EE382V: Embedded System Design and Modeling

Lecture 3 – The SpecC Language & Methodology

Source: R. Doemer, UC Irvine

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
`gerstl@ece.utexas.edu`

Lecture 3: Outline

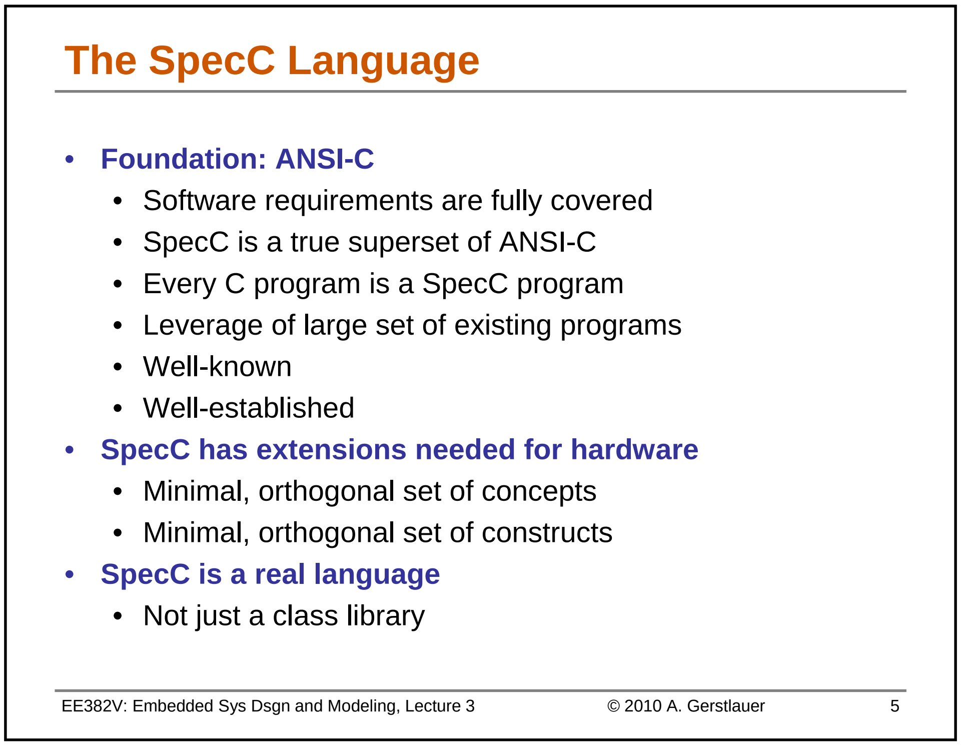
- **SpecC Language**
  - Syntax and semantics of core language
  - Channel library
- **SpecC tools**
  - Compiler and simulator
- **SpecC methodology**
  - Computation and communication refinement

Lecture 3: Outline

- **SpecC Language**
  - Syntax and semantics of core language
    - Foundation
    - Types
    - Structural and behavioral hierarchy
    - Concurrency
    - State transitions
    - Exception handling
    - Communication
    - Synchronization
    - Library Support
    - Persistent Annotation
    - Timing
    - (RTL)
- **SpecC tools**
  - Compiler and simulator
- **SpecC methodology**
  - Computation and communication refinement

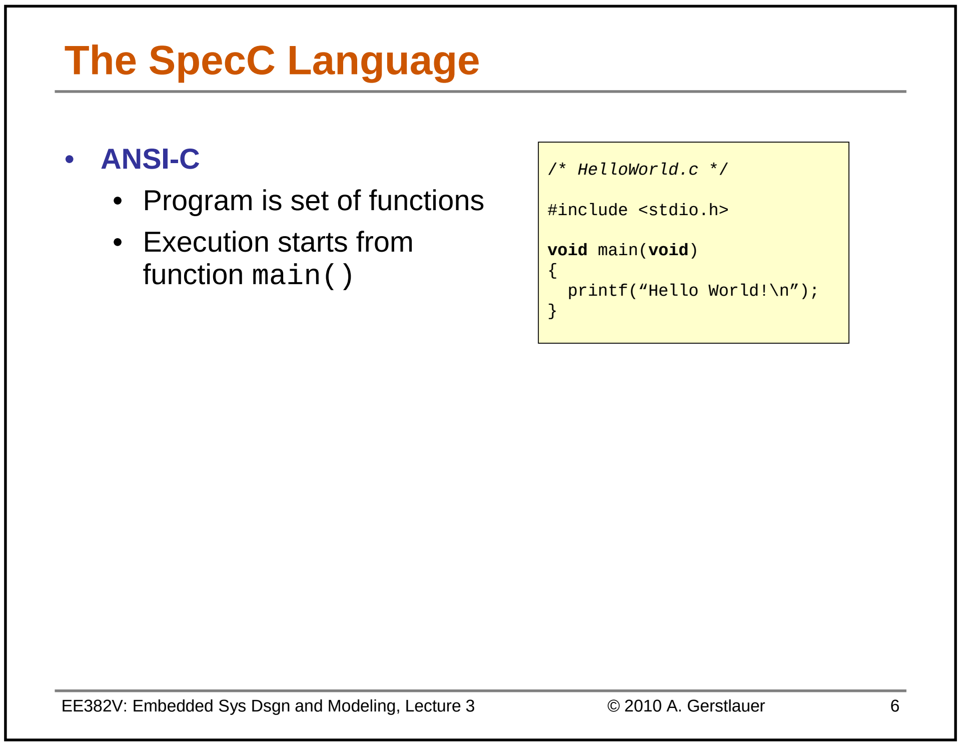
The SpecC Language

- **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

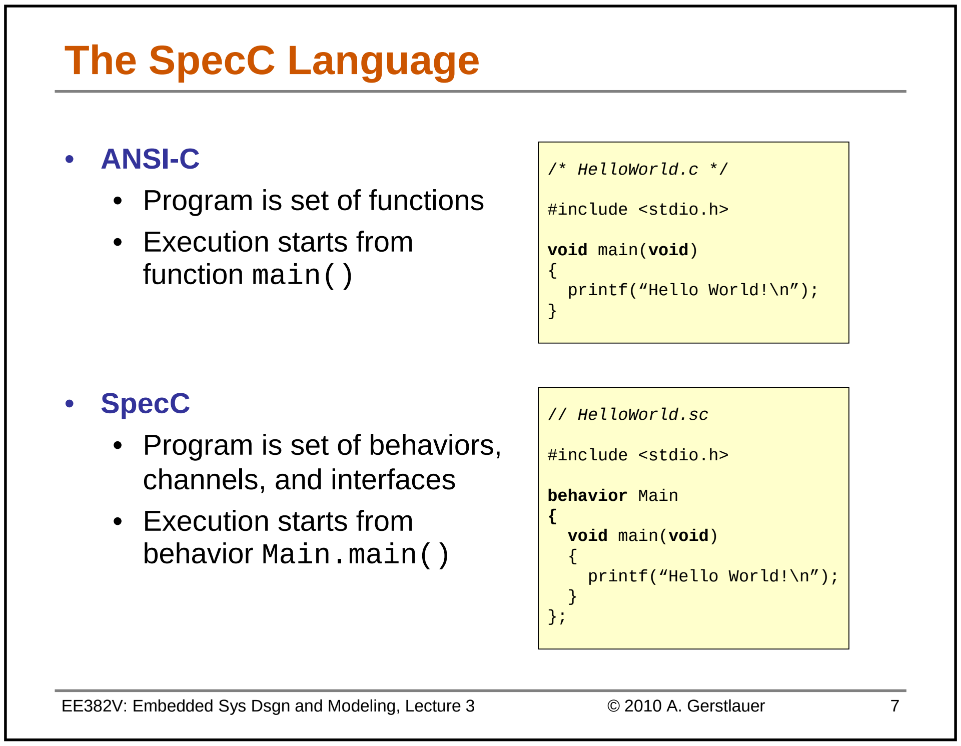
- **Foundation: ANSI-C**
  - Software requirements are fully covered
  - SpecC is a true superset of ANSI-C
  - Every C program is a SpecC program
  - Leverage of large set of existing programs
  - Well-known
  - Well-established
- **SpecC has extensions needed for hardware**
  - Minimal, orthogonal 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

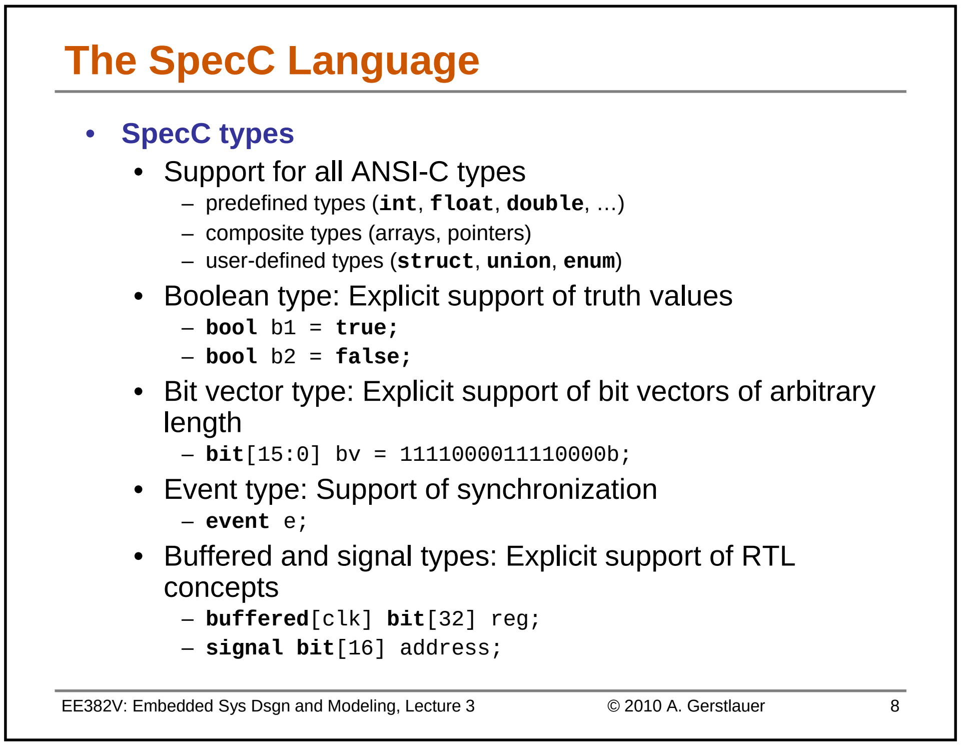
- **ANSI-C**
  - Program is set of functions
  - Execution starts from function `main()`

```
/* 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");
  }
};
```

