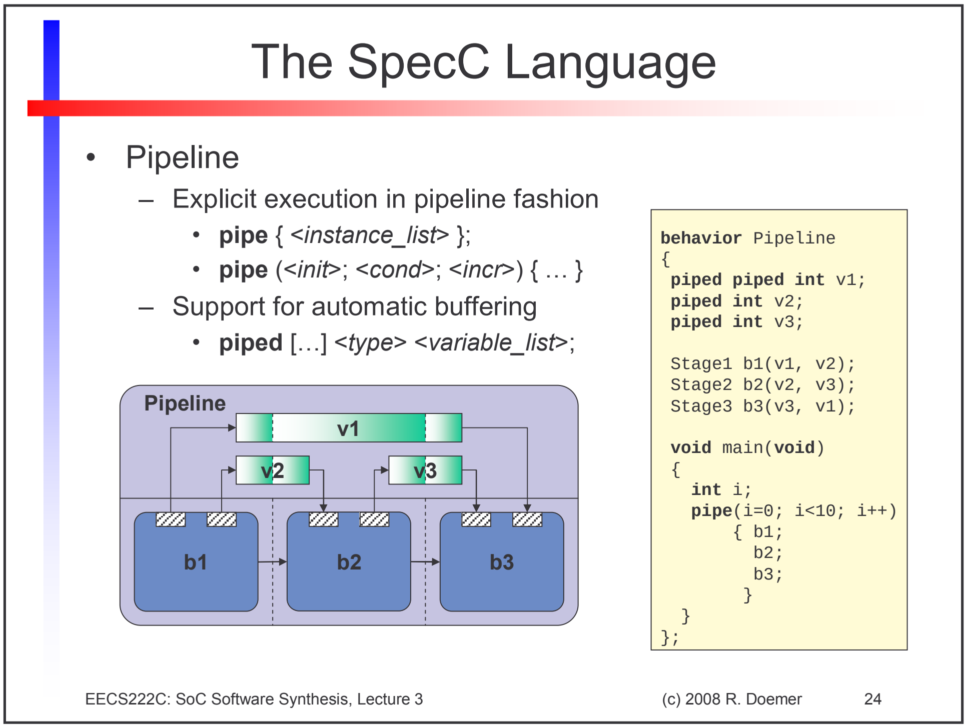
The SpecC Language

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

Pipeline
v1
v2
v3
b1
b2
b3

```
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;
         b2;
         b3;
        }
  }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 23



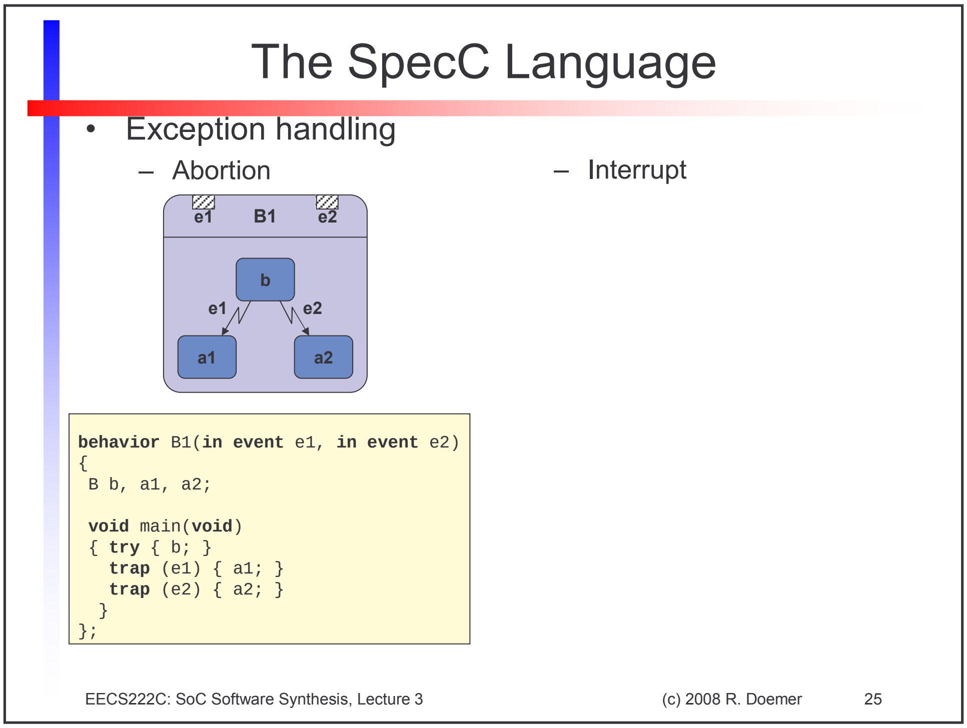
The SpecC Language

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

Pipeline
v1
v2
v3
b1
b2
b3

```
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;
         b2;
         b3;
        }
  }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 24

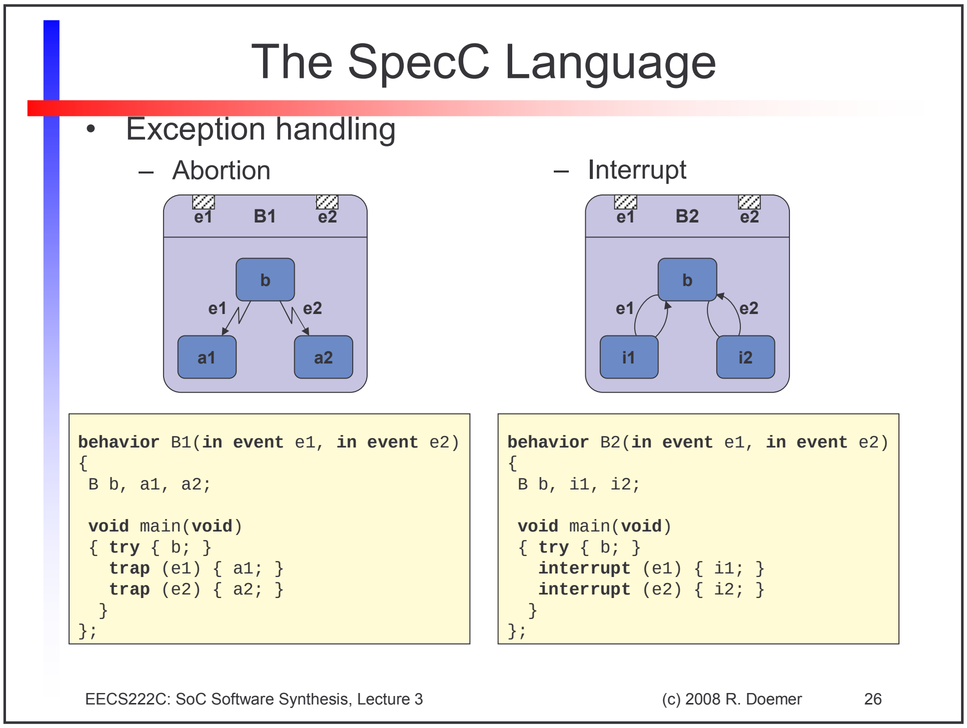


The SpecC Language

- 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; }
   trap (e1) { a1; }
   trap (e2) { a2; }
  }
};
```



The SpecC Language

- Exception handling
  - Abortion
  - Interrupt

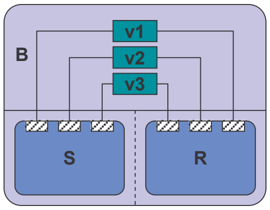
e1
B1
e2
b
e1
e2
a1
a2
e1
B2
e2
b
e1
e2
i1
i2

