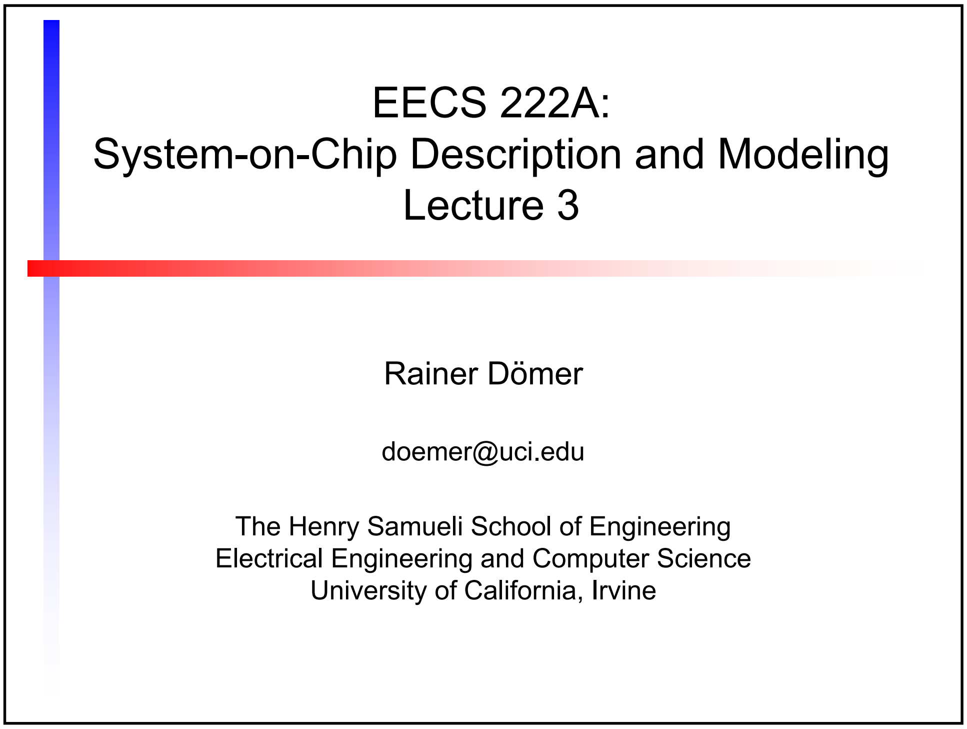
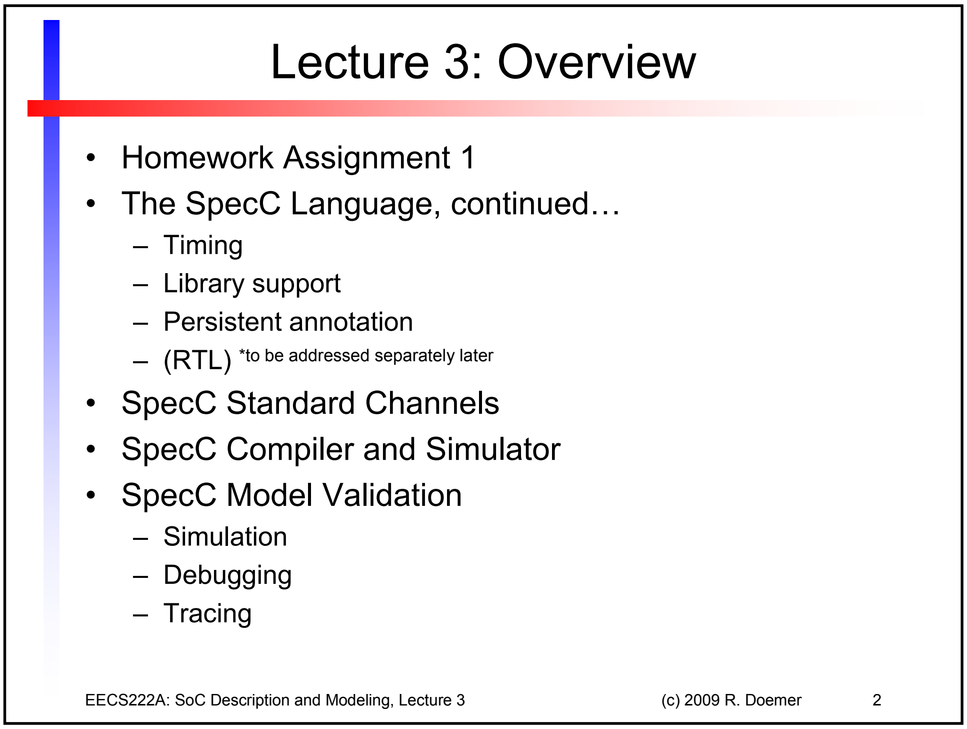


EECS 222A: System-on-Chip Description and Modeling Lecture 3

Rainer Dömer

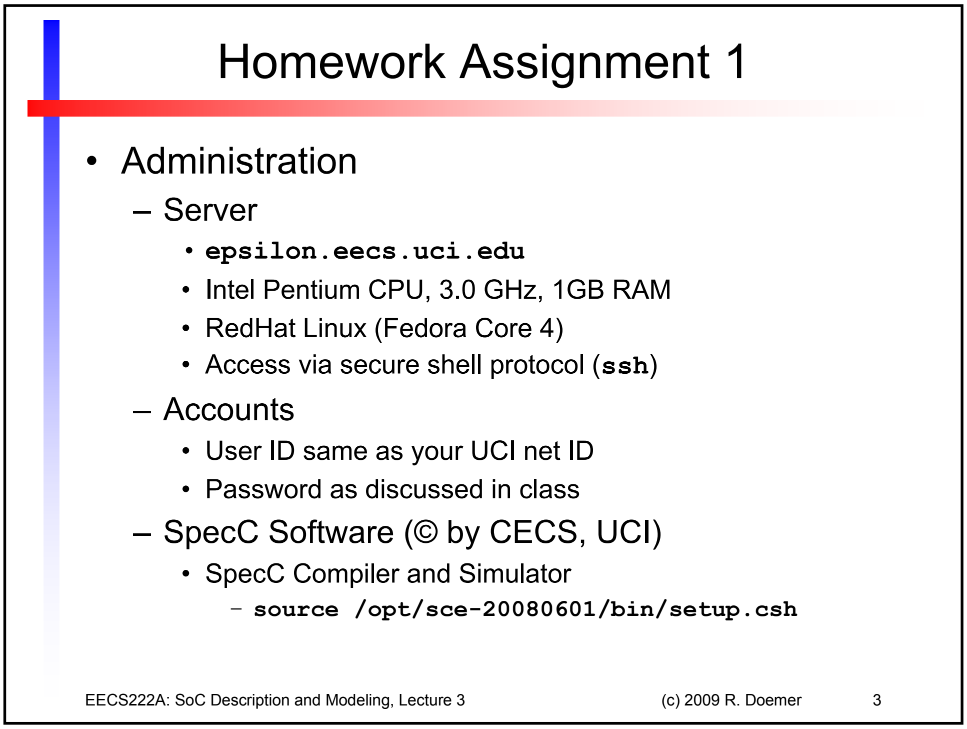
doemer@uci.edu

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



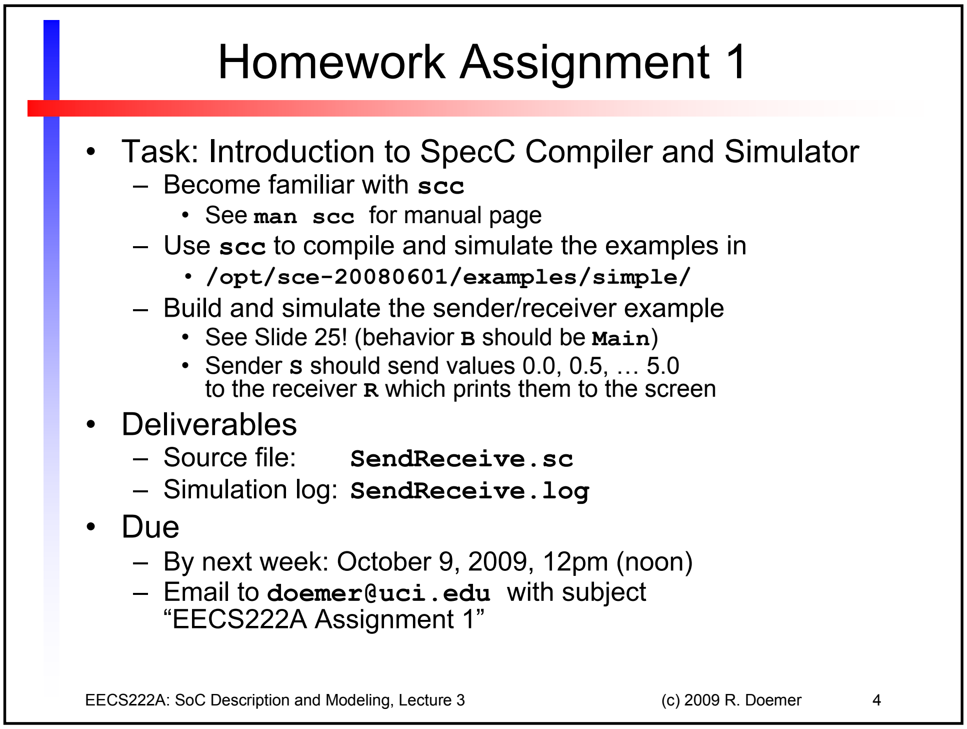
Lecture 3: Overview

- Homework Assignment 1
- The SpecC Language, continued…
  - Timing
  - Library support
  - Persistent annotation
  - (RTL) *to be addressed separately later
- SpecC Standard Channels
- SpecC Compiler and Simulator
- SpecC Model Validation
  - Simulation
  - Debugging
  - Tracing