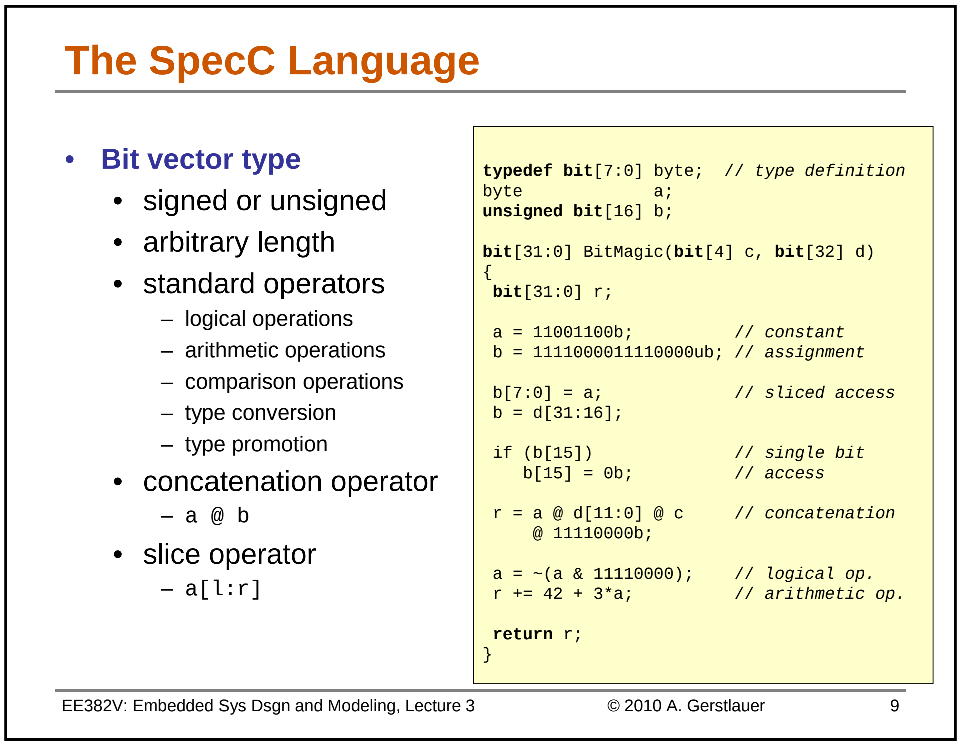
EE382V: Embedded Sys Dsgn and Modeling, Lecture 3   © 2010 A. Gerstlauer   7



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;`

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3   © 2010 A. Gerstlauer   8

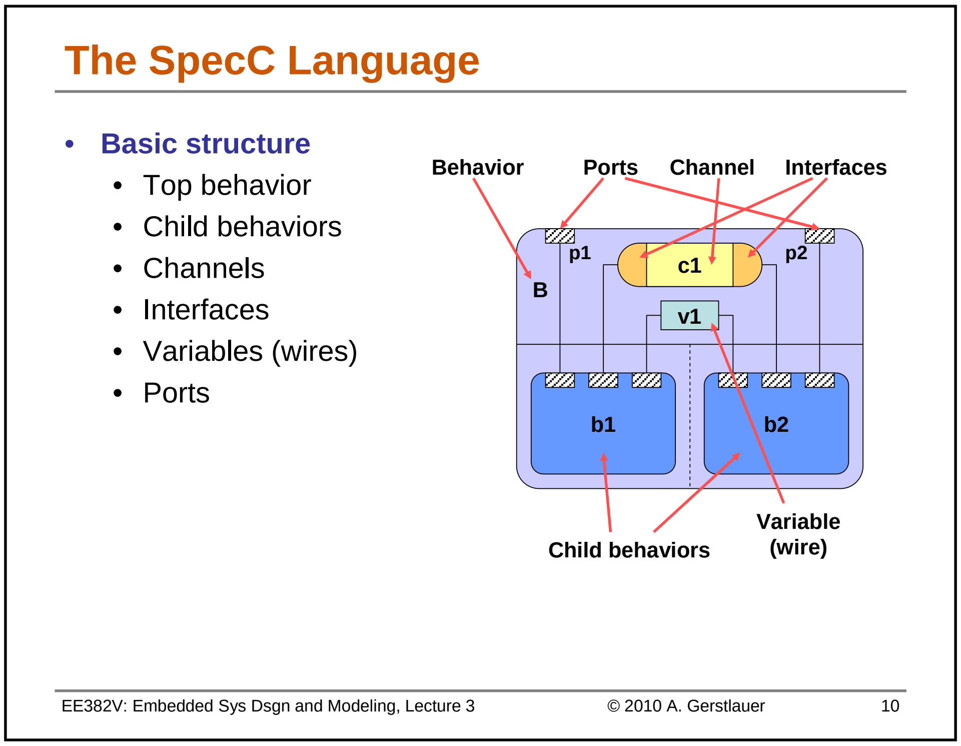


The SpecC Language

- **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]`

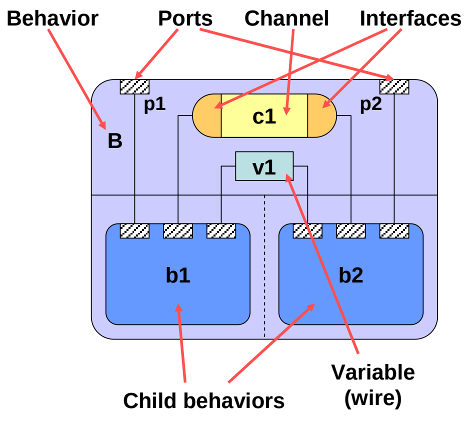
```
typedef bit[7:0] byte;  // type definition
byte             a;
unsigned bit[16] b;

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;
}
```

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 9



The SpecC Language

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

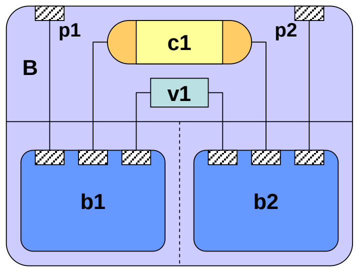


EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 10

The SpecC Language

- **Basic structure**

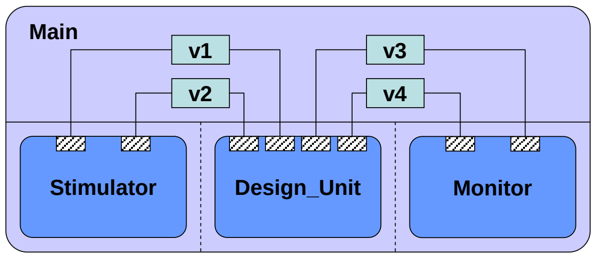
```
interface I1
{
  bit[63:0] Read(void);
  void Write(bit[63:0]);
};

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;
        }
  }
};
```



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

The SpecC Language

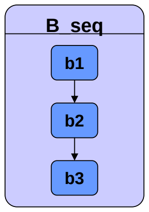
- **Typical test bench**
  - Top-level behavior: `Main`
  - Stimulator provides test vectors
  - Design unit under test
  - Monitor observes and checks outputs



The SpecC Language

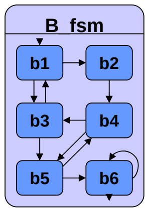
- Behavioral hierarchy

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:{…}
         …}
  }
};
```

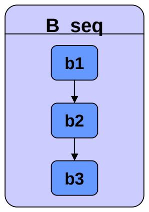
Concurrent execution

Pipelined execution

The SpecC Language

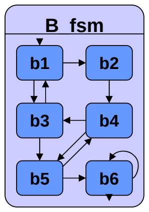
- Behavioral hierarchy

Sequential execution



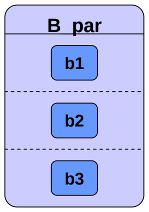
```
behavior B_seq
{
 B b1, b2, b3;

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

FSM execution



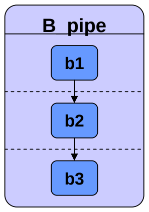
```
behavior B_fsm
{
 B b1, b2, b3,
   b4, b5, b6;
 void main(void)
 { fsm { b1:{…}
         b2:{…}
         …}
  }
};
```

Concurrent execution



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

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

Pipelined execution

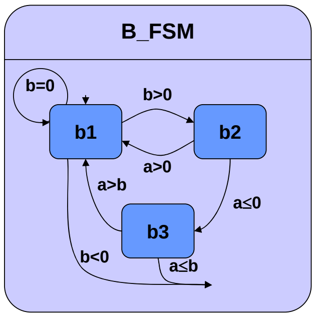


```
behavior B_pipe
{
 B b1, b2, b3;

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

The SpecC Language

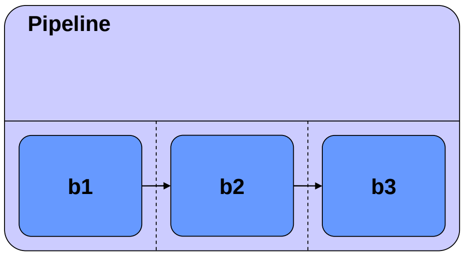
- **Finite State Machine (FSM)**
  - Explicit state transitions
    - triple < *current_state, condition, next_state* >
    - **fsm** { <*current_state*> : { **if** <*condition*> **goto** <*next_state*> } … }
  - Moore-type FSM
  - Mealy-type FSM

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



The SpecC Language

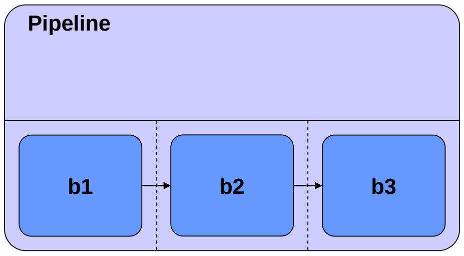
- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };



```
behavior Pipeline
{



 Stage1 b1;
 Stage2 b2;
 Stage3 b3;

 void main(void)
 {

   pipe
       { b1;
         b2;
         b3;
        }
  }
};
```

The SpecC Language

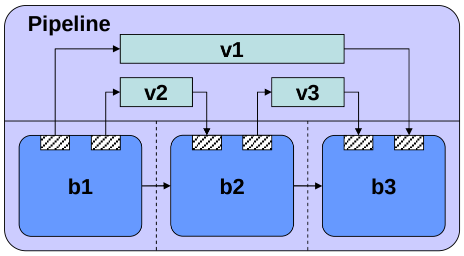
- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }



```
behavior Pipeline
{



 Stage1 b1;
 Stage2 b2;
 Stage3 b3;

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

The SpecC Language

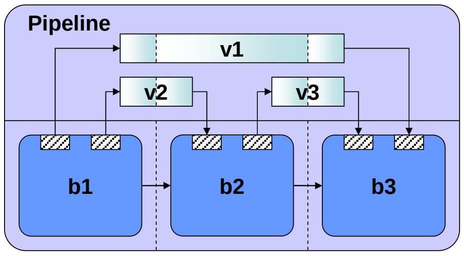
- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }
  - Support for automatic buffering



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

The SpecC Language

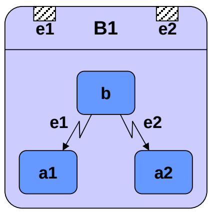
- **Pipeline**
  - Explicit execution in pipeline fashion
    - **pipe** { *<instance_list>* };
    - **pipe** (*<init>*; *<cond>*; *<incr>*) { … }
  - Support for automatic buffering
    - **piped** […] *<type>* *<variable_list>*;



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

The SpecC Language

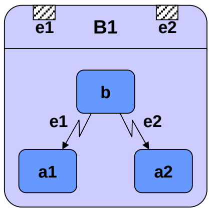
- **Exception handling**
  - Abortion
  - Interrupt