```
behavior B1(in event e1, in event e2)
{
 B b, a1, a2;

 void main(void)
 { try { b; }
   trap (e1) { a1; }
   trap (e2) { a2; }
  }
};
```

```
behavior B2(in event e1, in event e2)
{
 B b, i1, i2;

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

The SpecC Language

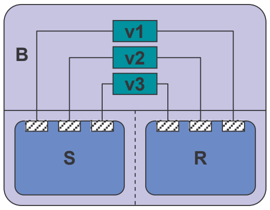
- Communication
  - via shared variable



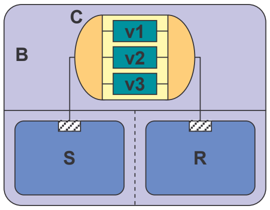
Shared memory

The SpecC Language

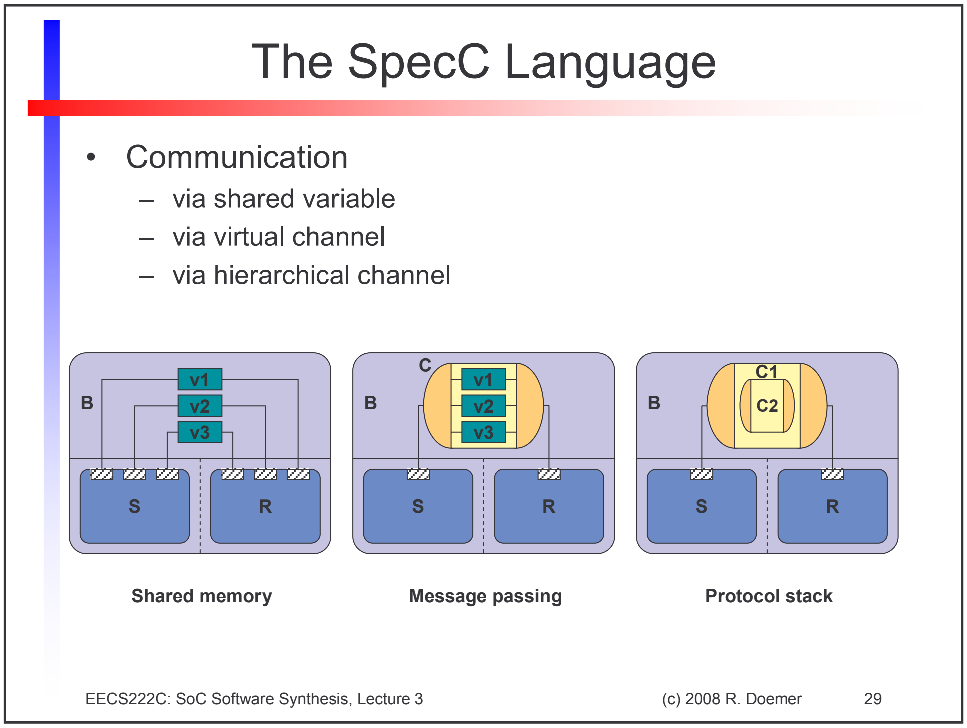
- Communication
  - via shared variable
  - via virtual channel



Shared memory



Message passing



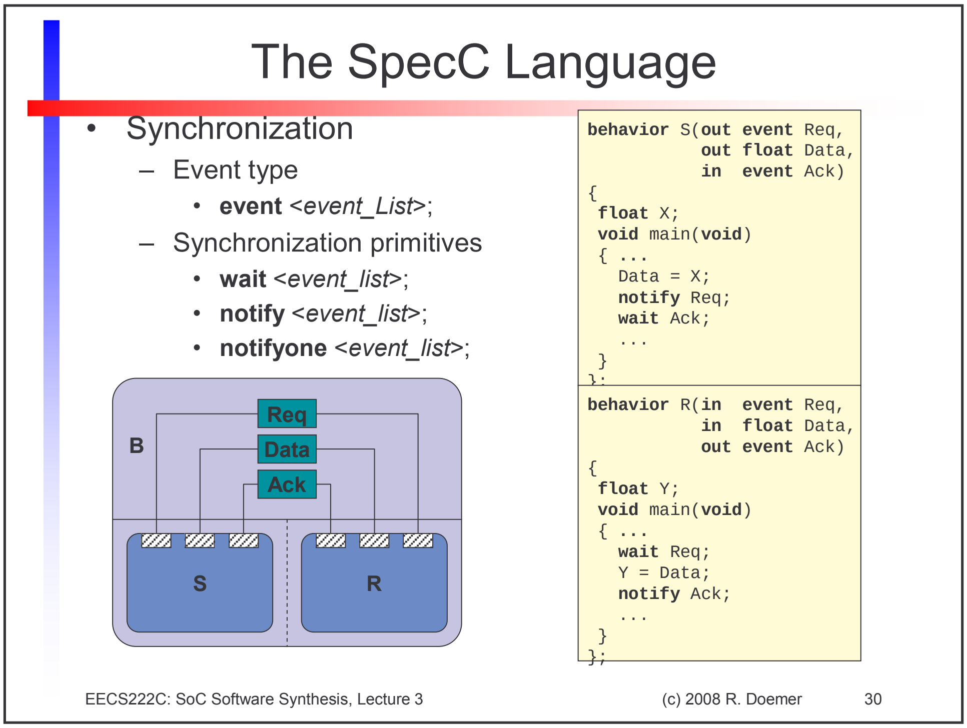
The SpecC Language

- Communication
  - via shared variable
  - via virtual channel
  - via hierarchical channel