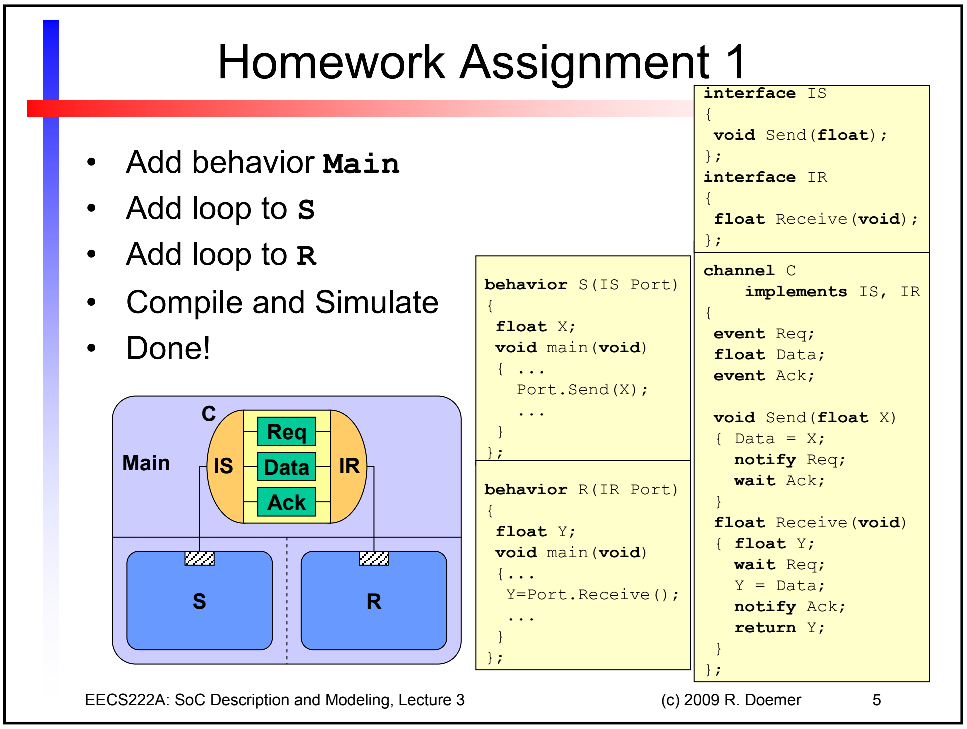
Homework Assignment 1

- Administration
  - Server
    - **`epsilon.eecs.uci.edu`**
    - Intel Pentium CPU, 3.0 GHz, 1GB RAM
    - RedHat Linux (Fedora Core 4)
    - Access via secure shell protocol (**`ssh`**)
  - Accounts
    - User ID same as your UCI net ID
    - Password as discussed in class
  - SpecC Software (© by CECS, UCI)
    - SpecC Compiler and Simulator
      - **`source /opt/sce-20080601/bin/setup.csh`**



Homework Assignment 1

- Task: Introduction to SpecC Compiler and Simulator
  - Become familiar with **`scc`**
    - See **`man scc`** for manual page
  - Use **`scc`** to compile and simulate the examples in
    - **`/opt/sce-20080601/examples/simple/`**
  - Build and simulate the sender/receiver example
    - See Slide 25! (behavior **`B`** should be **`Main`**)
    - Sender **`S`** should send values 0.0, 0.5, … 5.0 to the receiver **`R`** which prints them to the screen
- Deliverables
  - Source file: **`SendReceive.sc`**
  - Simulation log: **`SendReceive.log`**
- Due
  - By next week: October 9, 2009, 12pm (noon)
  - Email to **`doemer@uci.edu`** with subject “EECS222A Assignment 1”



Homework Assignment 1

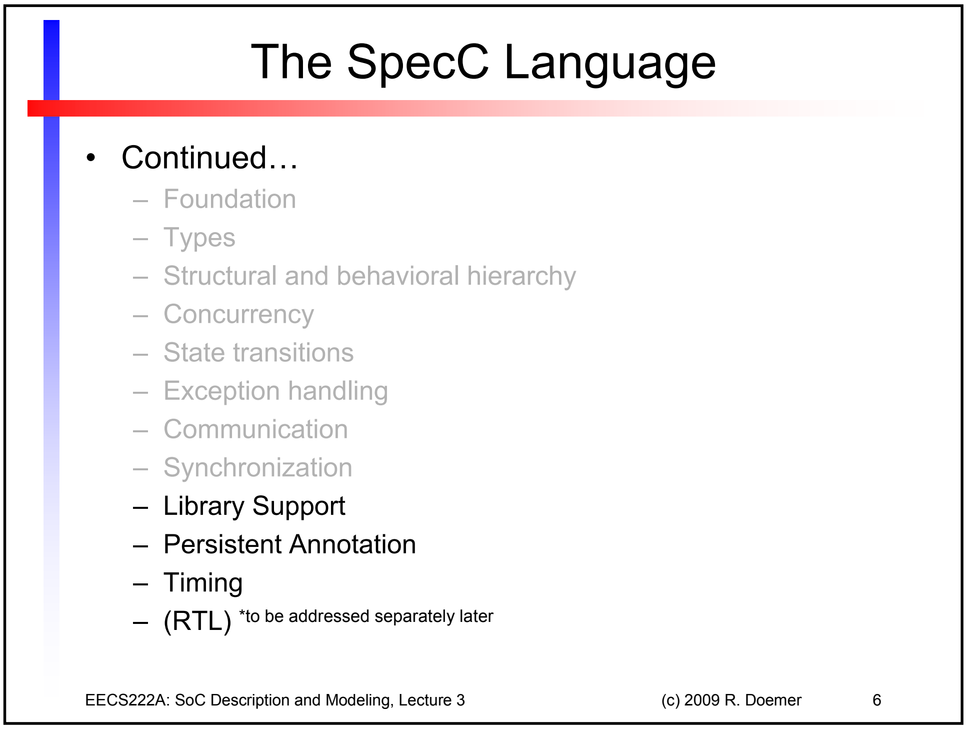
- Add behavior **`Main`**
- Add loop to **`S`**
- Add loop to **`R`**
- Compile and Simulate
- Done!

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

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

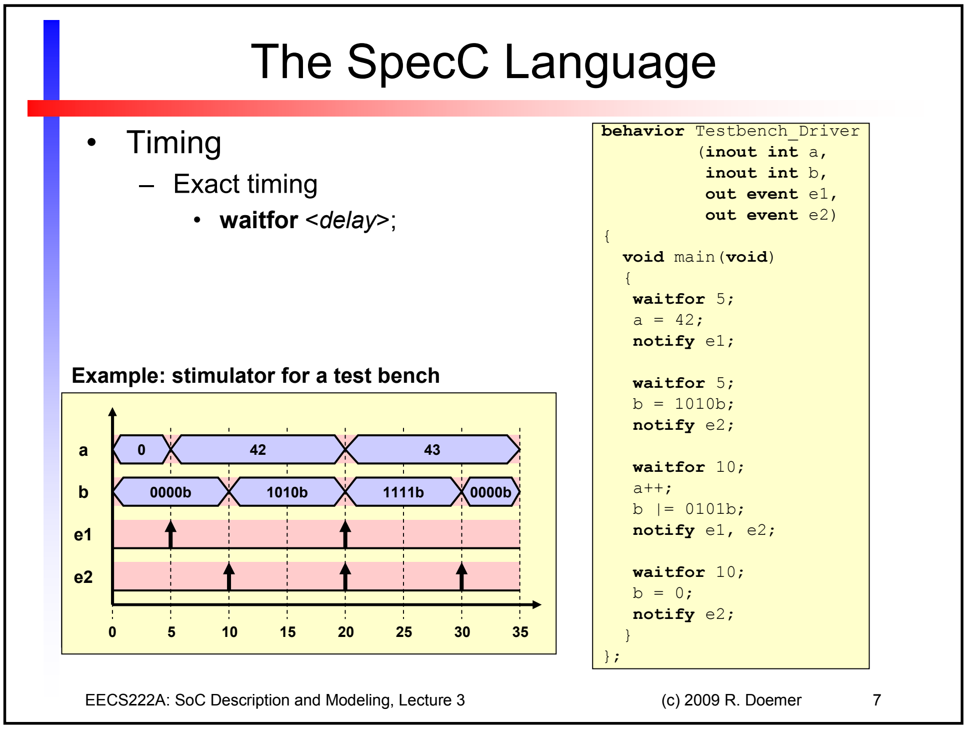
```
interface IS
{
 void Send(float);
};
interface IR
{
 float Receive(void);
};
```

```
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;
 }
};
```



The SpecC Language

- Continued…
  - Foundation
  - Types
  - Structural and behavioral hierarchy
  - Concurrency
  - State transitions
  - Exception handling
  - Communication
  - Synchronization
  - Library Support
  - Persistent Annotation
  - Timing
  - (RTL) *to be addressed separately later