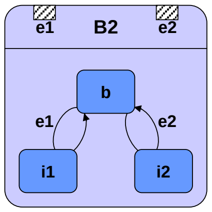


```
behavior B1(in event e1, in event e2)
{
 B b, a1, a2;

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

The SpecC Language

- **Exception handling**
  - Abortion
  - Interrupt



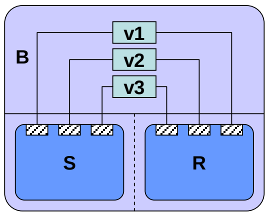


```
behavior B1(in event e1, in event e2)
{
 B b, a1, a2;

 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

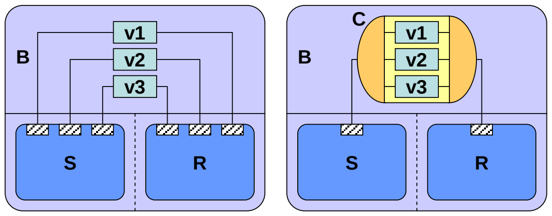
- **Communication**
  - via shared variable



Shared memory

The SpecC Language

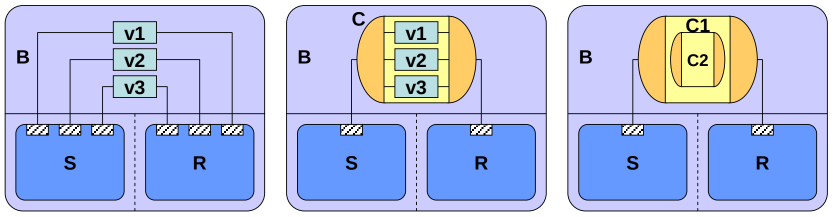
- **Communication**
  - via shared variable
  - via virtual channel



Shared memory **Message passing**

The SpecC Language

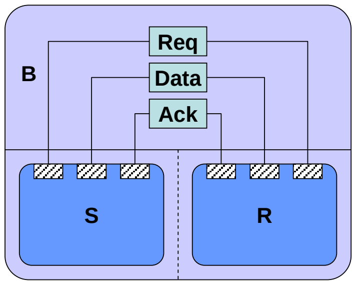
- **Communication**
  - via shared variable
  - via virtual channel
  - via hierarchical channel



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*>;

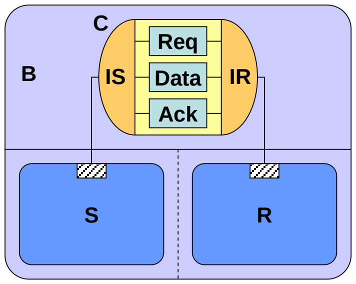


```
behavior S(out event Req,
           out float Data,
           in  event Ack)
{
 float X;
 void main(void)
 { ...
   Data = X;
   notify Req;
   wait Ack;
   ...
 }
};
```

```
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*> };



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

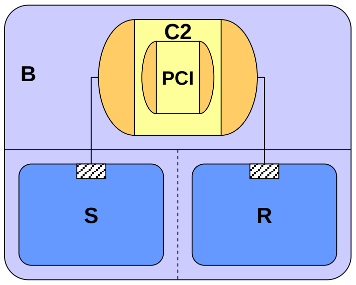
```
behavior R(IR Port)
{
 float Y;
 void main(void)
 {...
  Y=Port.Receive();
  ...
 }
};
```

```
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

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



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

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

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

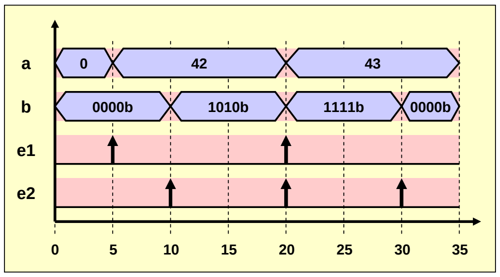
```
channel PCI
   implements PCI_IF;

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;
 }
};
```

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

The SpecC Language

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

Example: stimulator for a test bench

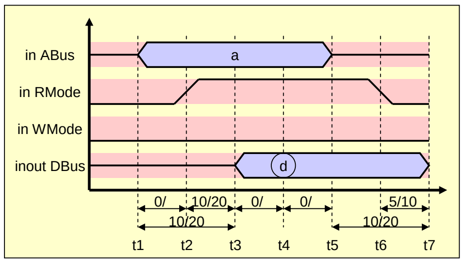


```
behavior Testbench_Driver
         (inout int a,
          inout int b,
          out event e1,
          out event e2)
{
  void main(void)
  {
   waitfor 5;
   a = 42;
   notify e1;

   waitfor 5;
   b = 1010b;
   notify e2;

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

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

The SpecC Language

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

Example: SRAM read protocol



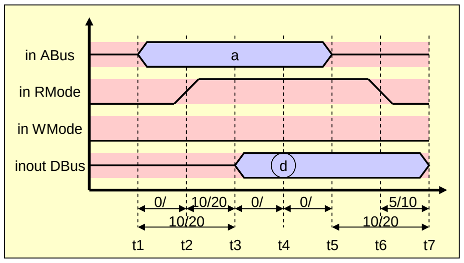
Specification

```
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

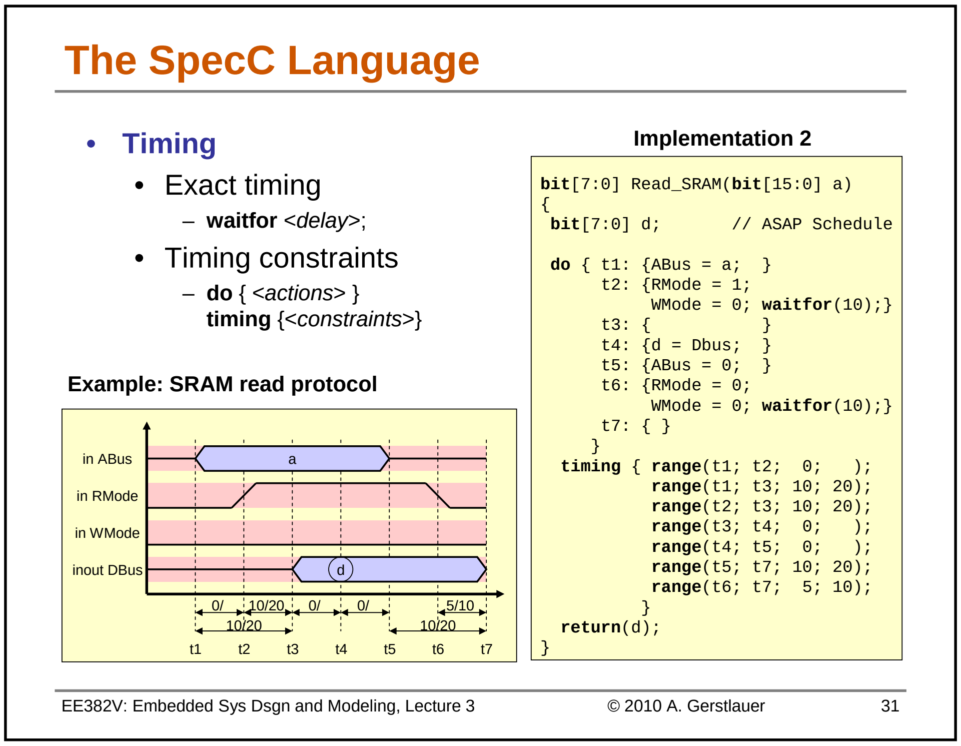
- **Timing**
  - Exact timing
    - **waitfor** <*delay*>;
  - Timing constraints
    - **do** { <*actions*> }
      timing {<*constraints*>}

Example: SRAM read protocol



Implementation 1

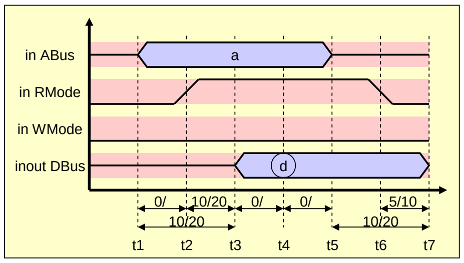
```
bit[7:0] Read_SRAM(bit[15:0] a)
{
 bit[7:0] d;

 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*>}

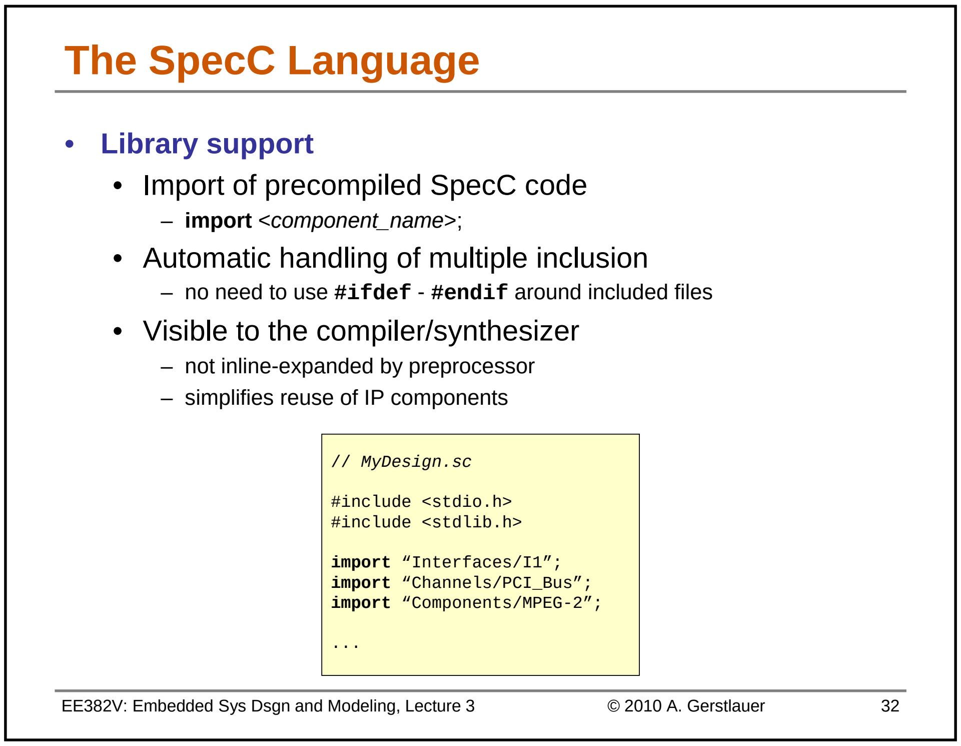
Example: SRAM read protocol



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);
}
```

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 31



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”;

...
```

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 32

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;
  ...
};
```