Shared memory

Message passing

Protocol stack

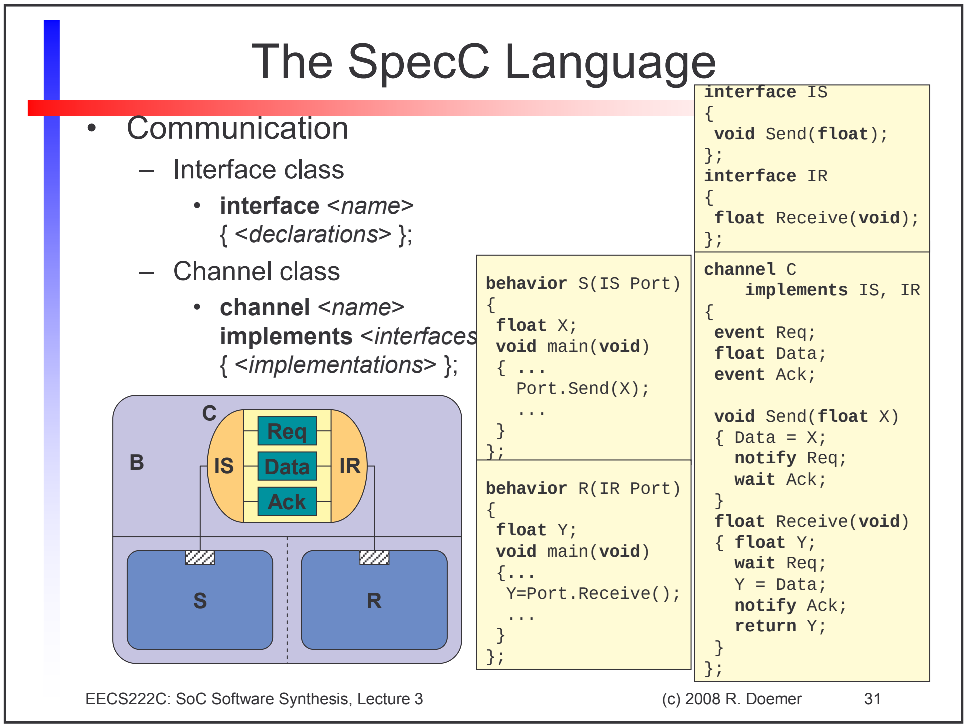


The SpecC Language

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

```
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

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

C
B
IS
Req
Data
Ack
IR
S
R

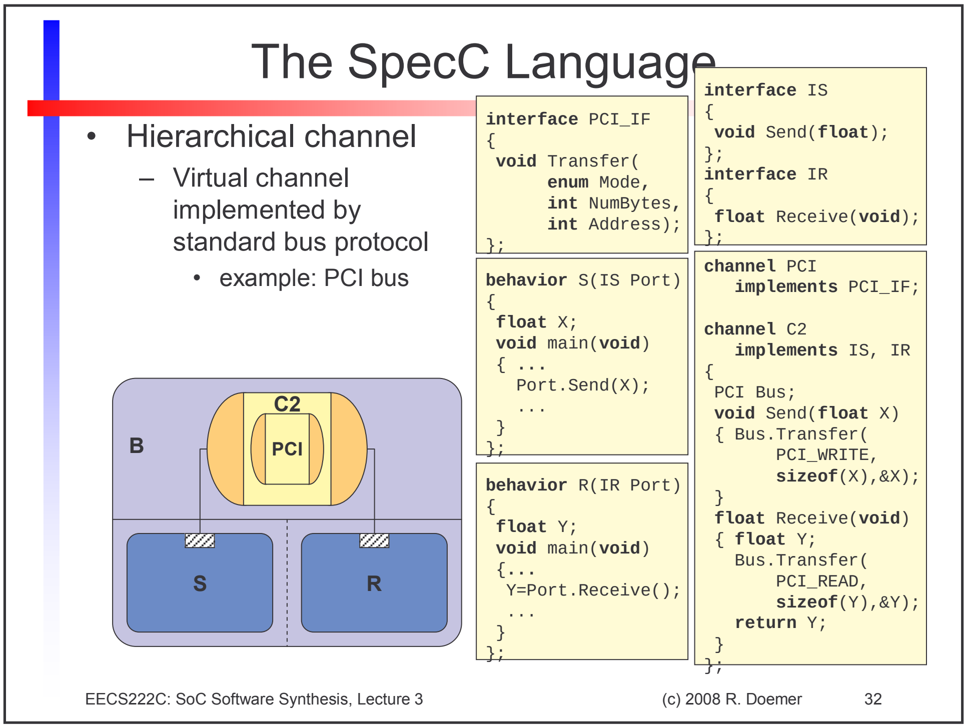
```
behavior S(IS Port)
{
 float X;
 void main(void)
 { ...
   Port.Send(X);
   ...
 }
};
```

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

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

```
channel 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;
 }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 31



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);
};
```

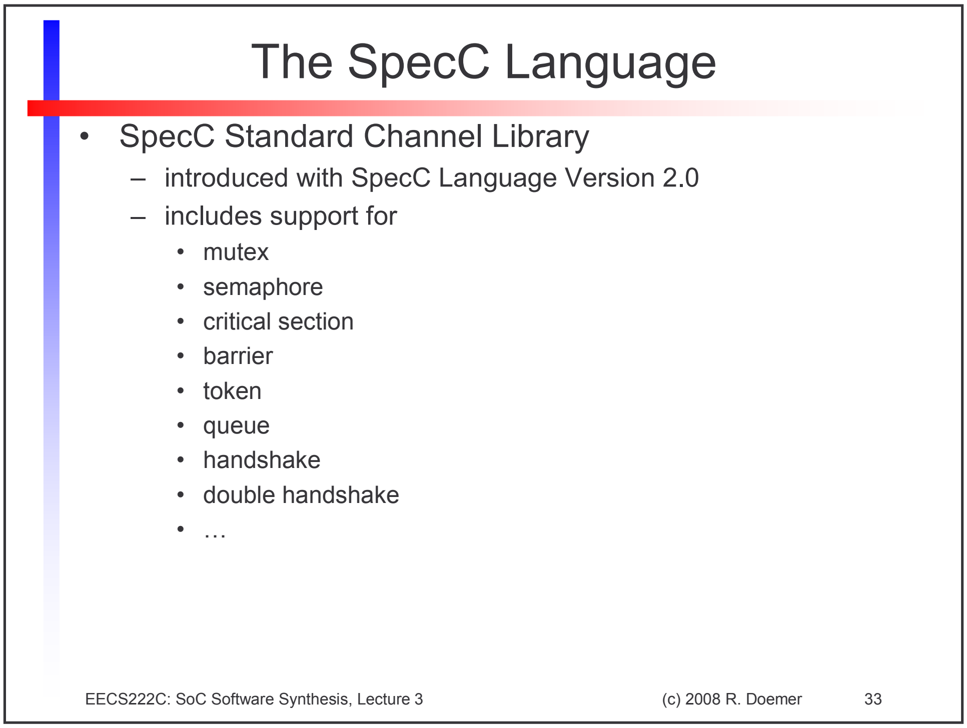
```
behavior S(IS Port)
{
 float X;
 void main(void)
 { ...
   Port.Send(X);
   ...
 }
};
```

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

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

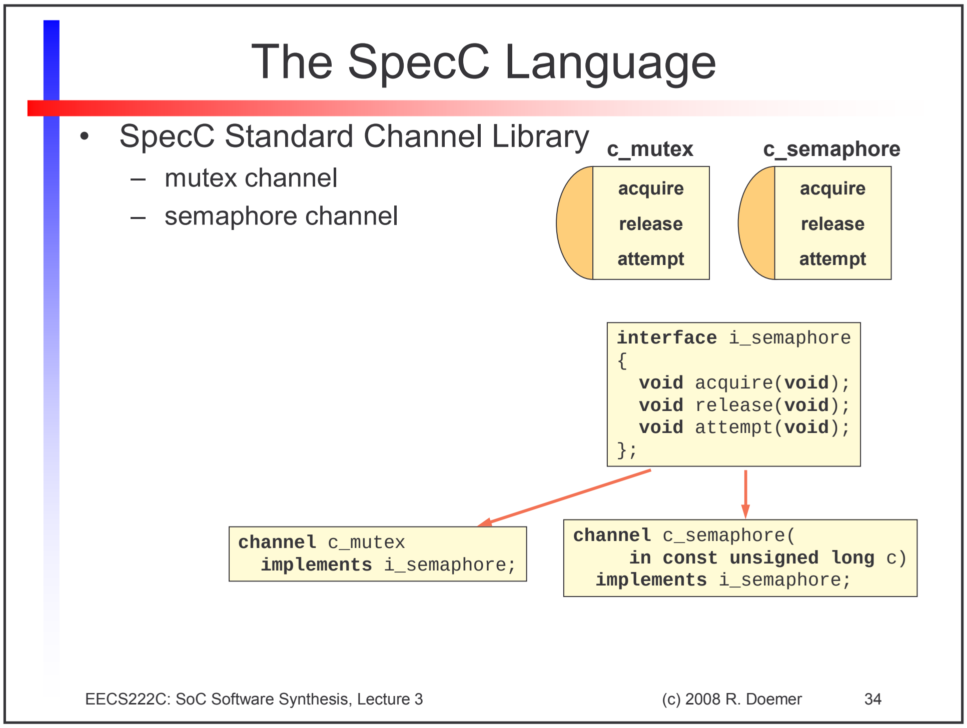
```
channel PCI
   implements PCI_IF;