The SpecC Language

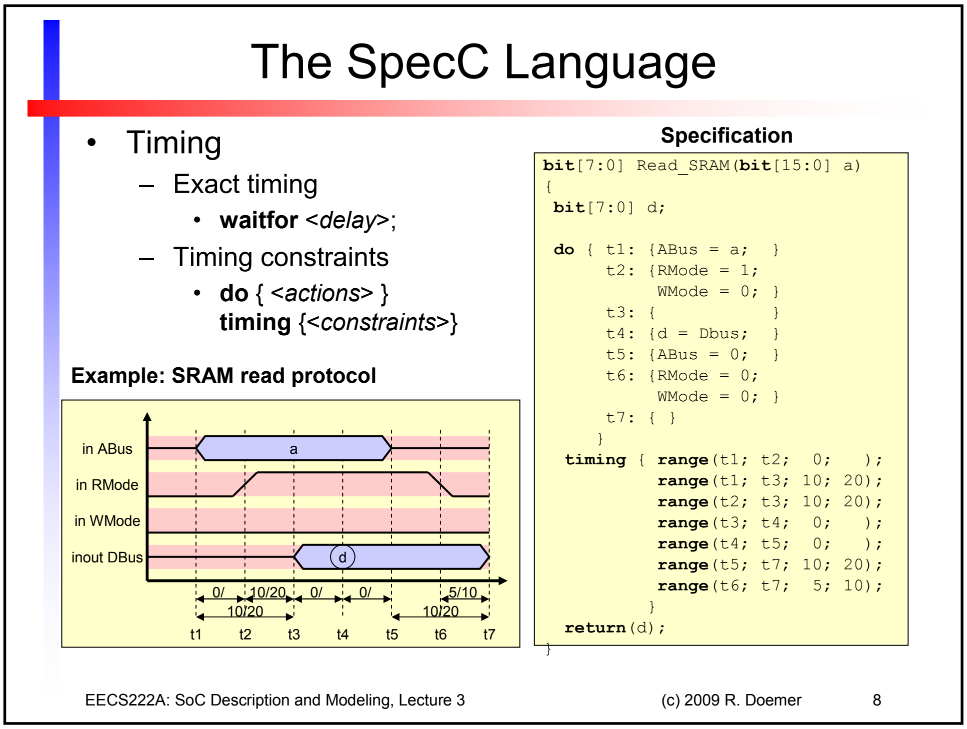
- Timing
  - Exact timing
    - **waitfor** <*delay*>;

Example: stimulator for a test bench

a
0
42
43
b
0000b
1010b
1111b
0000b
e1
e2
0
5
10
15
20
25
30
35

```
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;
   }
};
```

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 7



The SpecC Language

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

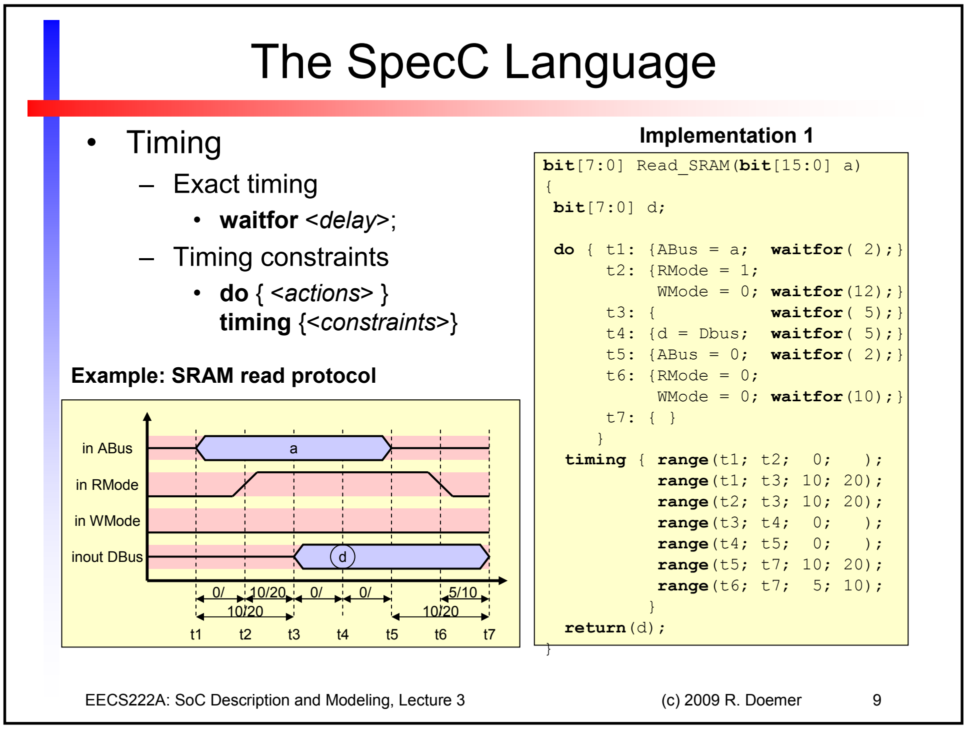
Example: SRAM read protocol

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

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);
}
```

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 8



The SpecC Language

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

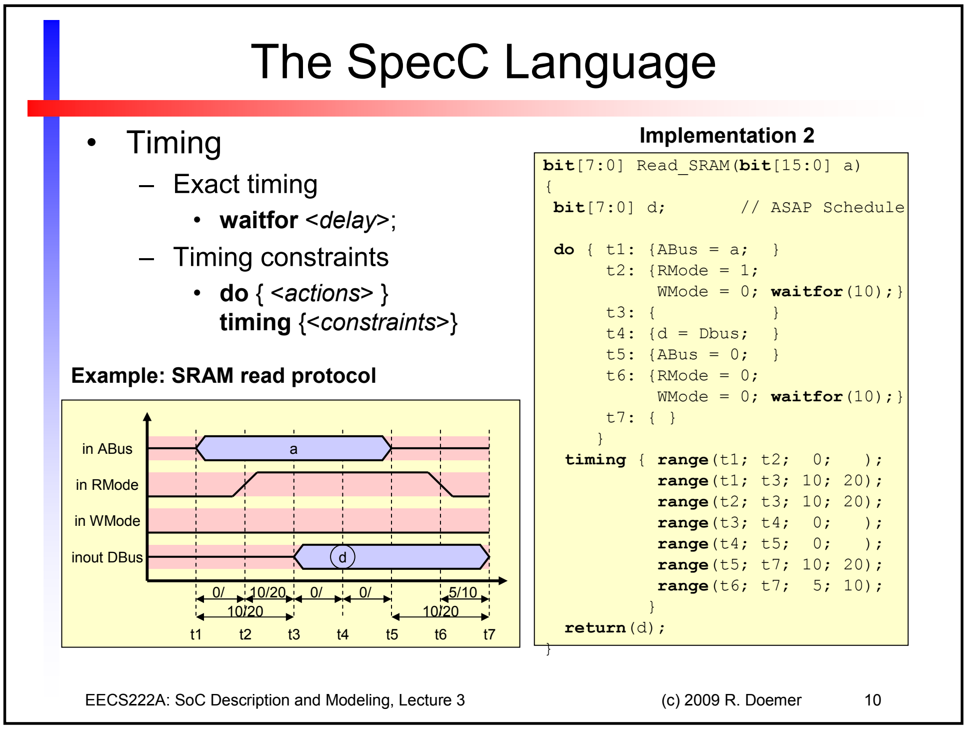
Example: SRAM read protocol

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);
}
```