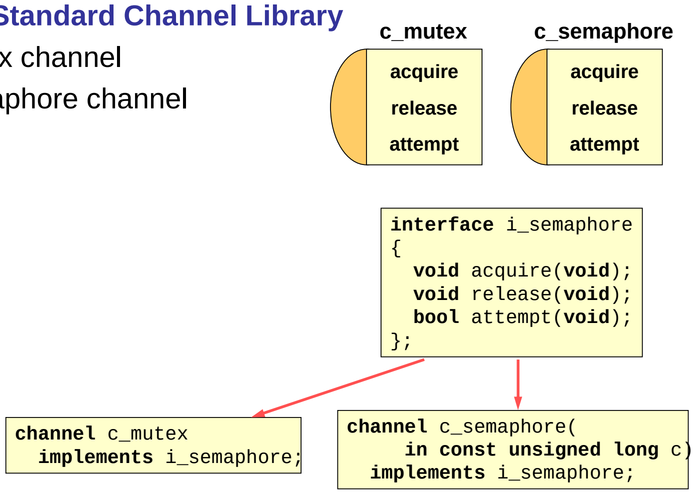
The SpecC Language

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

➢ Examples under
  ➢ **`$SPECC/examples/sync`**

The SpecC Language

- **SpecC Standard Channel Library**
  - mutex channel
  - semaphore channel



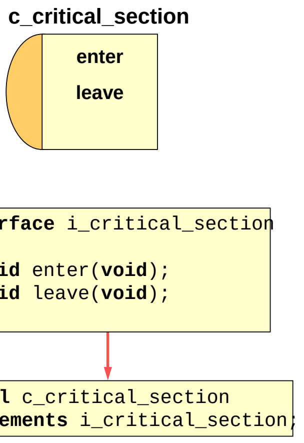
```
interface i_semaphore
{
  void acquire(void);
  void release(void);
  bool attempt(void);
};
```

```
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

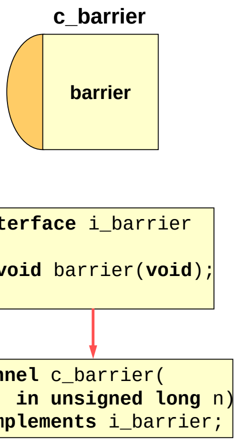


```
interface i_critical_section
{
  void enter(void);
  void leave(void);
};
```

```
channel c_critical_section
  implements i_critical_section;
```

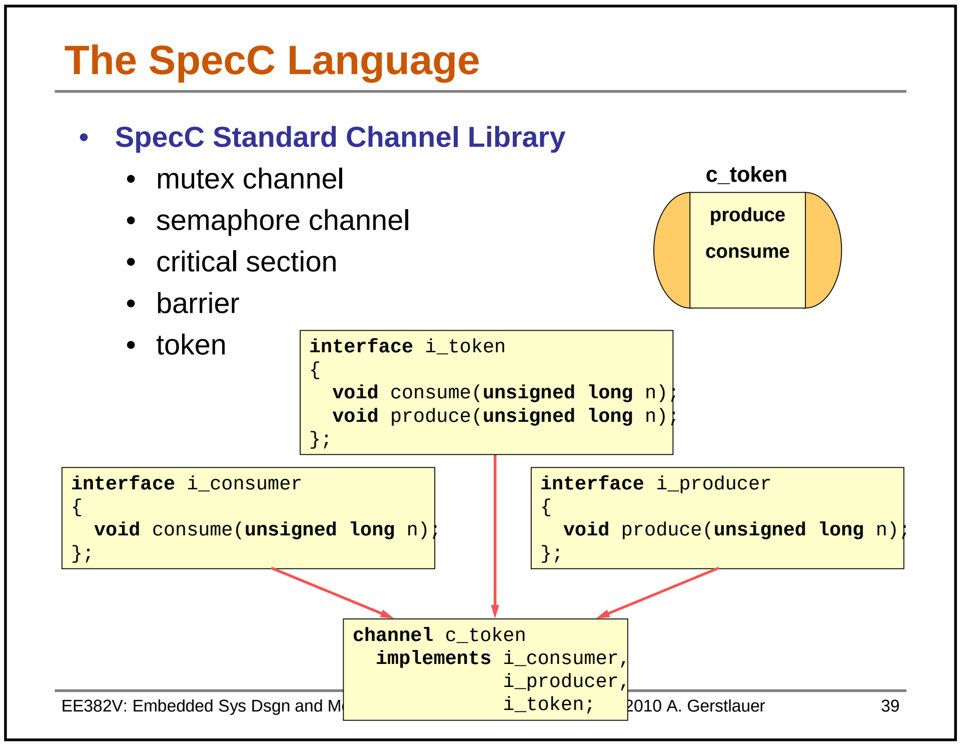
The SpecC Language

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



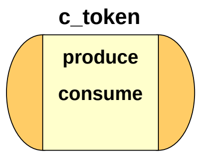
```
interface i_barrier
{
  void barrier(void);
};
```

```
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



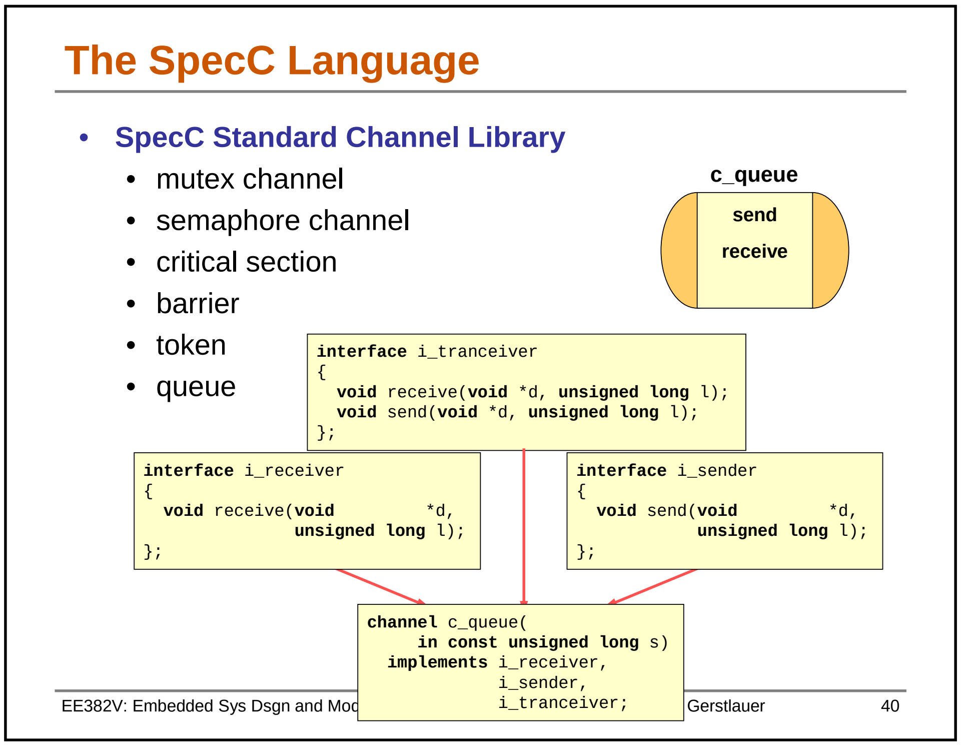
```
interface i_token
{
  void consume(unsigned long n);
  void produce(unsigned long n);
};
```

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

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

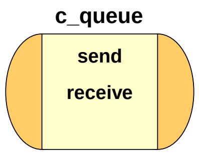
```
channel c_token
  implements i_consumer,
             i_producer,
             i_token;
```

EE382V: Embedded Sys Dsgn and M… 2010 A. Gerstlauer 39



The SpecC Language

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



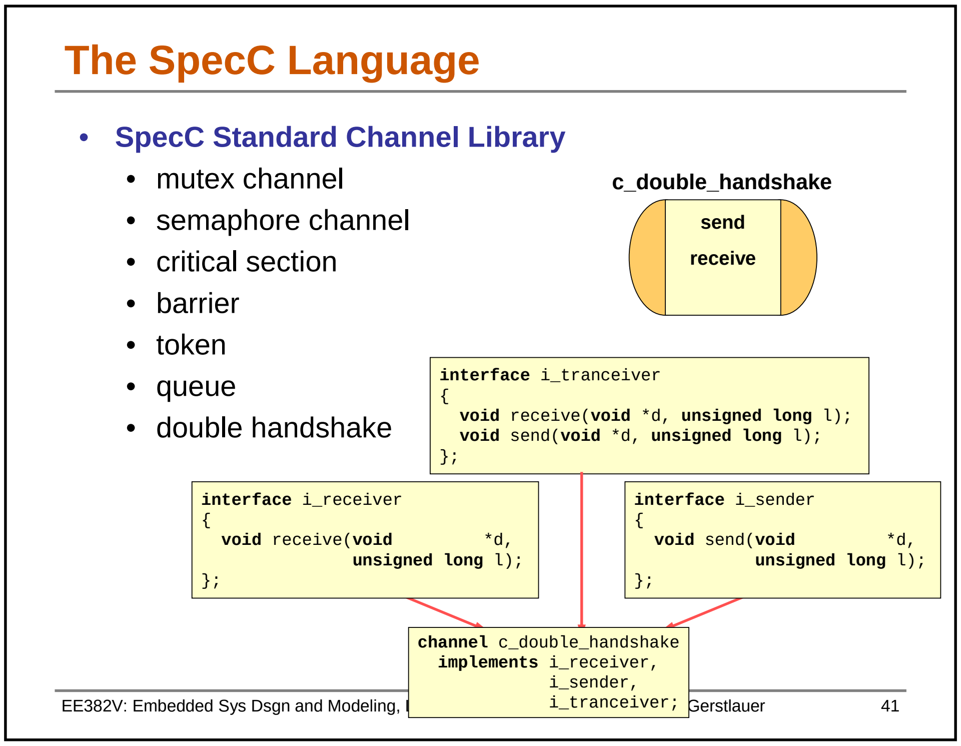
```
interface i_tranceiver
{
  void receive(void *d, unsigned long l);
  void send(void *d, unsigned long l);
};
```

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

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

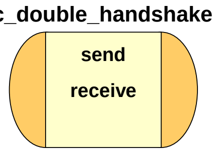
```
channel c_queue(
    in const unsigned long s)
  implements i_receiver,
             i_sender,
             i_tranceiver;
```

EE382V: Embedded Sys Dsgn and Mod… Gerstlauer 40



The SpecC Language

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



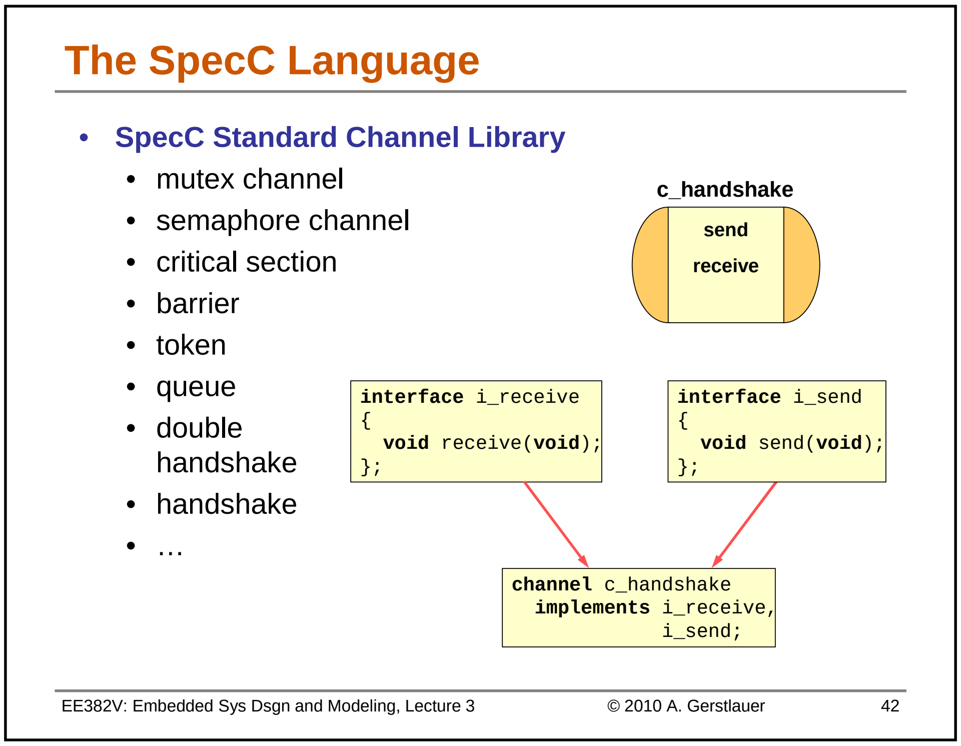
```
interface i_tranceiver
{
  void receive(void *d, unsigned long l);
  void send(void *d, unsigned long l);
};
```