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;
 }
};
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 32



The SpecC Language

- SpecC Standard Channel Library
  - introduced with SpecC Language Version 2.0
  - includes support for
    - mutex
    - semaphore
    - critical section
    - barrier
    - token
    - queue
    - handshake
    - double handshake
    - …



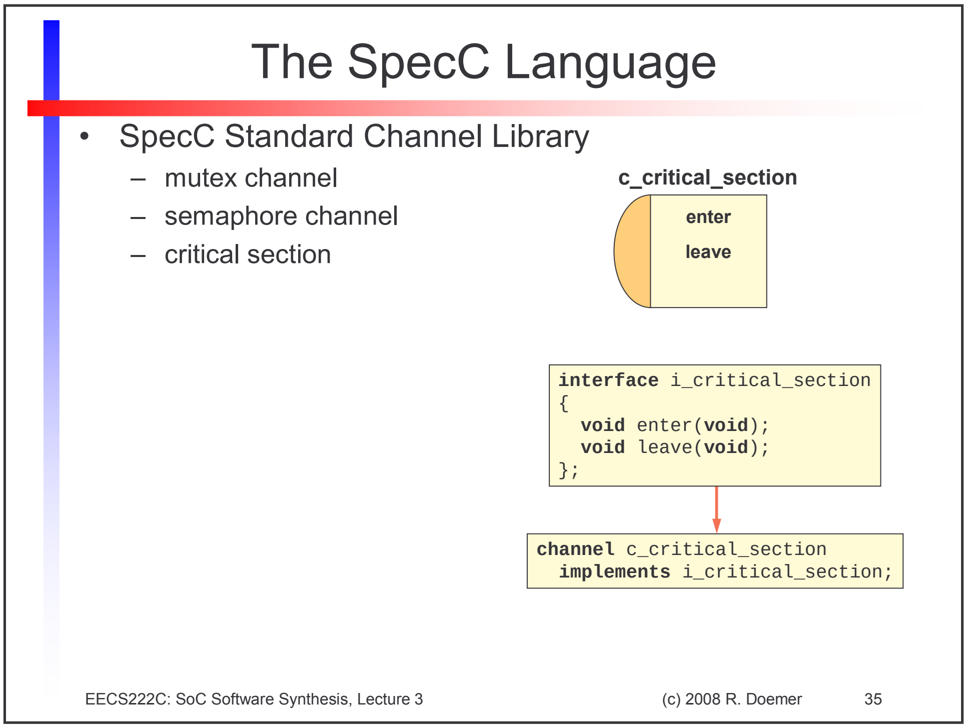
The SpecC Language

- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel

```
interface i_semaphore
{
  void acquire(void);
  void release(void);
  void attempt(void);
};
```

```
channel c_mutex
  implements i_semaphore;
```

```
channel c_semaphore(
     in const unsigned long c)
  implements i_semaphore;
```



The SpecC Language

- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section

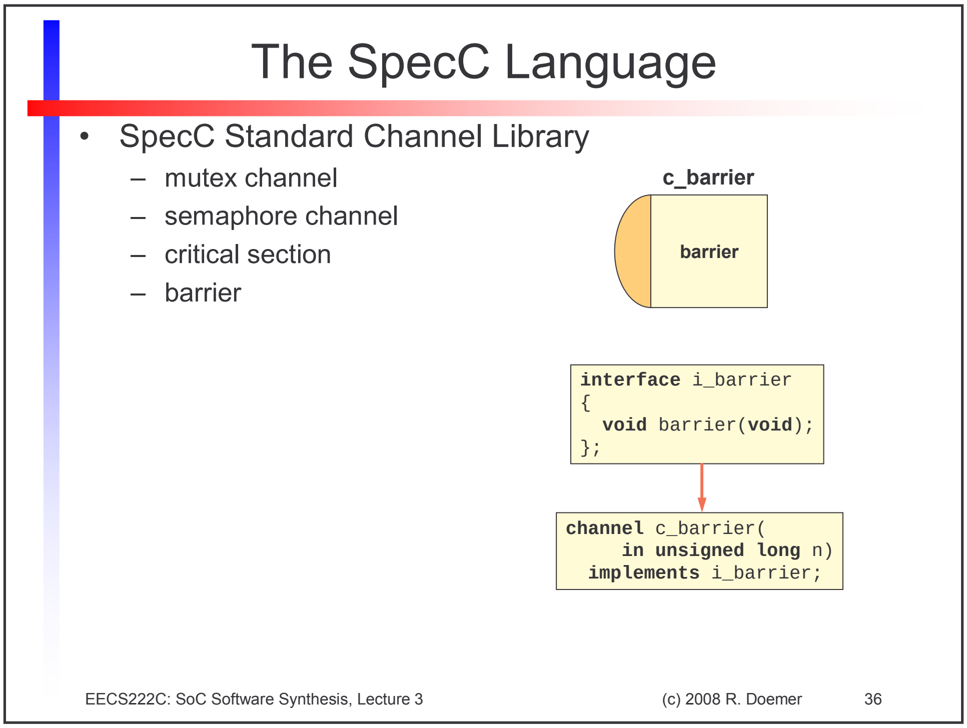
c_critical_section

enter

leave

```
interface i_critical_section
{
  void enter(void);
  void leave(void);
};
```

```
channel c_critical_section
  implements i_critical_section;
```



The SpecC Language

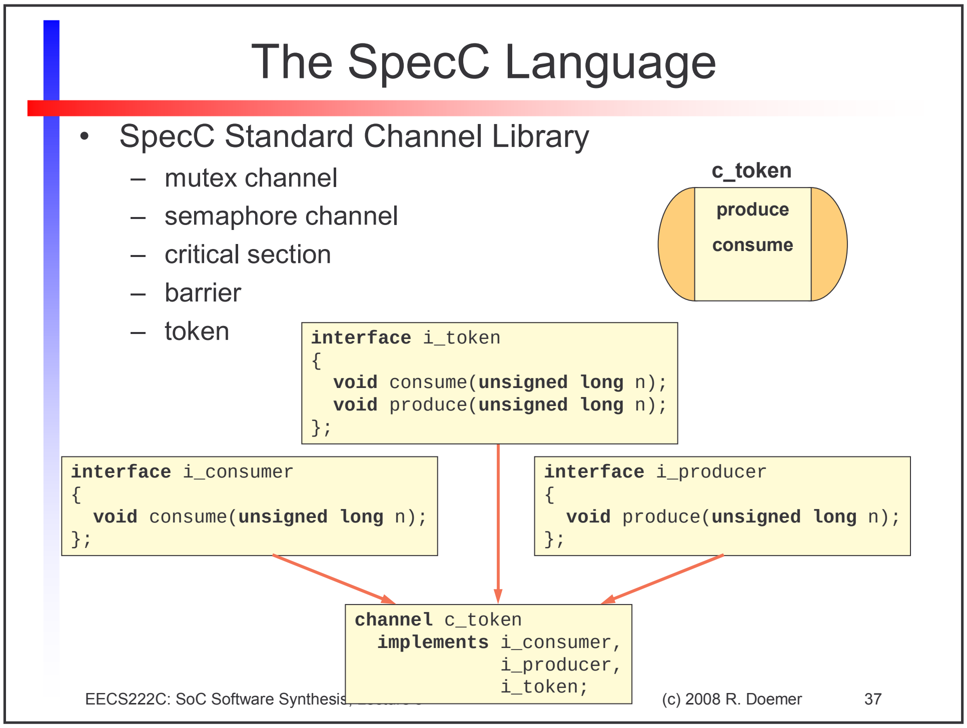
- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier

c_barrier

barrier

```
interface i_barrier
{
  void barrier(void);
};
```

```
channel c_barrier(
     in unsigned long n)
  implements i_barrier;
```