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 9



The SpecC Language

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

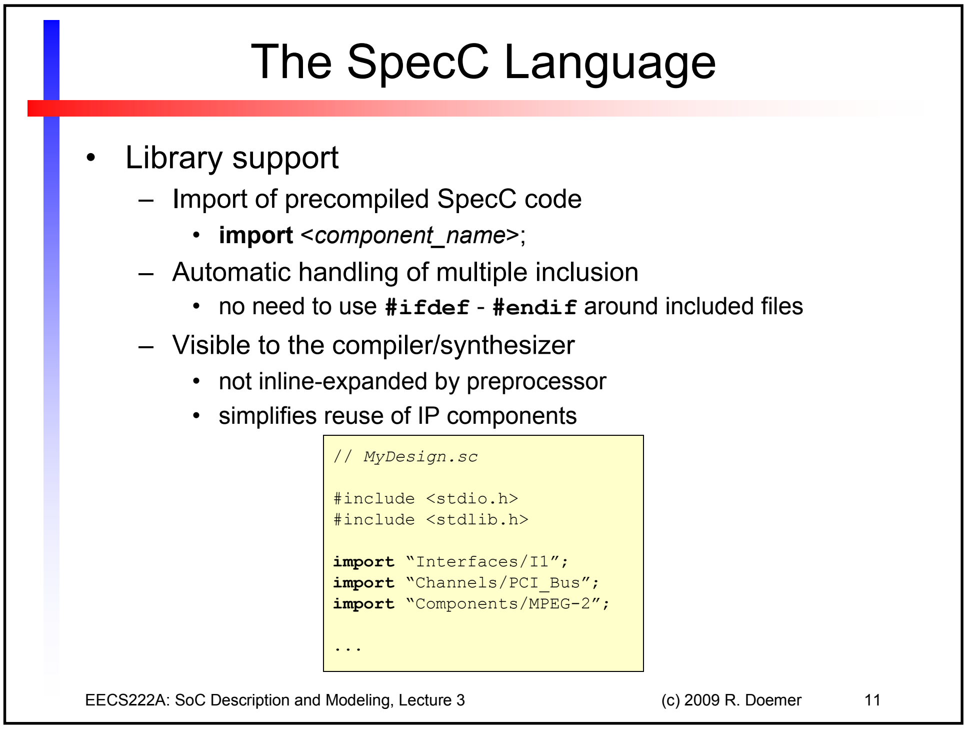
Example: SRAM read protocol

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);
}
```

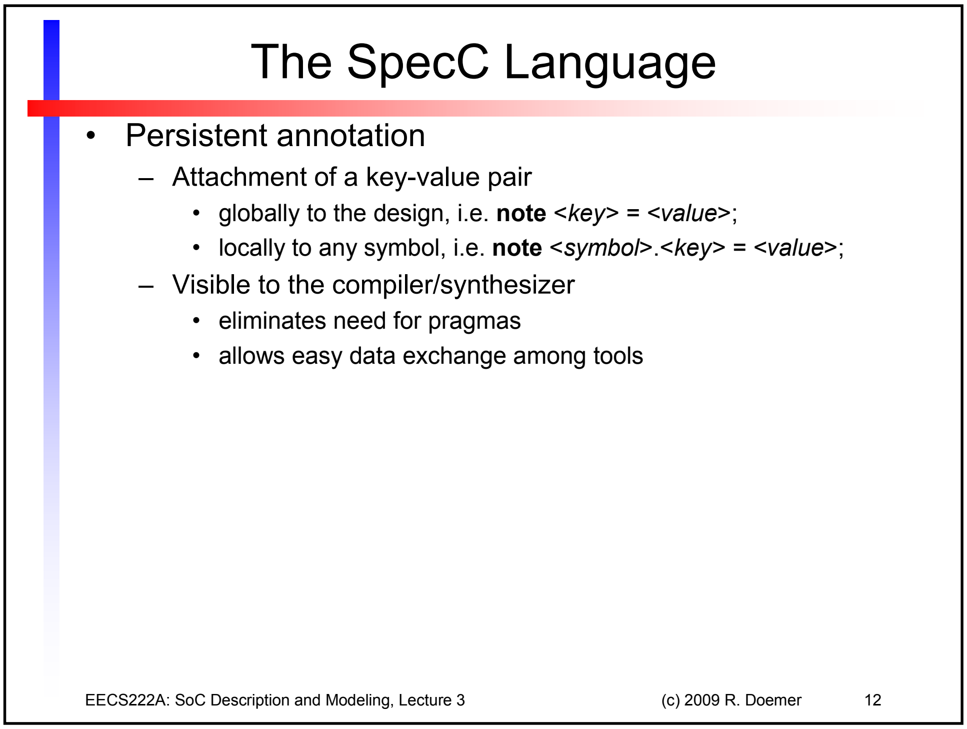
EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 10



The SpecC Language

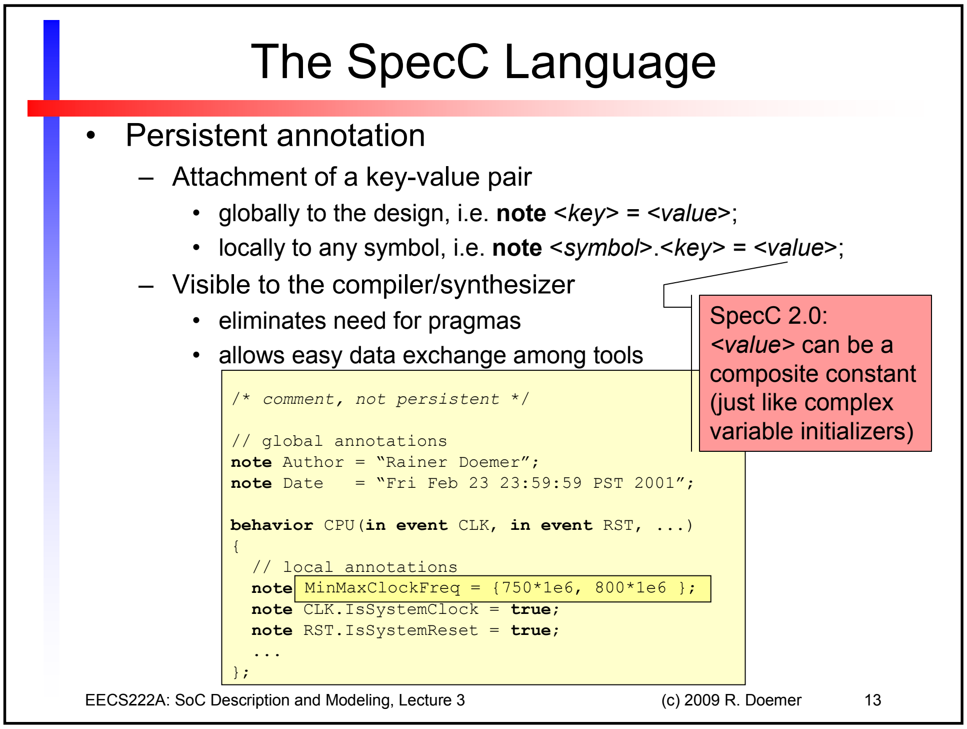
- Library support
  - Import of precompiled SpecC code
    - **import** <*component_name*>;
  - Automatic handling of multiple inclusion
    - no need to use **`#ifdef`** - **`#endif`** around included files
  - Visible to the compiler/synthesizer
    - not inline-expanded by preprocessor
    - simplifies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



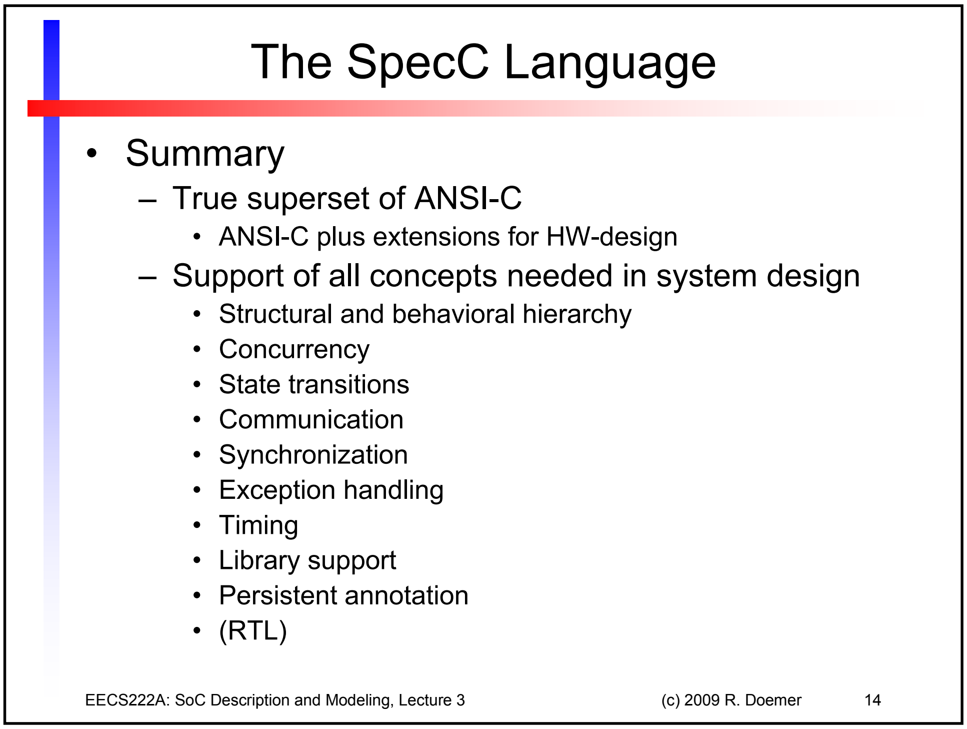
The SpecC Language

- 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

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

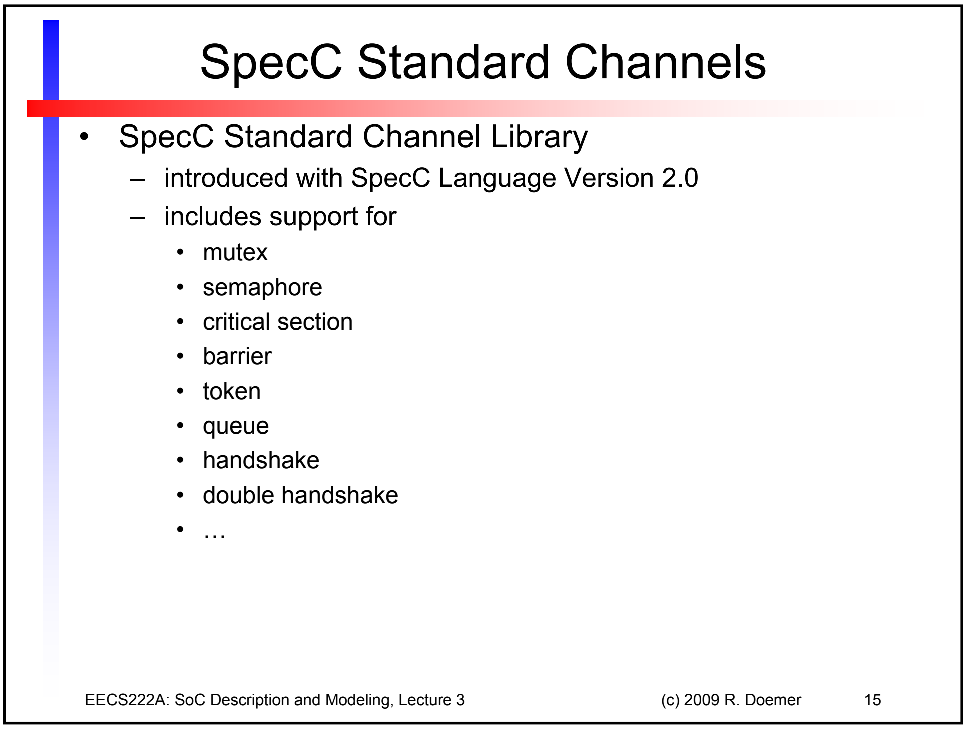
EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 13



The SpecC Language

- Summary
  - 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
    - Library support
    - Persistent annotation
    - (RTL)

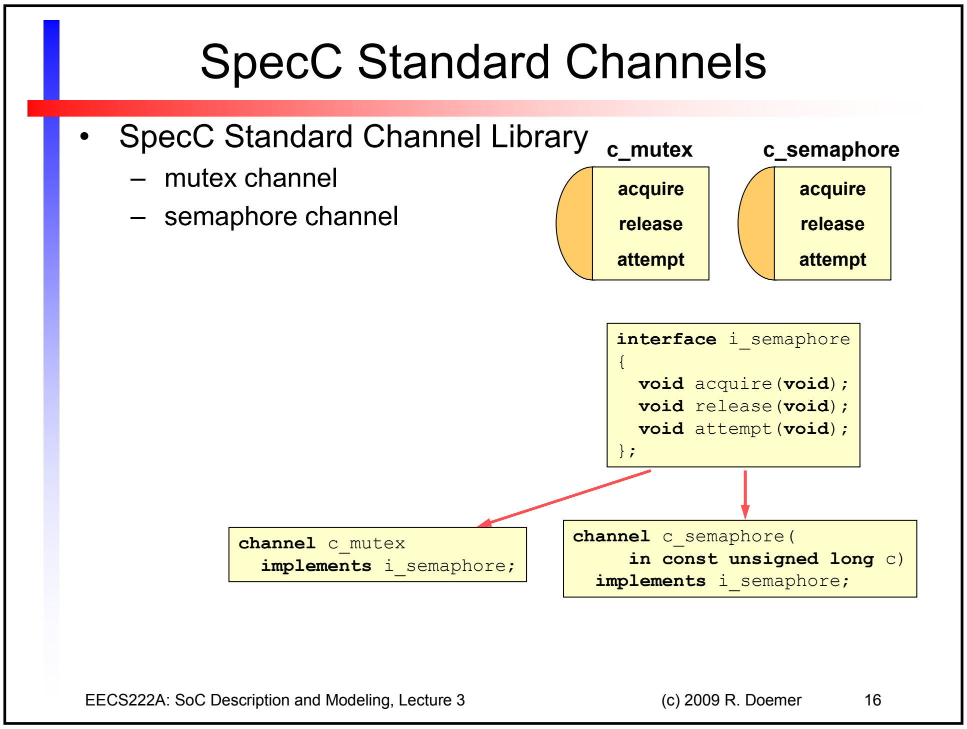
EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 14



SpecC Standard Channels

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

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 15



SpecC Standard Channels

- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel

c_mutex
acquire
release
attempt

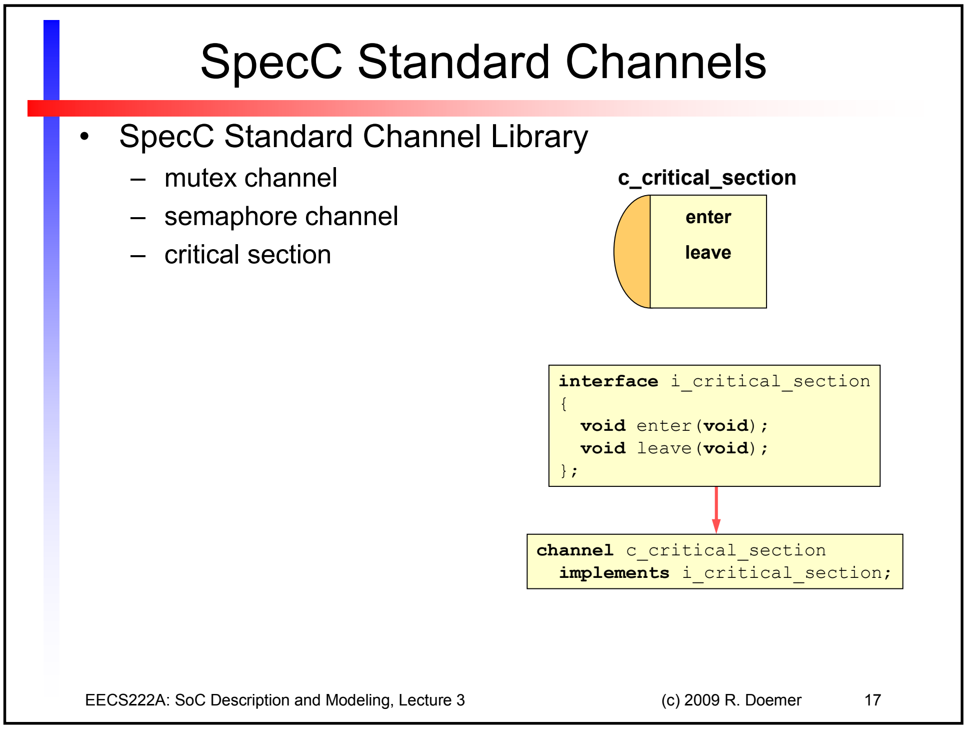
c_semaphore
acquire
release
attempt