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

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

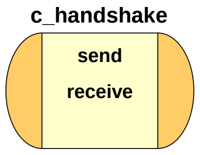
```
channel c_double_handshake
  implements i_receiver,
             i_sender,
             i_tranceiver;
```

EE382V: Embedded Sys Dsgn and Modeling, Gerstlauer 41



The SpecC Language

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



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

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

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

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 42

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";`

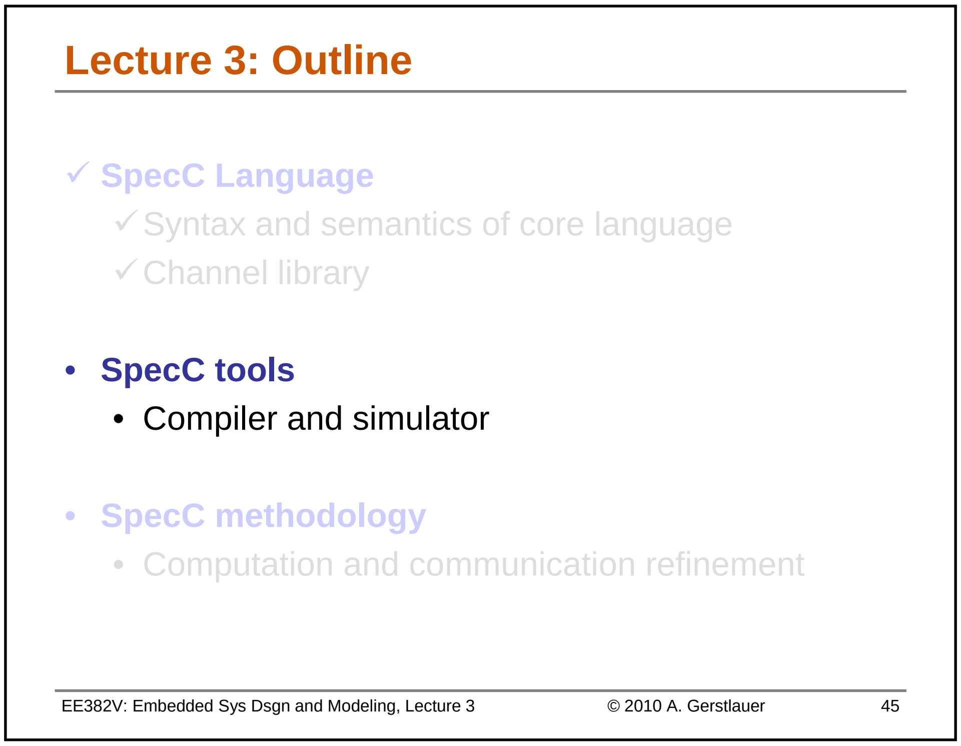
  - Communication channels (typeless)

– 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:

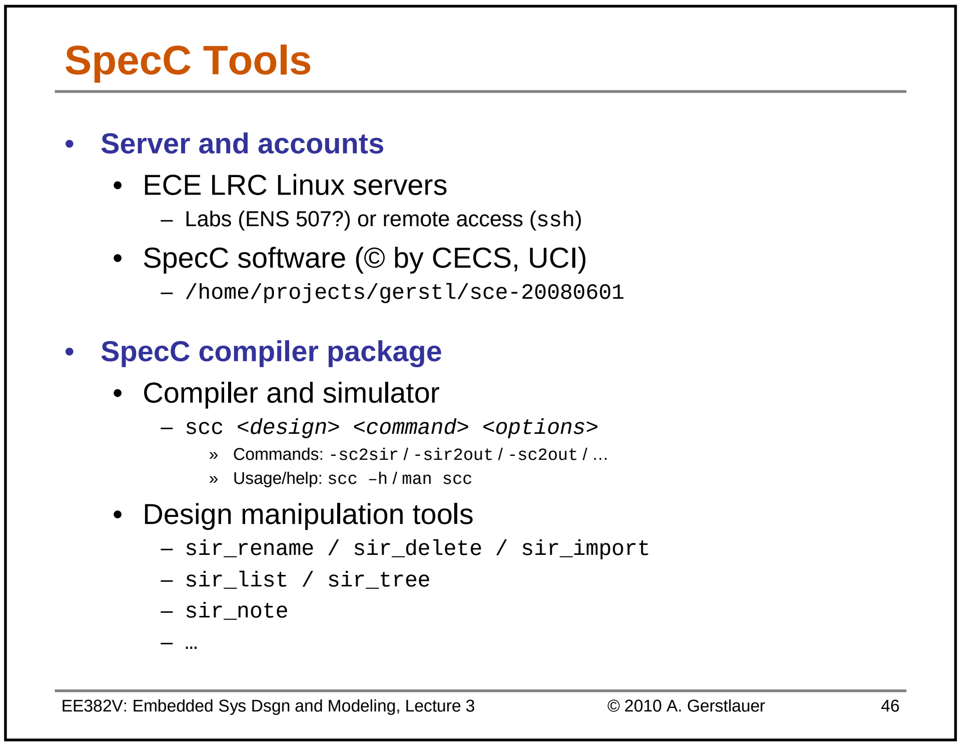
```
#include <c_typed_double_handshake.sh>

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);
      // ...
    }
};
```



Lecture 3: Outline

- ✓ SpecC Language
  - ✓ Syntax and semantics of core language
  - ✓ Channel library
- SpecC tools
  - Compiler and simulator
- SpecC methodology
  - Computation and communication refinement

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 45



SpecC Tools

- **Server and accounts**
  - ECE LRC Linux servers
    - Labs (ENS 507?) or remote access (`ssh`)
  - SpecC software (© by CECS, UCI)
    - `/home/projects/gerstl/sce-20080601`
- **SpecC compiler package**
  - Compiler and simulator
    - `scc` *`<design>`* *`<command>`* *`<options>`*
      - Commands: `-sc2sir` / `-sir2out` / `-sc2out` / …
      - Usage/help: `scc –h` / `man scc`
  - Design manipulation tools
    - `sir_rename / sir_delete / sir_import`
    - `sir_list / sir_tree`
    - `sir_note`
    - …

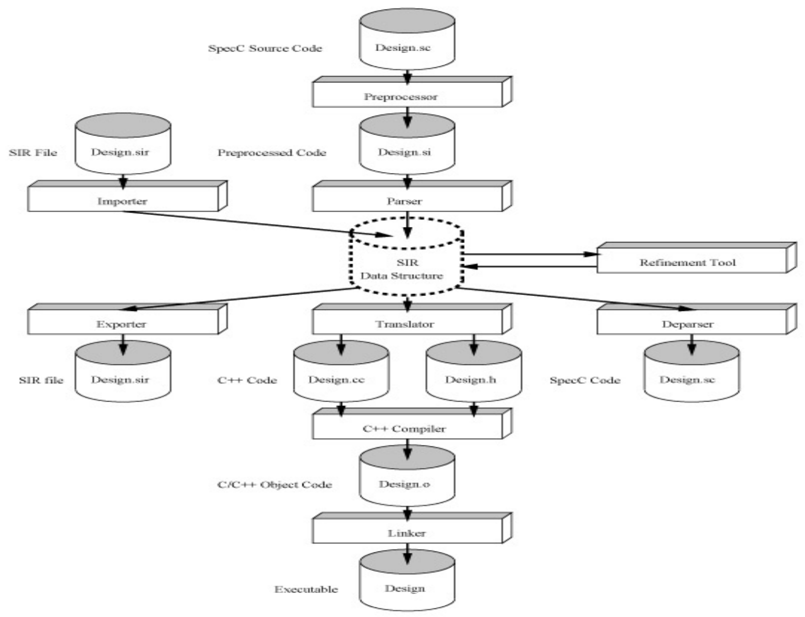
EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 46