The SpecC Language

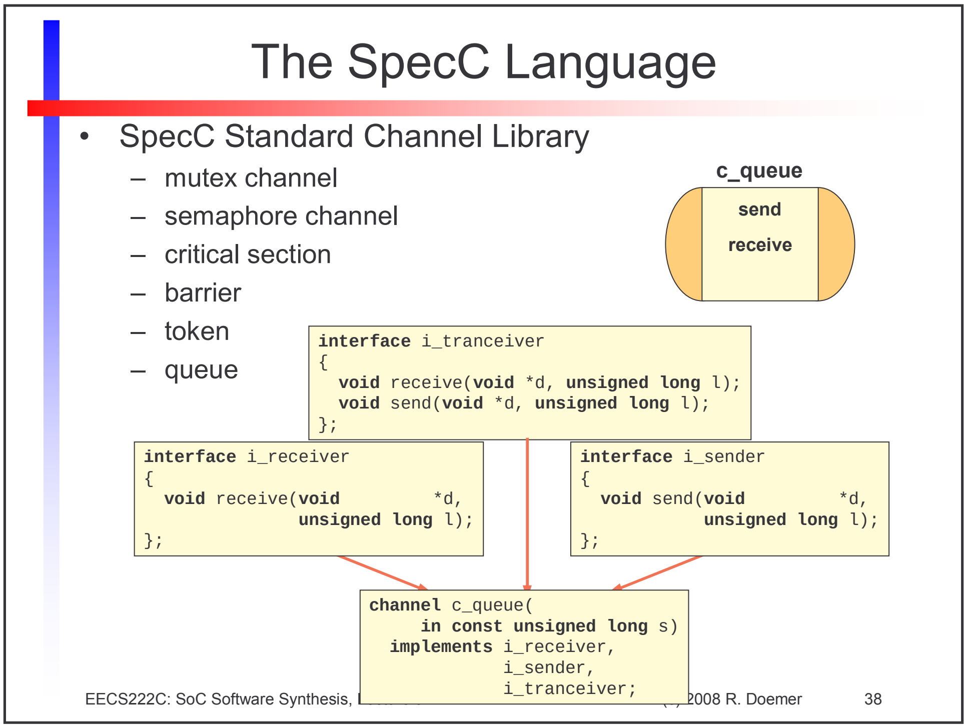
- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier
  - token

```
interface i_token
{
  void consume(unsigned long n);
  void produce(unsigned long n);
};
```

```
interface i_consumer
{
  void consume(unsigned long n);
};
```

```
interface i_producer
{
  void produce(unsigned long n);
};
```

```
channel c_token
  implements i_consumer,
             i_producer,
             i_token;
```



The SpecC Language

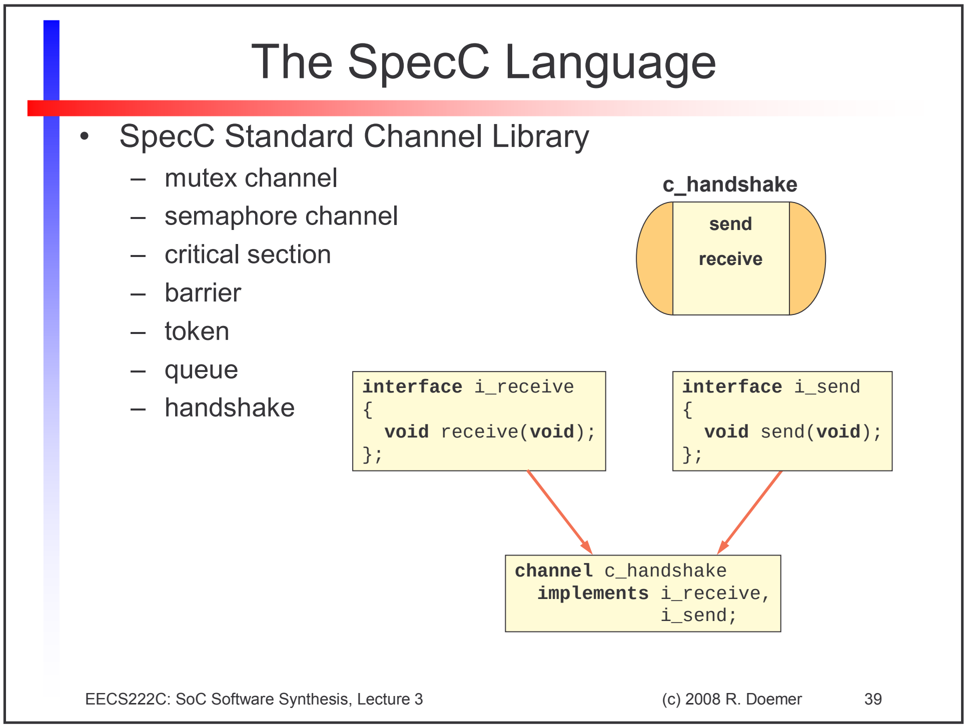
- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier
  - token
  - queue

```
interface i_tranceiver
{
  void receive(void *d, unsigned long l);
  void send(void *d, unsigned long l);
};
```

```
interface i_receiver
{
  void receive(void          *d,
               unsigned long l);
};
```

```
interface i_sender
{
  void send(void          *d,
            unsigned long l);
};
```

```
channel c_queue(
      in const unsigned long s)
  implements i_receiver,
             i_sender,
             i_tranceiver;
```



The SpecC Language

- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier
  - token
  - queue
  - handshake