```
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;
```

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 16



SpecC Standard Channels

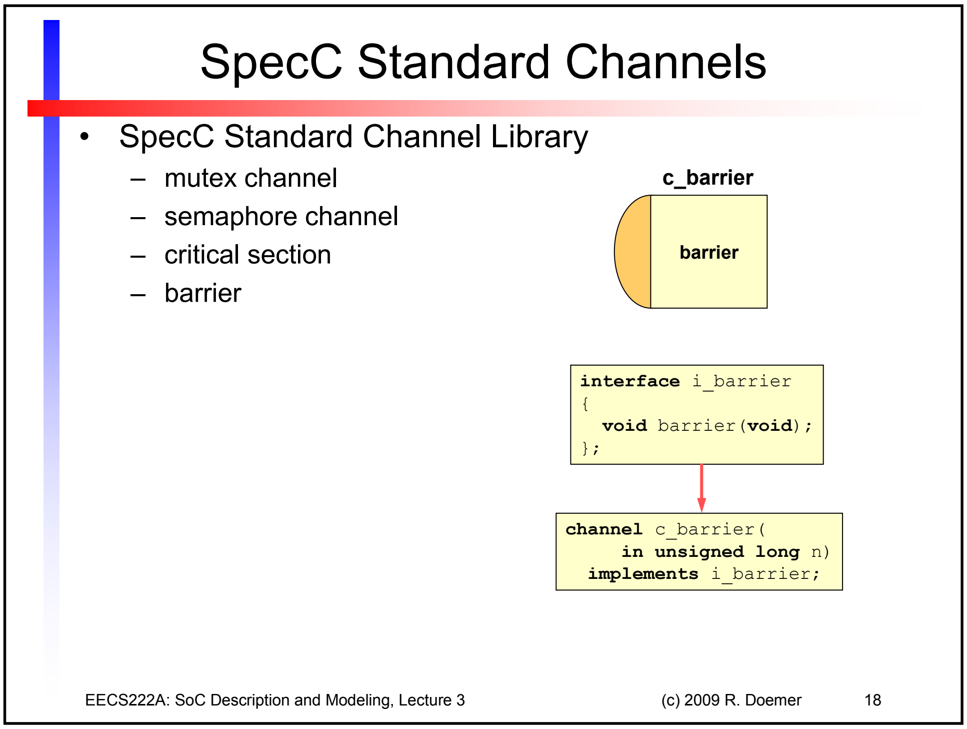
- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section

c_critical_section
enter
leave

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

```
channel c_critical_section
  implements i_critical_section;
```

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 17



SpecC Standard Channels

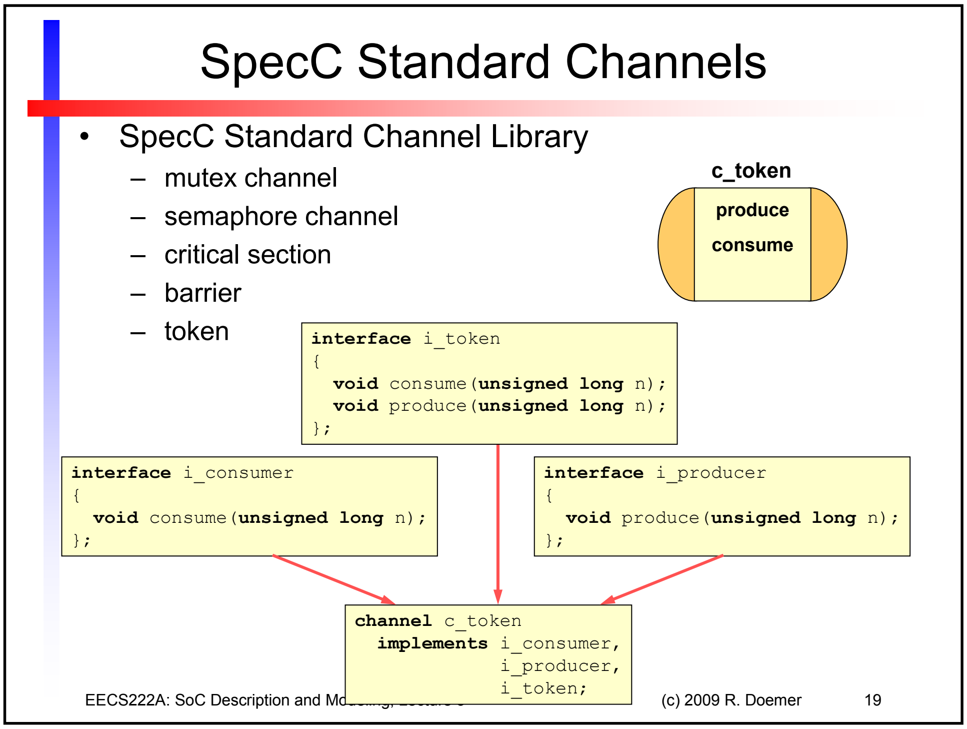
- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier

c_barrier
barrier

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

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

EECS222A: SoC Description and Modeling, Lecture 3 (c) 2009 R. Doemer 18



SpecC Standard Channels

- 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);
};
```