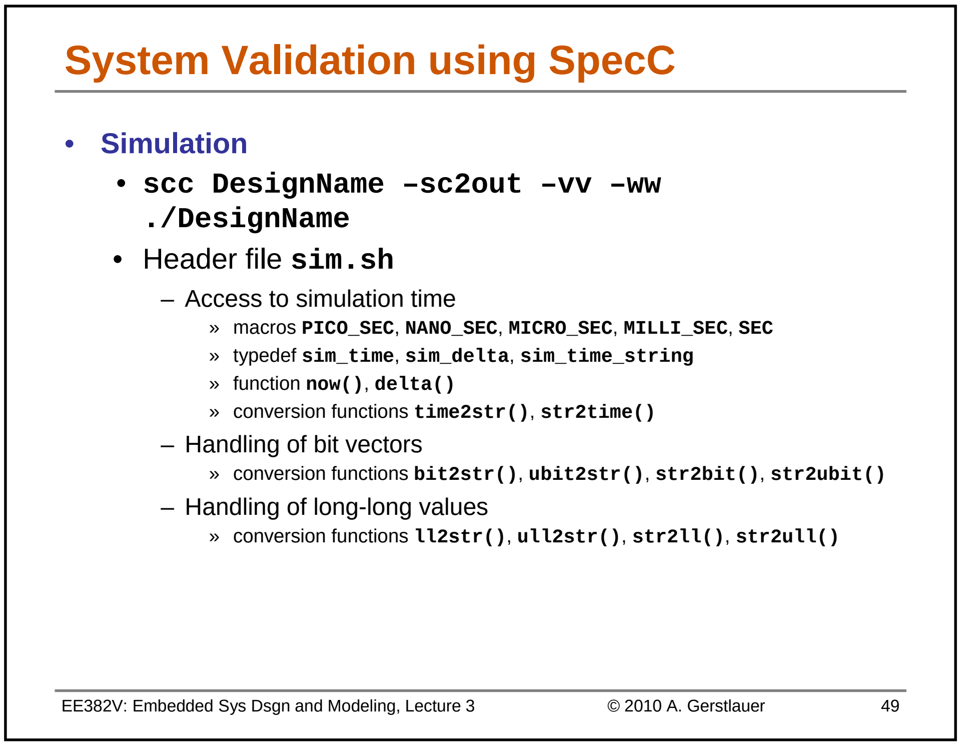
SpecC Tools

- **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));
...
```

SpecC Compiler (scc)

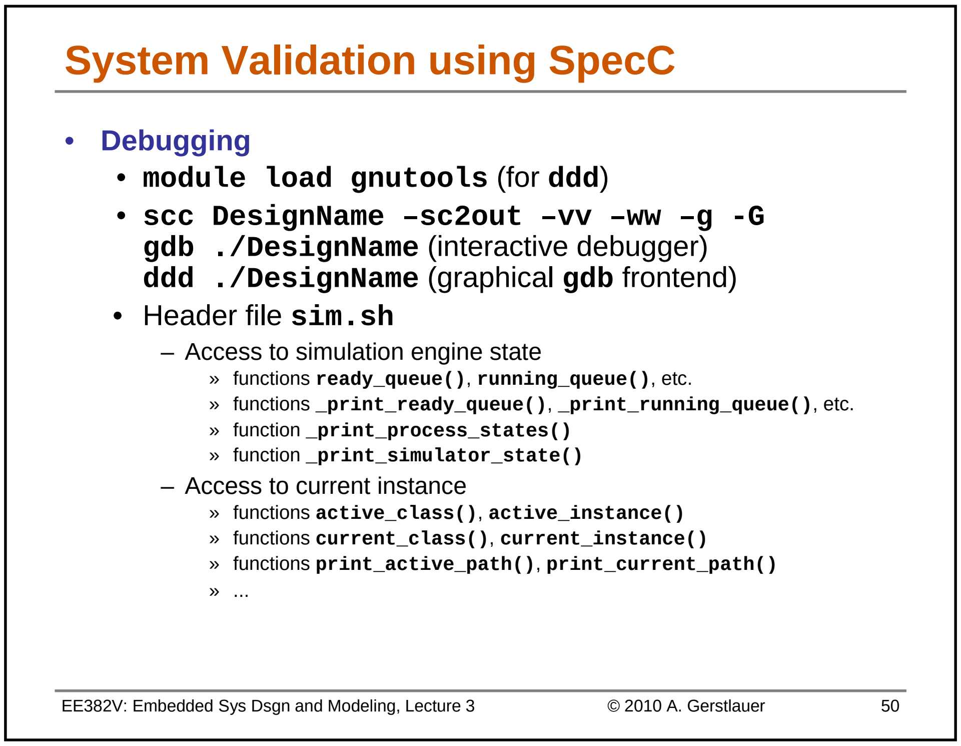




System Validation using SpecC

- **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()`**

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 49



System Validation using SpecC

- **Debugging**
  - **`module load gnutools`** (for **`ddd`**)
  - **`scc DesignName –sc2out –vv –ww –g -G`**
    `gdb ./DesignName` (interactive debugger)
    `ddd ./DesignName` (graphical **`gdb`** frontend)
  - 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()`**
      - » ...

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 50

System Validation using SpecC

- Tracing
  - `module load gnutools` (for `gtkwave`)
  - `scc DesignName –sc2out –vv –ww -Tvcds`
    `./DesignName`
    `gtkwave DesignName.vcd`
  - Trace instructions in file **`DesignName.do`**
  - Trace log in file **`DesignName.vcd`**
  - Waveform display, e.g. **`gtkwave`**

➢ Examples
  ➢ **`$SPECC/examples/trace`**, see README

➢ Documentation
  ➢ E. Johnson, A. Gerstlauer, R. Dömer: *"Efficient Debugging and Tracing of System Level Designs"*, CECS Technical Report 06-08, May 2006.

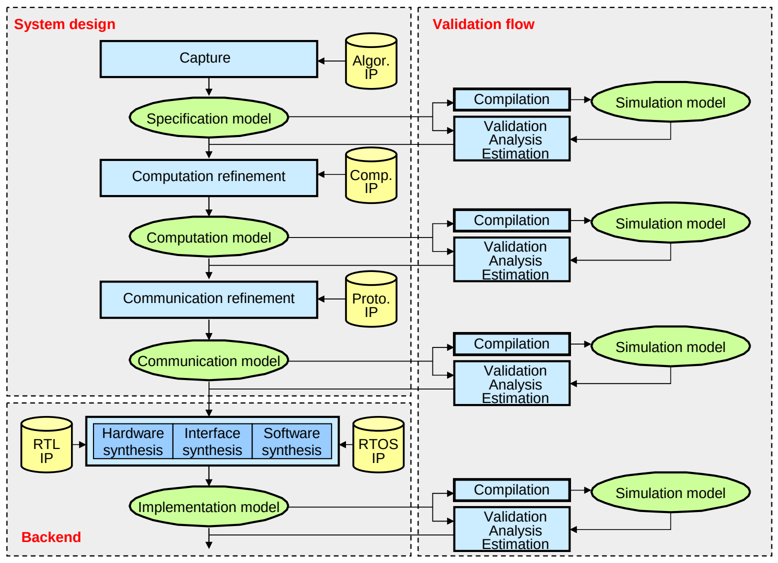
Lecture 3: Outline

✓ SpecC Language
  ✓ Syntax and semantics of core language
  ✓ Channel library

✓ SpecC tools
  ✓ Compiler and simulator

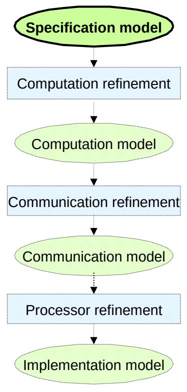
- SpecC methodology
  - Computation and communication refinement

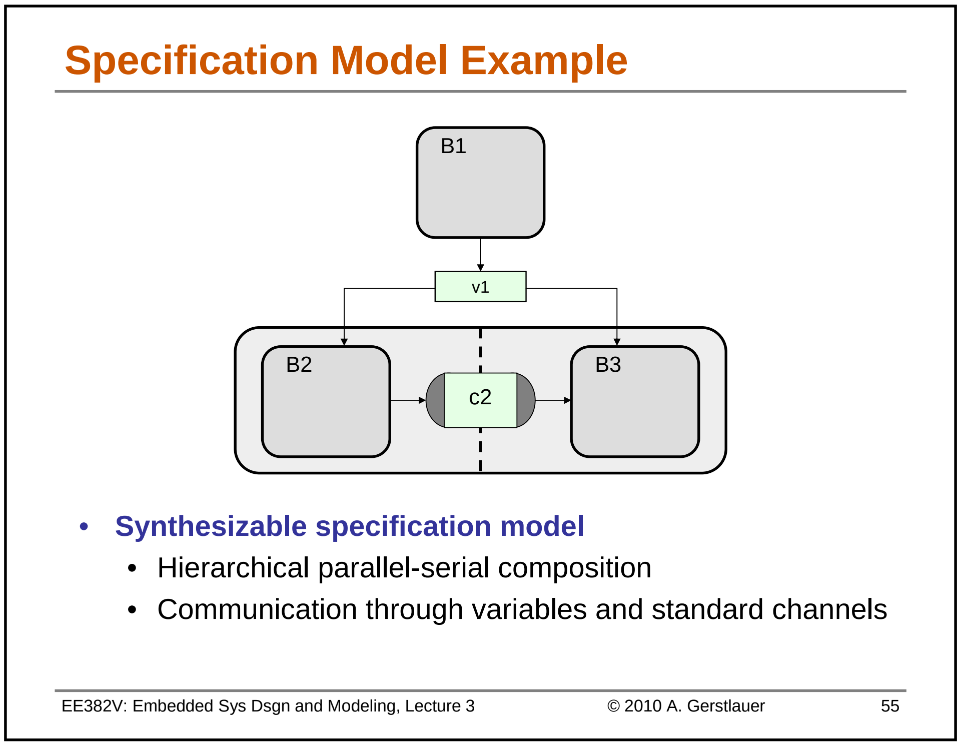
The SpecC Methodology



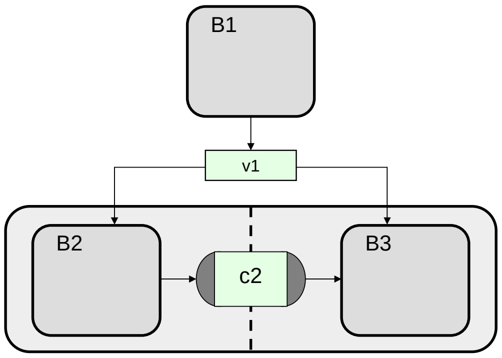
Specification Model

- **High-level, abstract model**
  - Pure system functionality
  - Algorithmic behavior
  - No implementation details
- **No implicit structure / architecture**
  - Behavioral hierarchy
- **Untimed**
  - Executes in zero (logical) time
  - Causal ordering
  - Events only for synchronization



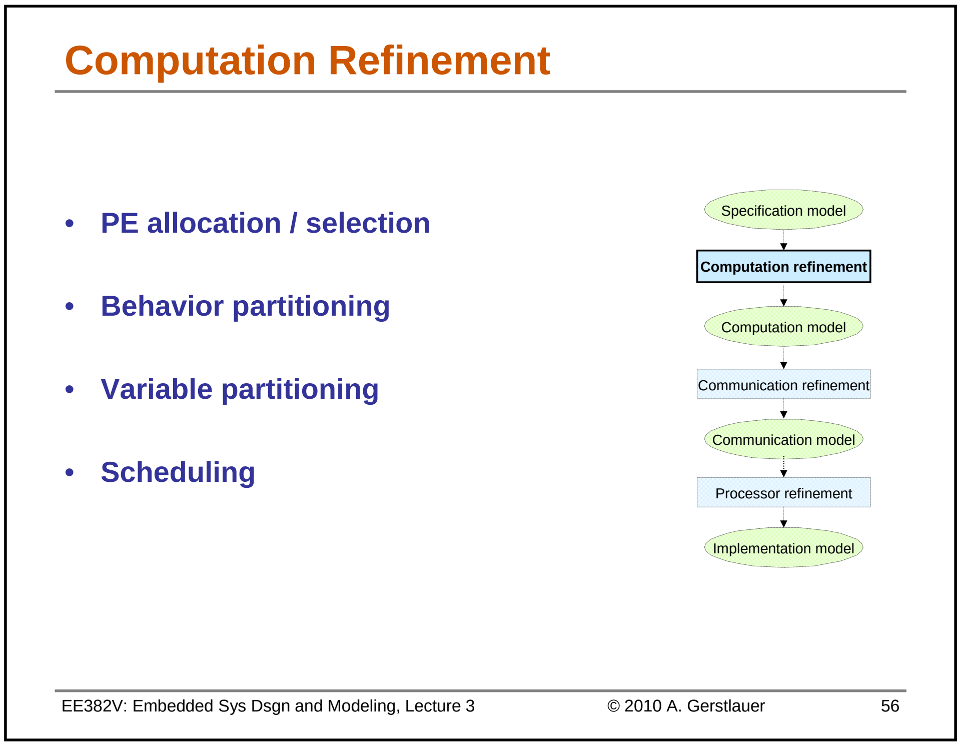