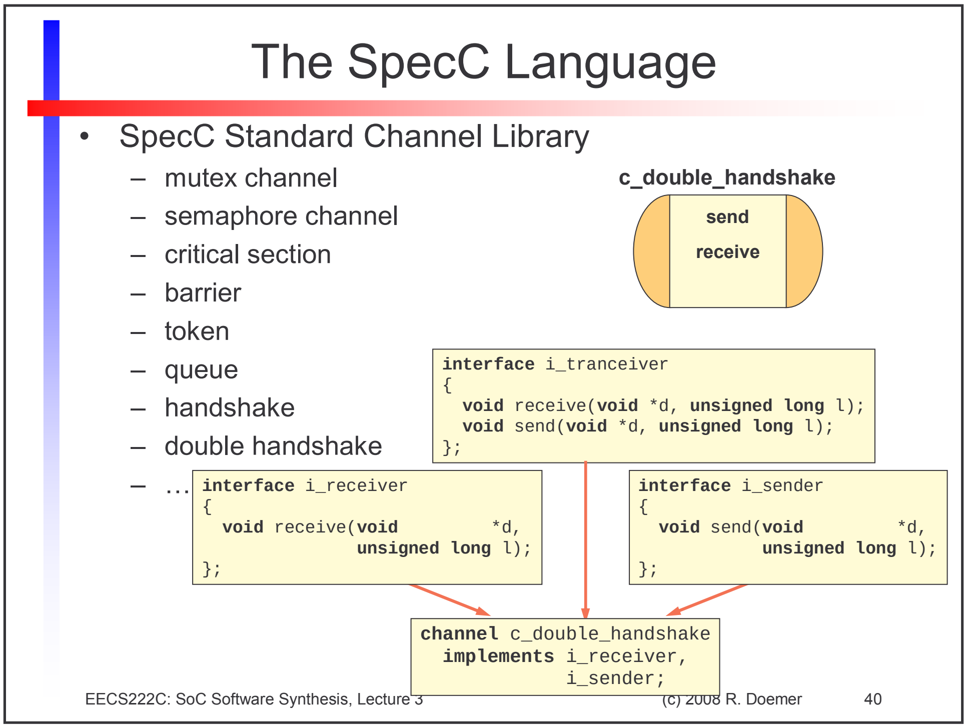
c_handshake
send
receive

```
interface i_receive
{
  void receive(void);
};
```

```
interface i_send
{
  void send(void);
};
```

```
channel c_handshake
  implements i_receive,
             i_send;
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 39



The SpecC Language

- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier
  - token
  - queue
  - handshake
  - double handshake
  - …

c_double_handshake
send
receive

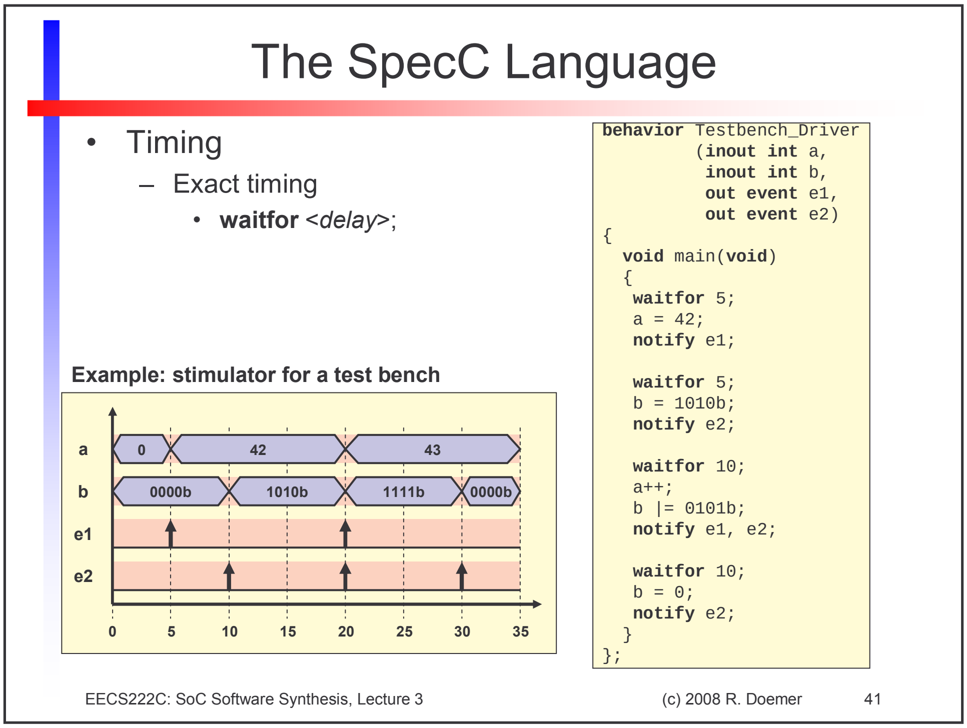
```
interface i_tranceiver
{
  void receive(void *d, unsigned long l);
  void send(void *d, unsigned long l);
};
```

```
interface i_receiver
{
  void receive(void          *d,
               unsigned long l);
};
```

```
interface i_sender
{
  void send(void          *d,
            unsigned long l);
};
```

```
channel c_double_handshake
  implements i_receiver,
             i_sender;
```

EECS222C: SoC Software Synthesis, Lecture 3 (c) 2008 R. Doemer 40



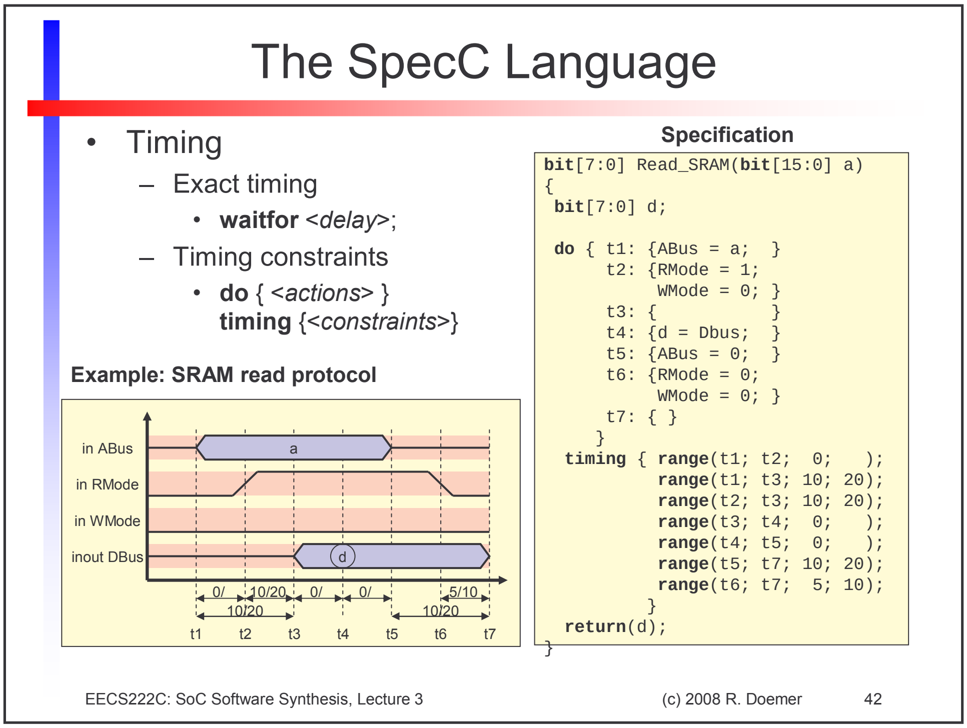
The SpecC Language

- 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;
  }
};
```

EECS222C: SoC Software Synthesis, Lecture 3   (c) 2008 R. Doemer   41



The SpecC Language

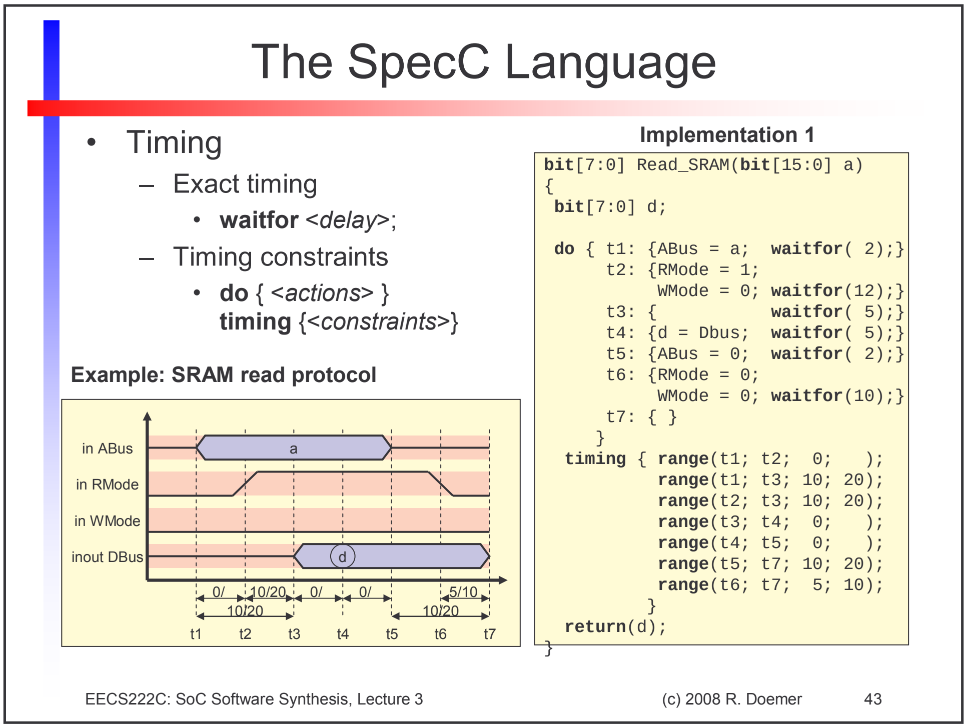
- Timing
  - Exact timing
    - **waitfor** <*delay*>;
  - Timing constraints
    - **do** { <*actions*> }
      timing {<*constraints*>}