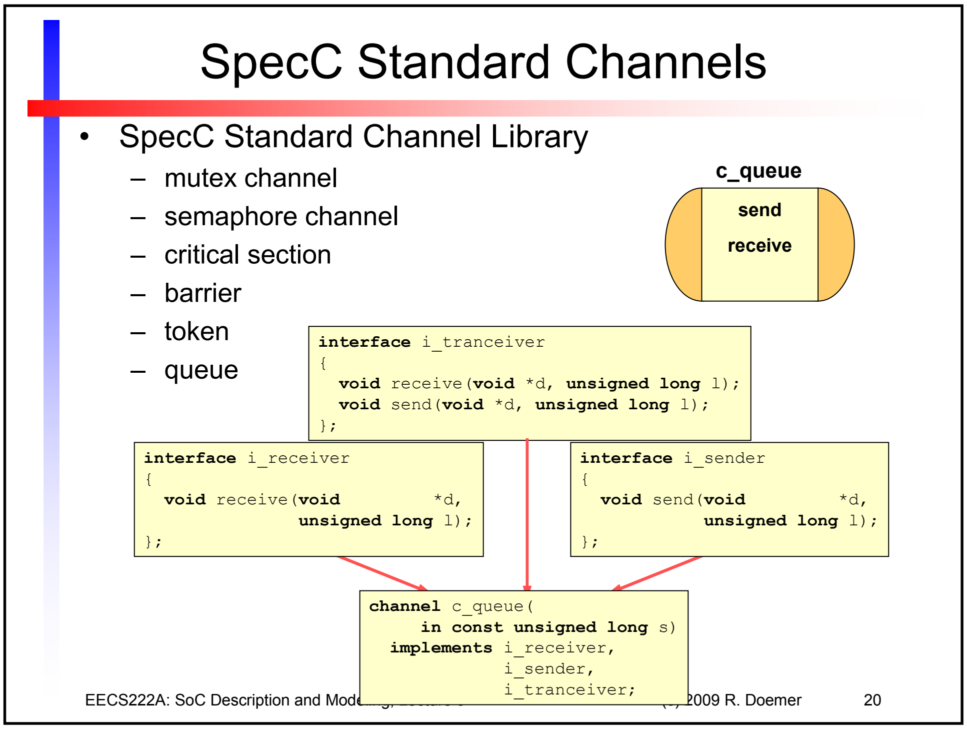
```
interface i_consumer
{
  void consume(unsigned long n);
};
```

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

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

EECS222A: SoC Description and Mo

(c) 2009 R. Doemer 19



SpecC Standard Channels

- 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);
};
```

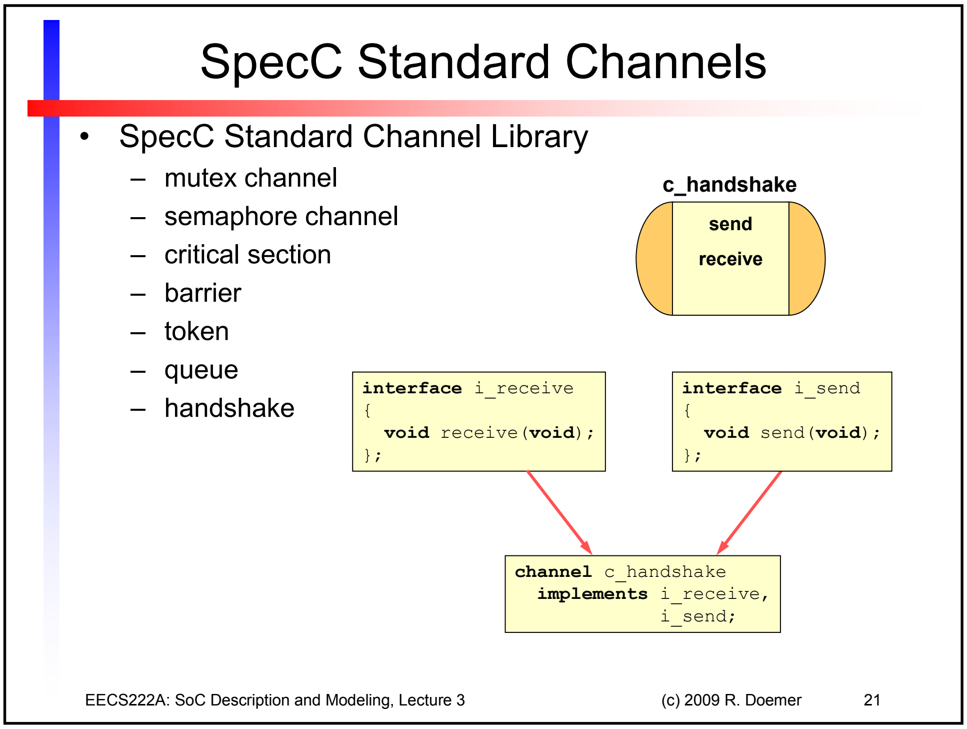
```
interface i_receiver
{
  void receive(void          *d,
               unsigned long l);
};
```

```
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;
```

EECS222A: SoC Description and Mode

2009 R. Doemer 20



SpecC Standard Channels

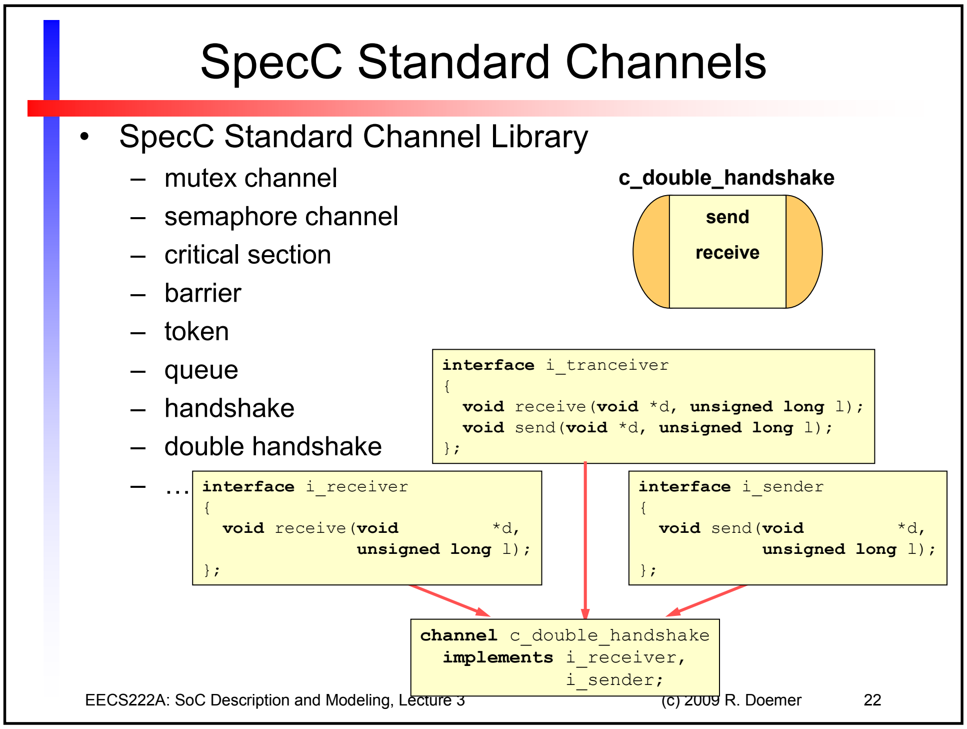
- SpecC Standard Channel Library
  - mutex channel
  - semaphore channel
  - critical section
  - barrier
  - token
  - queue
  - handshake

c_handshake
send
receive

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

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

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



SpecC Standard Channels

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

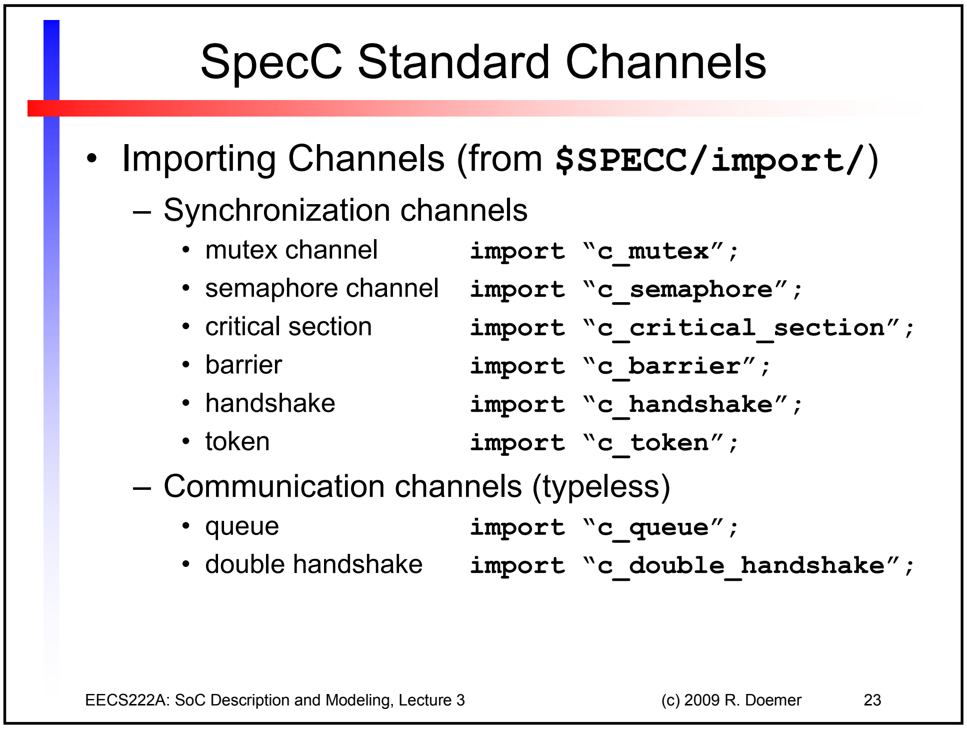
c_double_handshake
send
receive