Specification Model Example



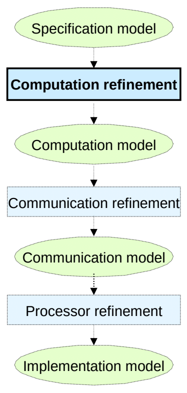
- **Synthesizable specification model**
  - Hierarchical parallel-serial composition
  - Communication through variables and standard channels

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 55



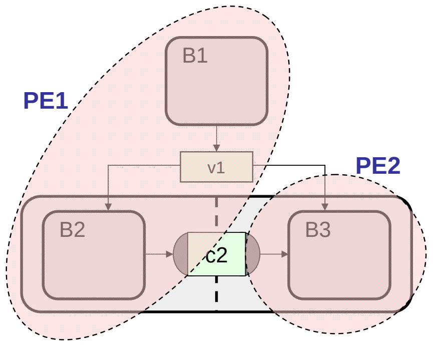
Computation Refinement

- **PE allocation / selection**
- **Behavior partitioning**
- **Variable partitioning**
- **Scheduling**



EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 56

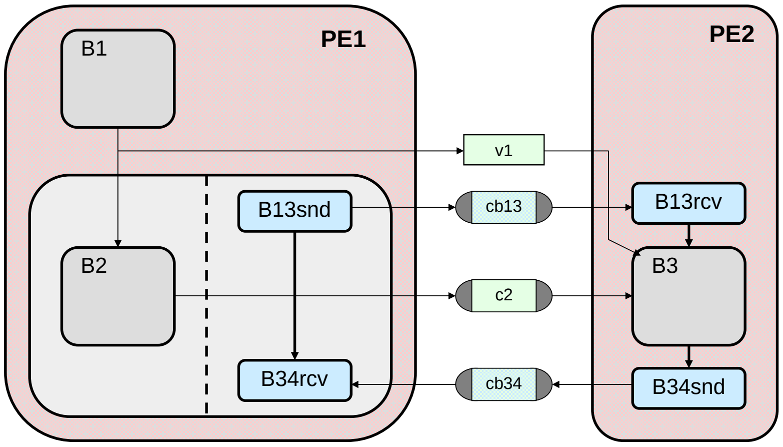
PE Allocation, Behavior Partitioning



- Allocate PEs
- Partition behaviors
- Globalize communication

➢ **Additional level of hierarchy to model PE structure**

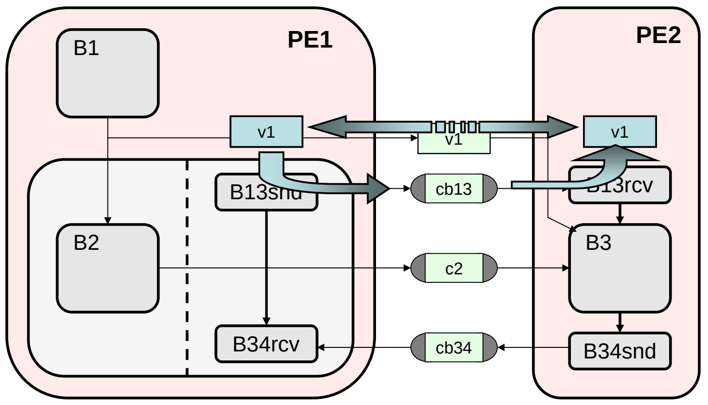
Model after Behavior Partitioning



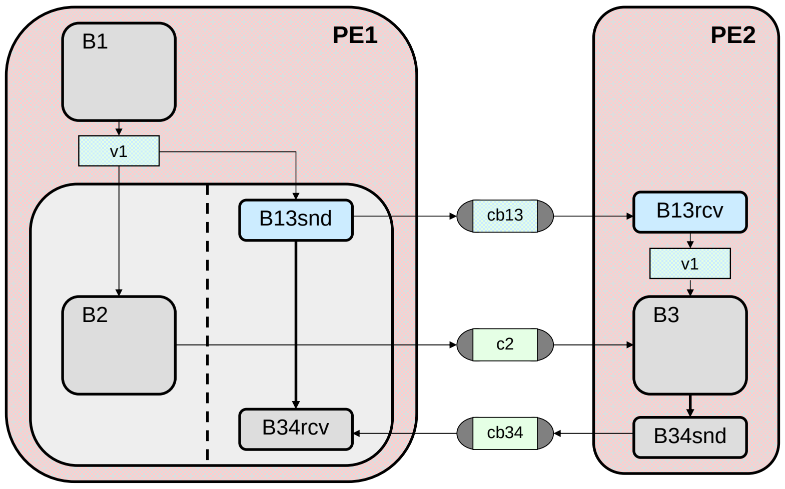
➢ **Synchronization to preserve execution order/semantics**

Variable Partitioning

- **Shared memory vs. message passing implementation**
  - Map global variables to local memories
  - Communicate data over message-passing channels



Model after Variable Partitioning



- **Keep local variable copies in sync**
  - Communicate updated values at synchronization points
  - Transfer control & data over message-passing channel

Timed Computation

- **Execution time of behaviors**
  - Estimated target delay / timing budget
- **Granularity**
  - Behavior / function / basic-block level

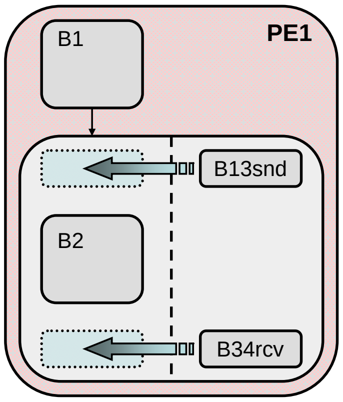
➢ **Annotate behaviors**

- Simulation feedback
- Synthesis constraints

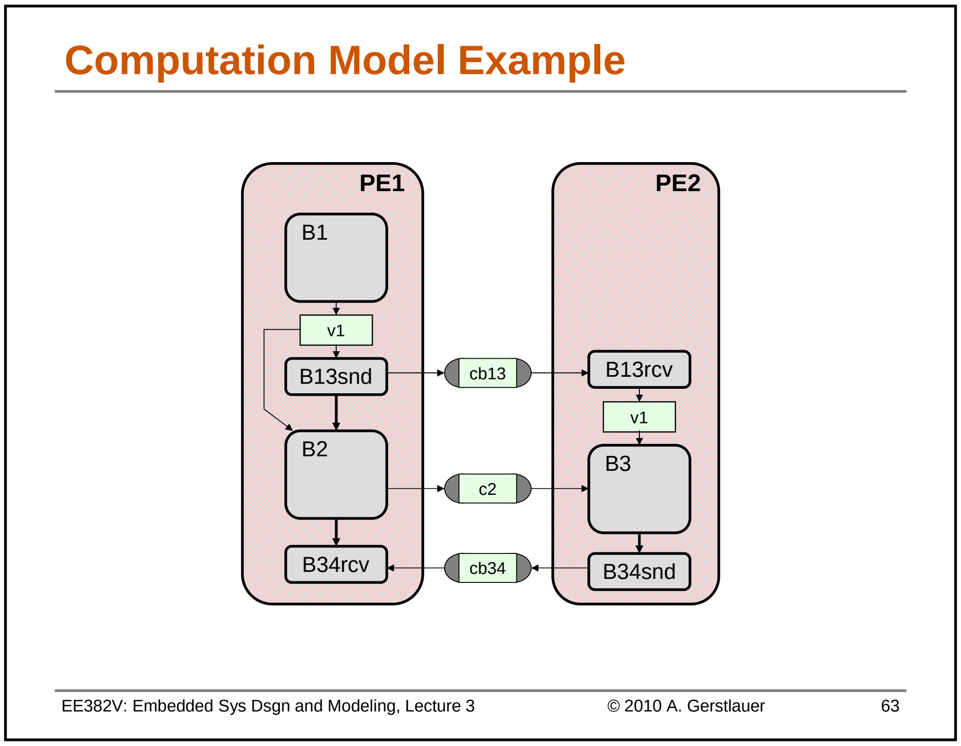
```
behavior B2( in int v1, ISend c2 )
{
  void main(void) {
    …
    waitfor( B2_DELAY1 );
    c2.send( … );
    …
    waitfor( B2_DELAY2 );
  }
};
```

Scheduling

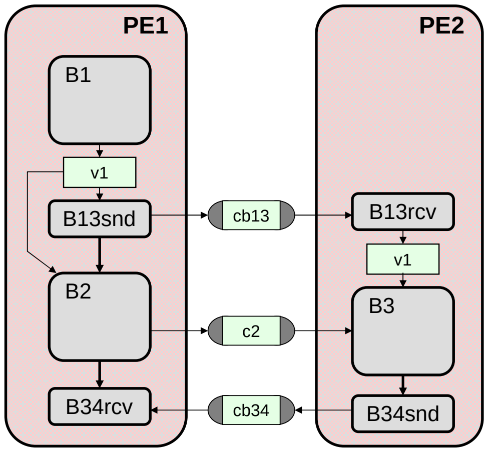
➢ **Serialize behavior execution on components**



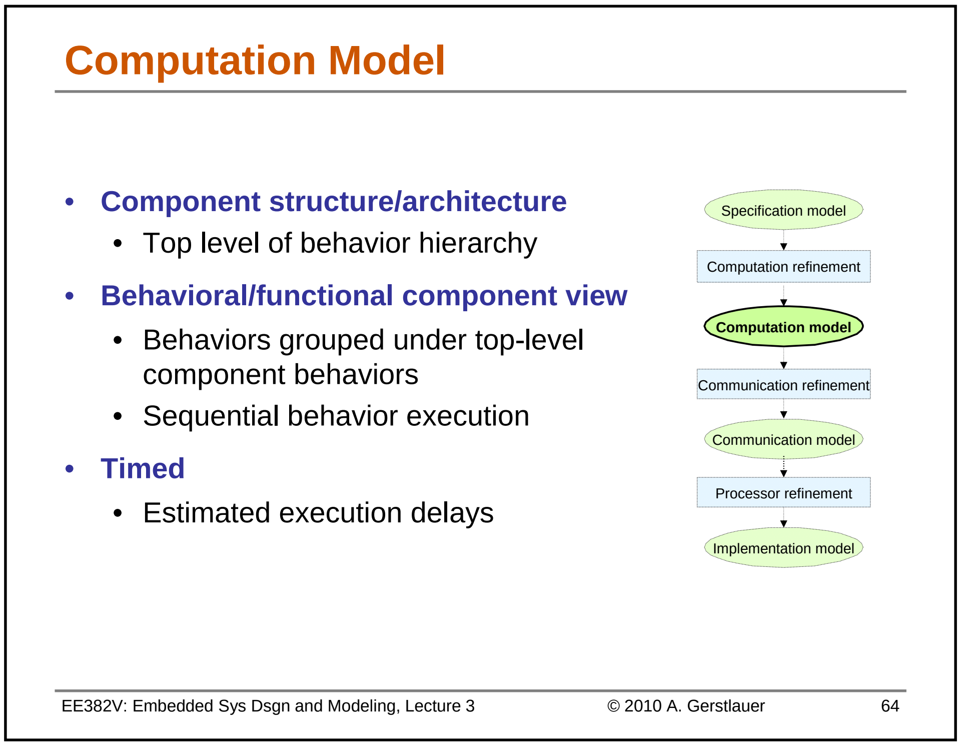
- Static scheduling
  - Fixed behavior execution order
  - Flattened behavior hierarchy
- Dynamic scheduling
  - Pool of tasks