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);
}
```

EECS222C: SoC Software Synthesis, Lecture 3   (c) 2008 R. Doemer   42



The SpecC Language

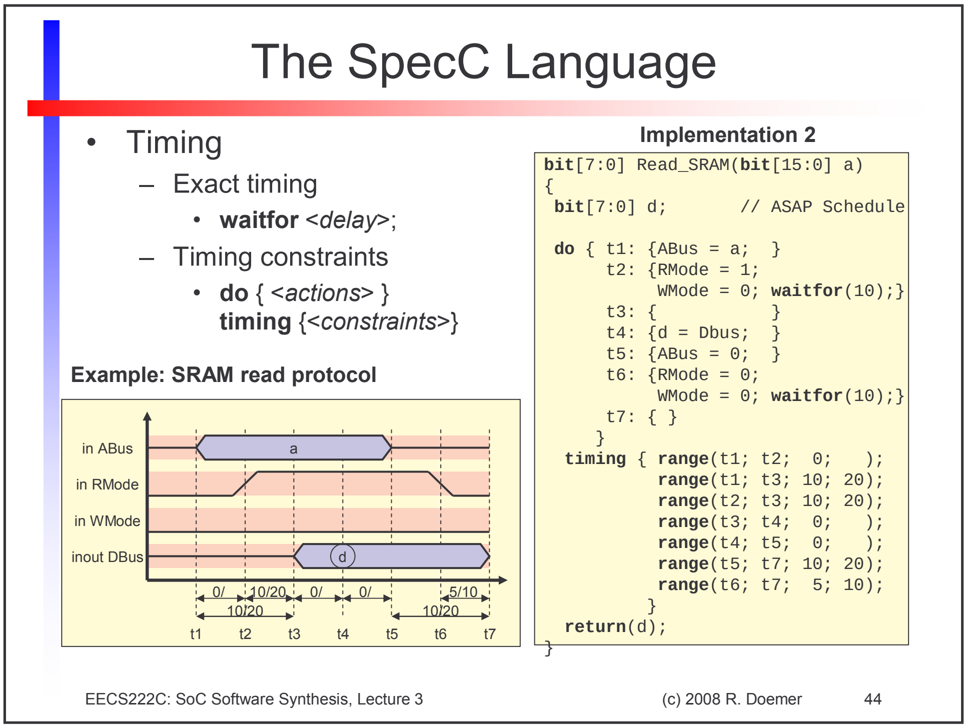
- Timing
  - Exact timing
    - **waitfor** <*delay*>;
  - Timing constraints
    - **do** { <*actions*> }
      timing {<*constraints*>}

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 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);
}
```

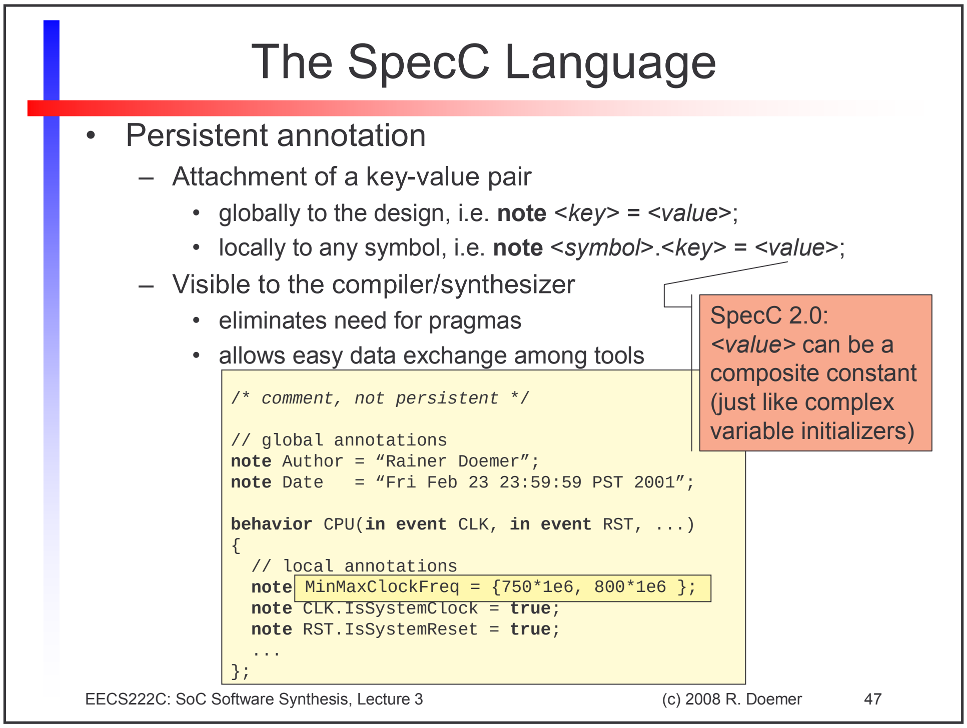
The SpecC Language

- 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

- 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

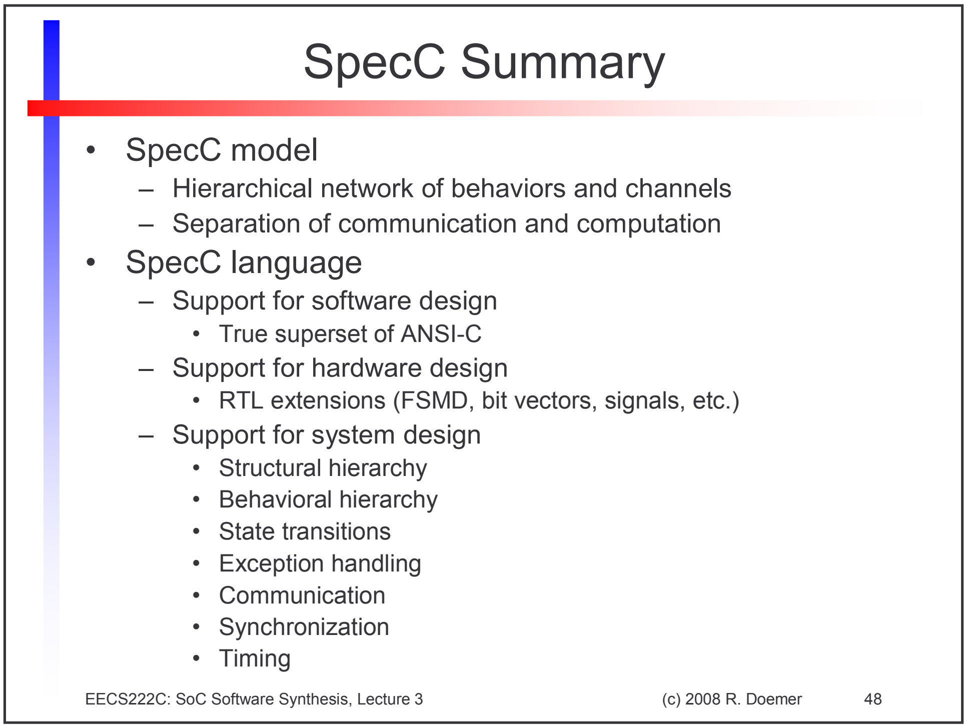


The SpecC Language

- 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

SpecC 2.0:
<*value*> can be a composite constant (just like complex variable initializers)

```
/* 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 MinMaxClockFreq = {750*1e6, 800*1e6 };
  note CLK.IsSystemClock = true;
  note RST.IsSystemReset = true;
  ...
};
```