```
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;
```



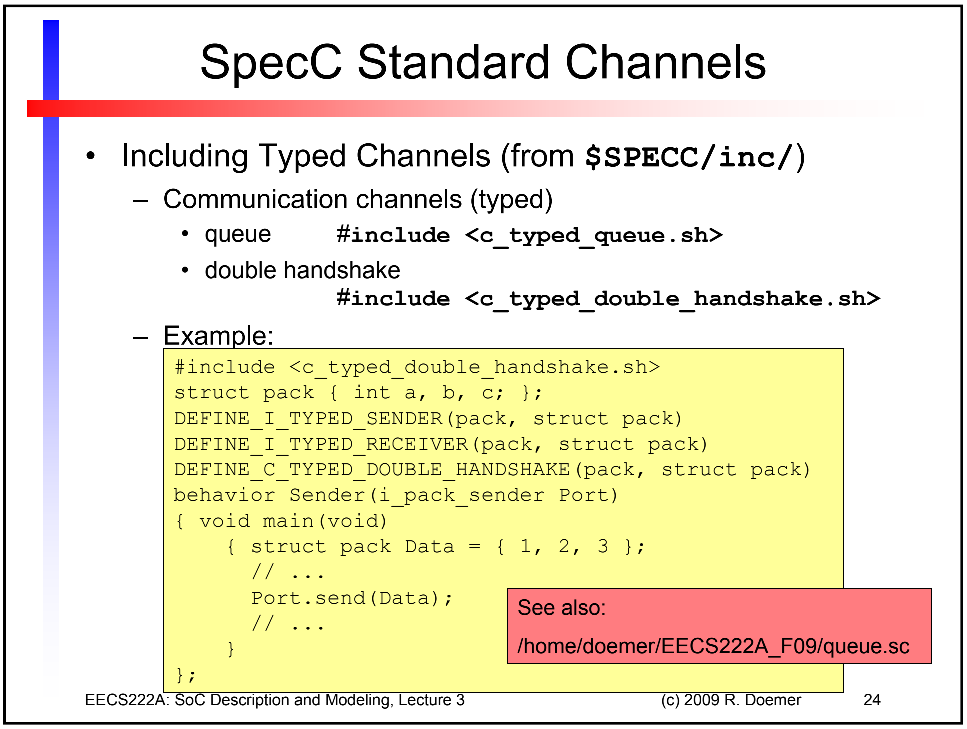
SpecC Standard Channels

- Importing Channels (from **$SPECC/import/**)
  - Synchronization channels

mutex channel	**import “c_mutex”;**
semaphore channel	**import “c_semaphore”;**
critical section	**import “c_critical_section”;**
barrier	**import “c_barrier”;**
handshake	**import “c_handshake”;**
token	**import “c_token”;**

  - Communication channels (typeless)

queue	**import “c_queue”;**
double handshake	**import “c_double_handshake”;**



SpecC Standard Channels

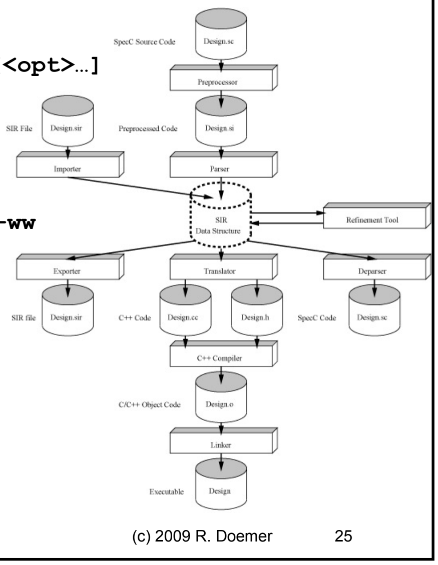
- Including Typed Channels (from **$SPECC/inc/**)
  - Communication channels (typed)
    - queue **#include <c_typed_queue.sh>**
    - double handshake
      #include <c_typed_double_handshake.sh>
  - Example:

```
#include <c_typed_double_handshake.sh>
struct pack { int a, b, c; };
DEFINE_I_TYPED_SENDER(pack, struct pack)
DEFINE_I_TYPED_RECEIVER(pack, struct pack)
DEFINE_C_TYPED_DOUBLE_HANDSHAKE(pack, struct pack)
behavior Sender(i_pack_sender Port)
{ void main(void)
    { struct pack Data = { 1, 2, 3 };
      // ...
      Port.send(Data);
      // ...
    }
};
```

See also:
/home/doemer/EECS222A_F09/queue.sc

The SpecC Compiler and Simulator

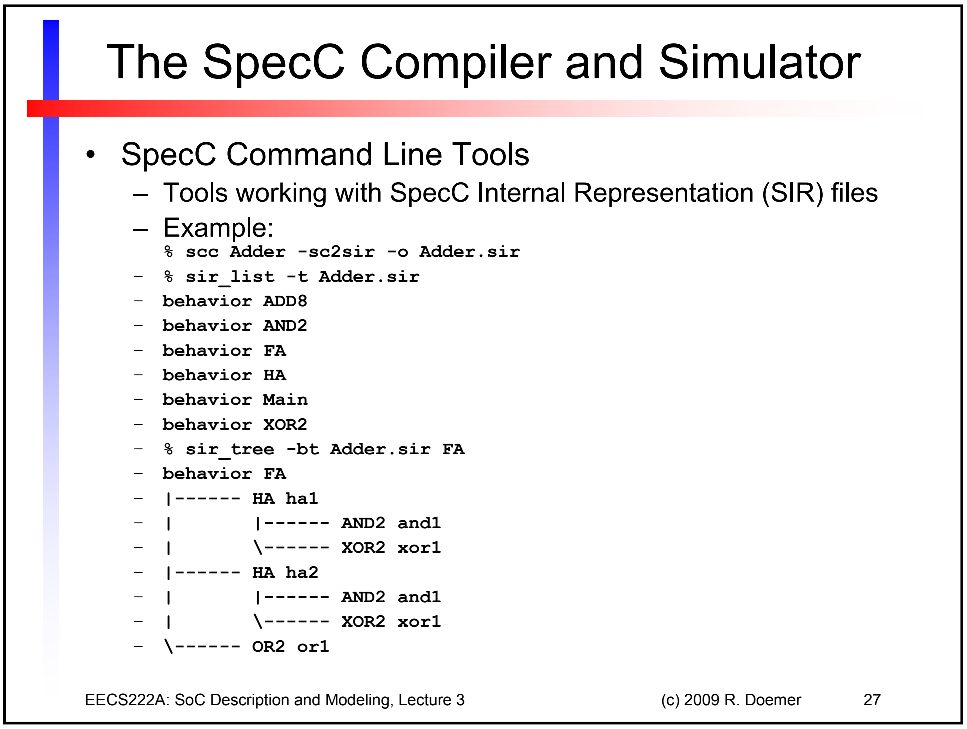
- SpecC Compiler
  - Command line interface
  - Usage: `scc <design> [<cmd>] [<opt>…]`
  - Help:
    ```
    scc -h
    man scc
    ```
  - Example:
    ```
    % scc HelloWorld -sc2out -v -ww
    scc: SpecC Compiler V 2.2.1
    (c)2008 CECS, UC Irvine
    Preprocessing...
    Parsing...
    Translating...
    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

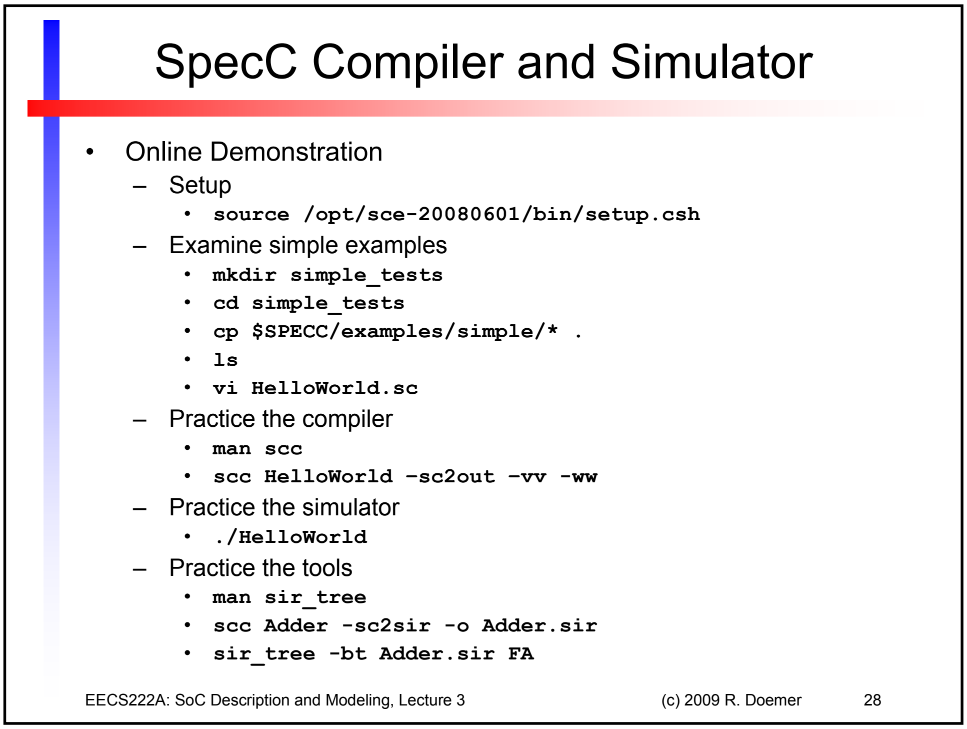
```
– #include <sim.sh>
– ...
– sim_time t;
– sim_delta d;
– sim_time_string buffer;
– ...
– t = now();  d = delta();
– printf("Time is now %s pico seconds.\n", time2str(buffer, t));
– printf("(delta count is %s)\n", time2str(buffer, d);
– waitfor 10 NANO_SEC;
– printf("Time is now %s pico seconds.\n",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