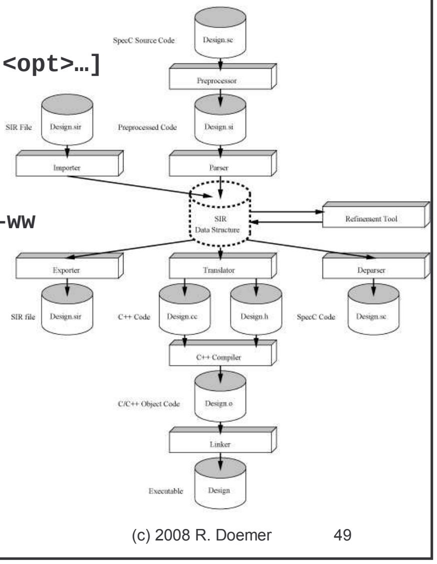
SpecC Summary

- SpecC model
  - Hierarchical network of behaviors and channels
  - Separation of communication and computation
- SpecC language
  - Support for software design
    - True superset of ANSI-C
  - Support for hardware design
    - RTL extensions (FSMD, bit vectors, signals, etc.)
  - Support for system design
    - Structural hierarchy
    - Behavioral hierarchy
    - State transitions
    - Exception handling
    - Communication
    - Synchronization
    - Timing

The SpecC Compiler and Simulator

- SpecC Compiler
  - Command line interface
  - Usage: `scc <design> [<cmd>] [<opt>…]`
  - Help: `scc -h`
    `man scc`
  - Example:

```
% scc HelloWorld –sc2out –v –ww
scc: SpecC Compiler V 2.2.1
(c)2008 CECS, UC Irvine
Preprocessing...
Parsing...
Translating...
Compiling...
Linking…
Done.
```



The SpecC Compiler and Simulator

- SpecC Simulator
  - Execution as regular program
  - Example:

```
% ./HelloWorld
Hello World!
```

  - Simulation library
    - Access via inclusion of SpecC header files
    - Example: Print the current simulation time

```
– #include <sim.sh>
– ...
– sim_time t;
– sim_delta d;
– sim_time_string buffer;
– ...
– t = now();  d = delta();
– printf(“Time is now %s pico seconds.\n”, time2str(buffer, t));
– printf(“(delta count is %s)\n”, time2str(buffer, d);
– waitfor 10 NANO_SEC;
– printf(“Time is now %s pico seconds.\n”, time2str(buffer, t));
– ...
```

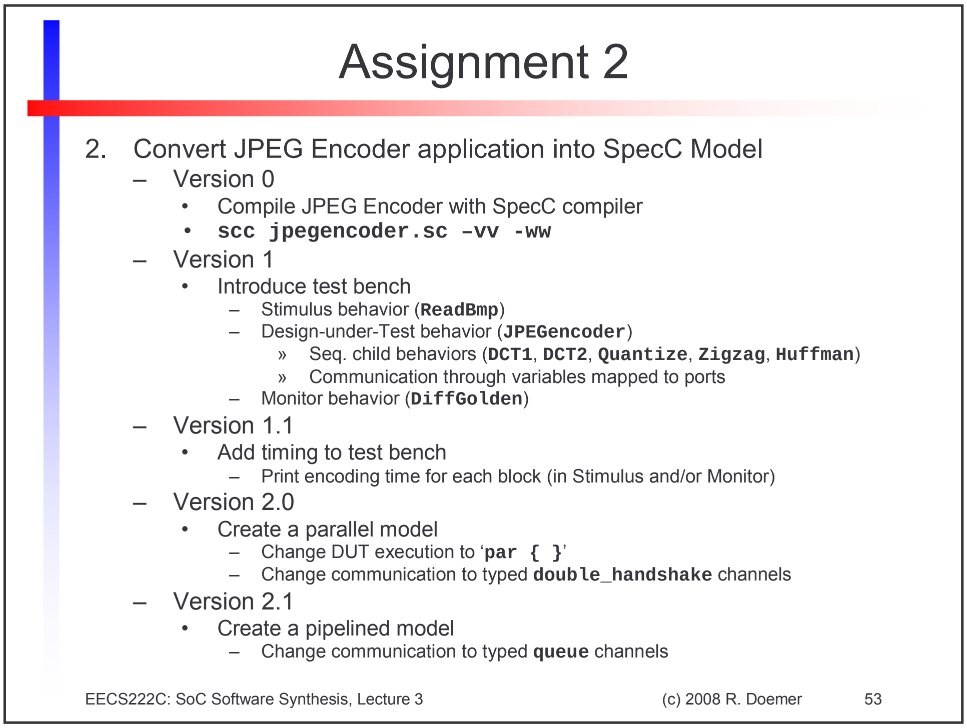
The SpecC Compiler and Simulator

- SpecC Command Line Tools
  - Tools working with SpecC Internal Representation (SIR) files
  - Example:

```
  % scc Adder -sc2sir -o Adder.sir
– % sir_list -t Adder.sir
– behavior ADD8
– behavior AND2
– behavior FA
– behavior HA
– behavior Main
– behavior XOR2
– % sir_tree -bt Adder.sir FA
– behavior FA
– |------ HA ha1
– |       |------ AND2 and1
– |       \------ XOR2 xor1
– |------ HA ha2
– |       |------ AND2 and1
– |       \------ XOR2 xor1
– \------ OR2 or1
```

Assignment 2

1. Practice SpecC Tools
   - Setup
     - `source /opt/sce-20080601/bin/setup.csh`
   - Examine simple examples
     - `mkdir simple_tests`
     - `cd simple_tests`
     - `cp $SPECC/examples/simple/* .`
     - `ls`
     - `vi HelloWorld.sc`
   - Practice the compiler
     - `man scc`
     - `scc HelloWorld –sc2out –vv -ww`
   - Practice the simulator
     - `./HelloWorld`
   - Practice the tools
     - `man sir_tree`
     - `scc Adder -sc2sir -o Adder.sir`
     - `sir_tree -bt Adder.sir FA`



Assignment 2

2. Convert JPEG Encoder application into SpecC Model
   - Version 0
     - Compile JPEG Encoder with SpecC compiler
     - **`scc jpegencoder.sc –vv -ww`**
   - Version 1
     - Introduce test bench
       - Stimulus behavior (**`ReadBmp`**)
       - Design-under-Test behavior (**`JPEGencoder`**)
         - » Seq. child behaviors (**`DCT1`**, **`DCT2`**, **`Quantize`**, **`Zigzag`**, **`Huffman`**)
         - » Communication through variables mapped to ports
       - Monitor behavior (**`DiffGolden`**)
   - Version 1.1
     - Add timing to test bench
       - Print encoding time for each block (in Stimulus and/or Monitor)
   - Version 2.0
     - Create a parallel model
       - Change DUT execution to ‘**`par { }`**’
       - Change communication to typed **`double_handshake`** channels
   - Version 2.1
     - Create a pipelined model
       - Change communication to typed **`queue`** channels