The SpecC Compiler and Simulator

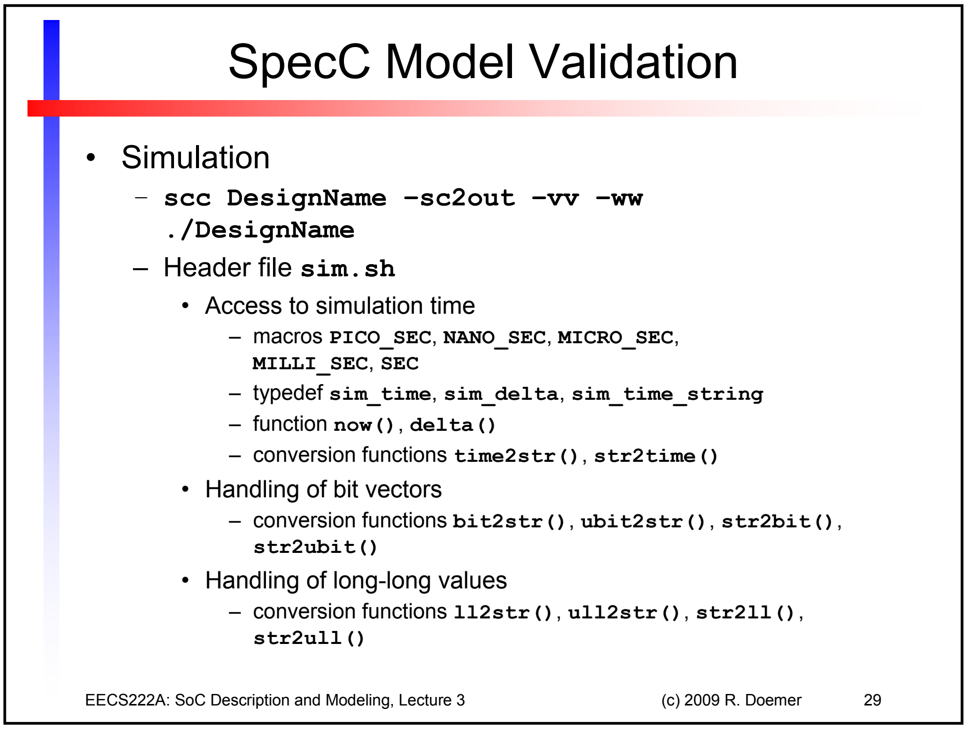
- SpecC Command Line Tools
  - Tools working with SpecC Internal Representation (SIR) files
  - Example:

```
  % 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
```



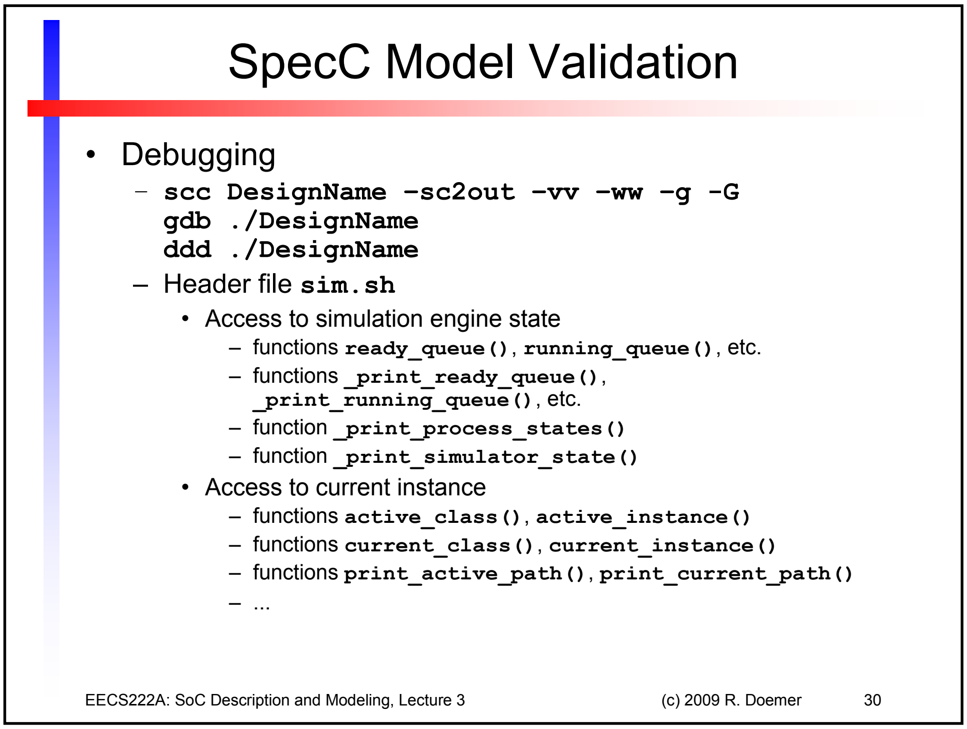
SpecC Compiler and Simulator

- Online Demonstration
  - 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`



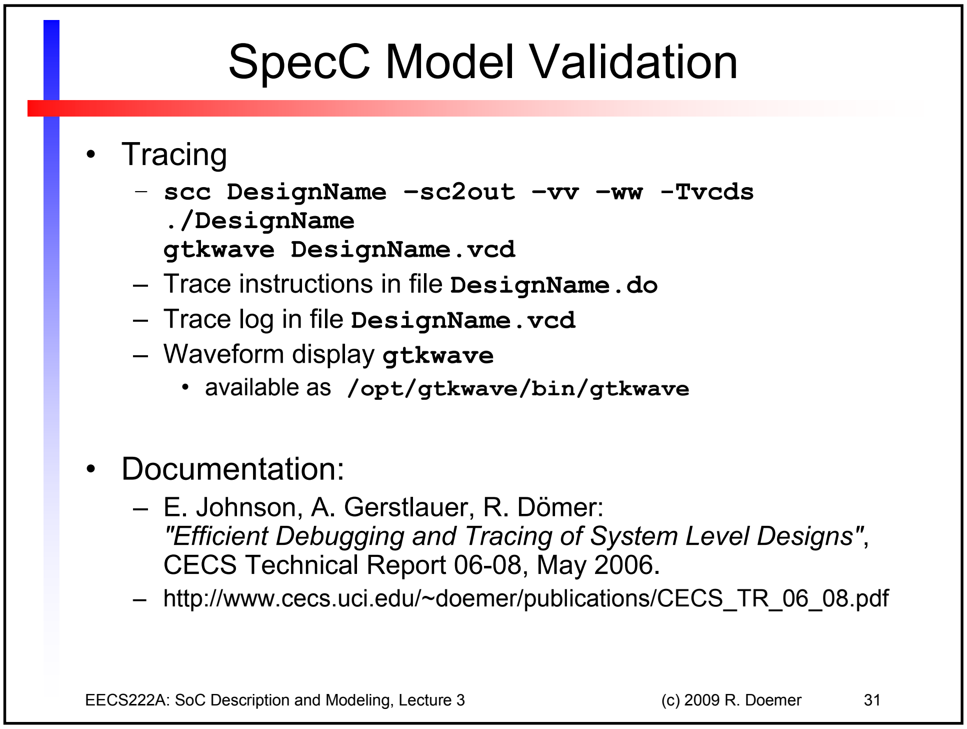
SpecC Model Validation

- Simulation
  - `scc DesignName -sc2out -vv -ww`
    `./DesignName`
  - Header file `sim.sh`
    - Access to simulation time
      - macros `PICO_SEC`, `NANO_SEC`, `MICRO_SEC`, `MILLI_SEC`, `SEC`
      - typedef `sim_time`, `sim_delta`, `sim_time_string`
      - function `now()`, `delta()`
      - conversion functions `time2str()`, `str2time()`
    - Handling of bit vectors
      - conversion functions `bit2str()`, `ubit2str()`, `str2bit()`, `str2ubit()`
    - Handling of long-long values
      - conversion functions `ll2str()`, `ull2str()`, `str2ll()`, `str2ull()`



SpecC Model Validation

- Debugging
  - `scc DesignName -sc2out -vv -ww -g -G`
    `gdb ./DesignName`
    `ddd ./DesignName`
  - Header file `sim.sh`
    - Access to simulation engine state
      - functions `ready_queue()`, `running_queue()`, etc.
      - functions `_print_ready_queue()`, `_print_running_queue()`, etc.
      - function `_print_process_states()`
      - function `_print_simulator_state()`
    - Access to current instance
      - functions `active_class()`, `active_instance()`
      - functions `current_class()`, `current_instance()`
      - functions `print_active_path()`, `print_current_path()`
      - ...



SpecC Model Validation

- Tracing

  -
    ```
    scc DesignName -sc2out -vv -ww -Tvcds
    ./DesignName
    gtkwave DesignName.vcd
    ```
  - Trace instructions in file `DesignName.do`
  - Trace log in file `DesignName.vcd`
  - Waveform display `gtkwave`
    - available as `/opt/gtkwave/bin/gtkwave`

- Documentation:
  - E. Johnson, A. Gerstlauer, R. Dömer: *"Efficient Debugging and Tracing of System Level Designs"*, CECS Technical Report 06-08, May 2006.
  - http://www.cecs.uci.edu/~doemer/publications/CECS_TR_06_08.pdf