  - Scheduler, abstracted OS



Computation Model Example

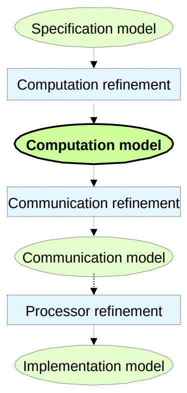


EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 63



Computation Model

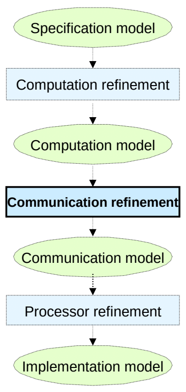
- **Component structure/architecture**
  - Top level of behavior hierarchy
- **Behavioral/functional component view**
  - Behaviors grouped under top-level component behaviors
  - Sequential behavior execution
- **Timed**
  - Estimated execution delays



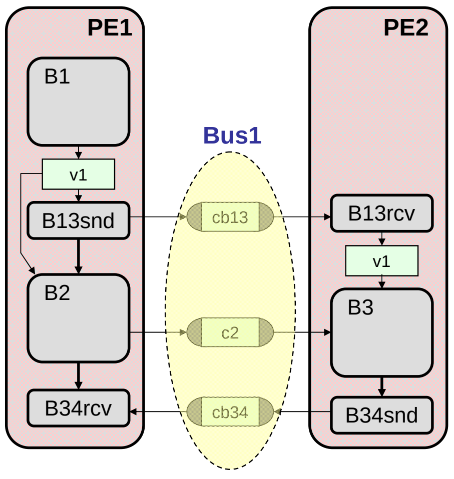
EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 64

Communication Refinement

- Network allocation / protocol selection
- Channel partitioning
- Protocol stack insertion
- Inlining



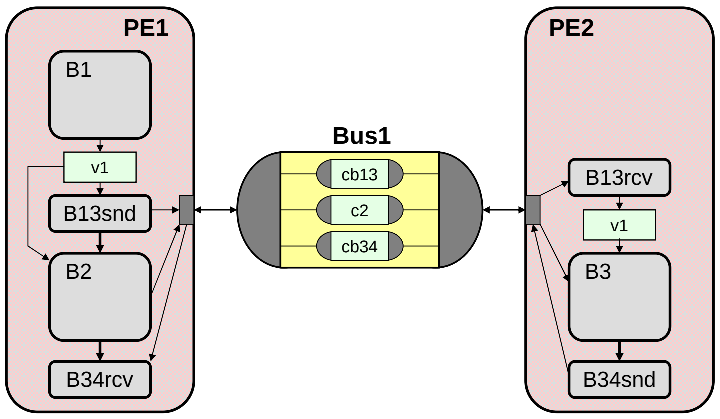
Network Allocation / Channel Partitioning



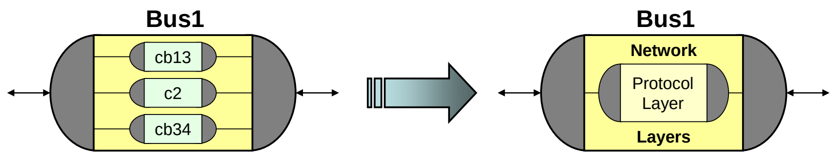
- Allocate busses
- Partition channels
- Update communication

➢ Additional level of hierarchy to model bus structure

Model after Channel Partitioning

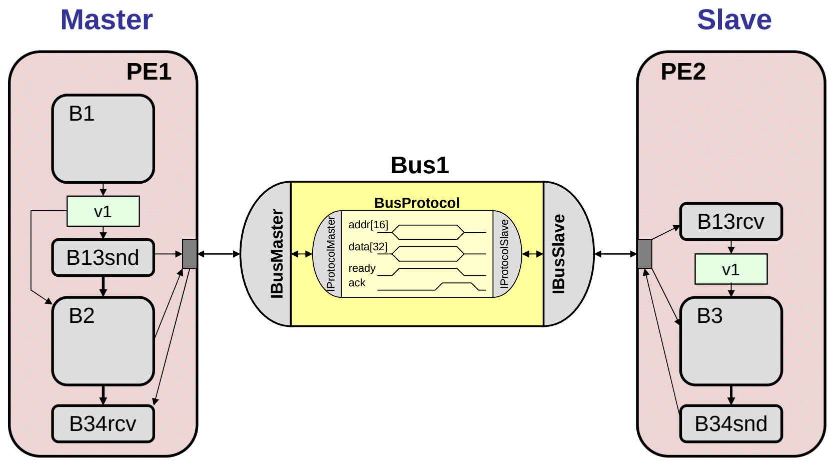


Protocol Insertion

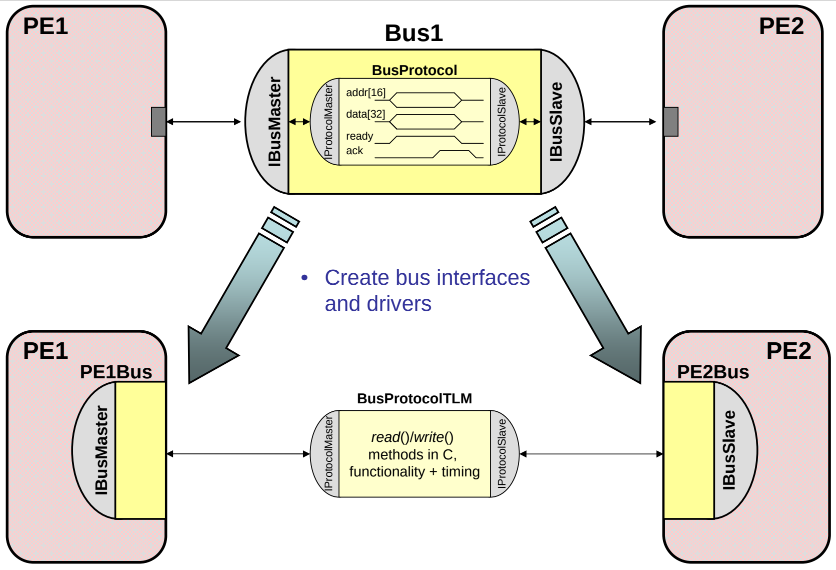


- **Insert protocol layer**
  - Bus protocol channel from database
- **Create network layers**
  - Implement message-passing over bus protocol
- **Replace bus channel**
  - Hierarchical combination of complete protocol stack

Model after Protocol Insertion

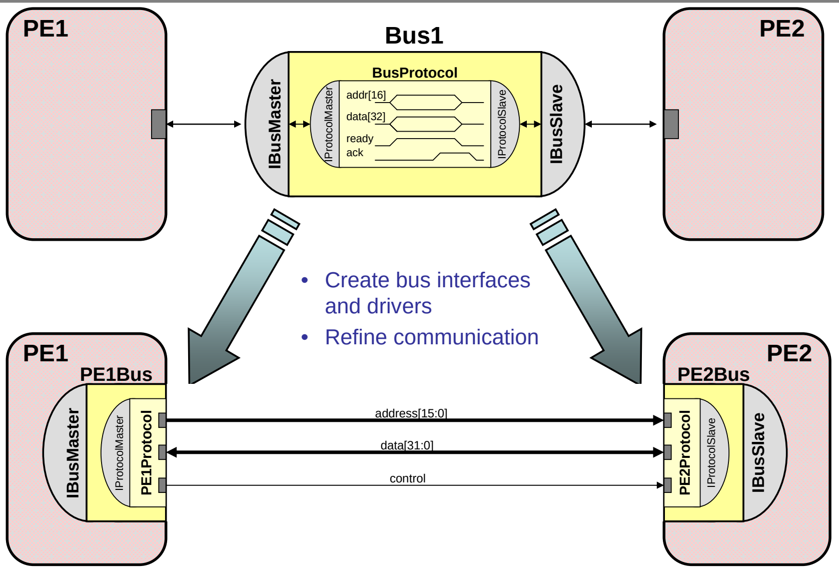


Inlining: Transaction-Level Model (TLM)



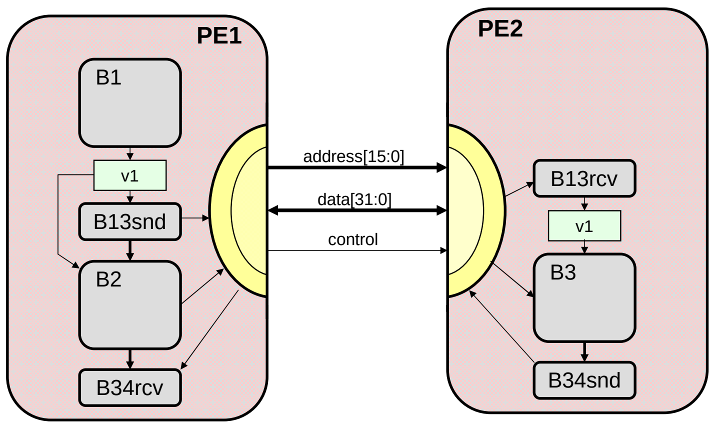
- Create bus interfaces and drivers

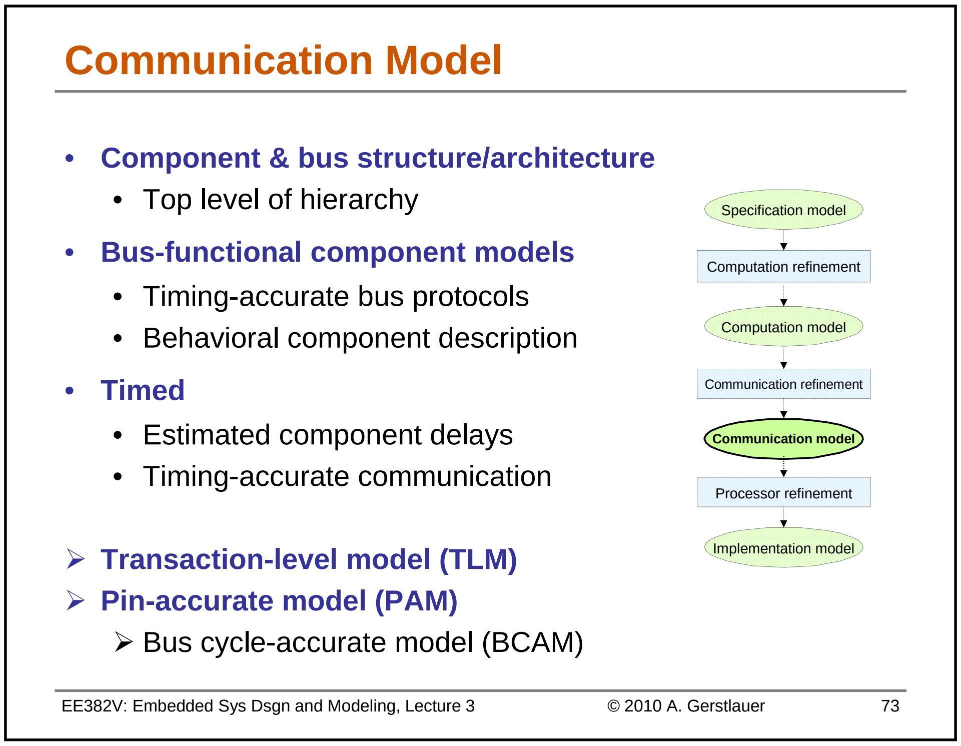
Inlining: Pin-Accurate Model (PAM)



- Create bus interfaces and drivers
- Refine communication

Communication Model Example





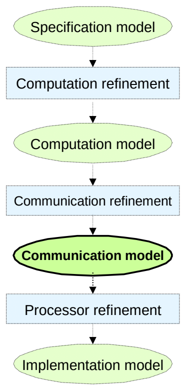
Communication Model

- **Component & bus structure/architecture**
  - Top level of hierarchy
- **Bus-functional component models**
  - Timing-accurate bus protocols
  - Behavioral component description
- **Timed**
  - Estimated component delays
  - Timing-accurate communication

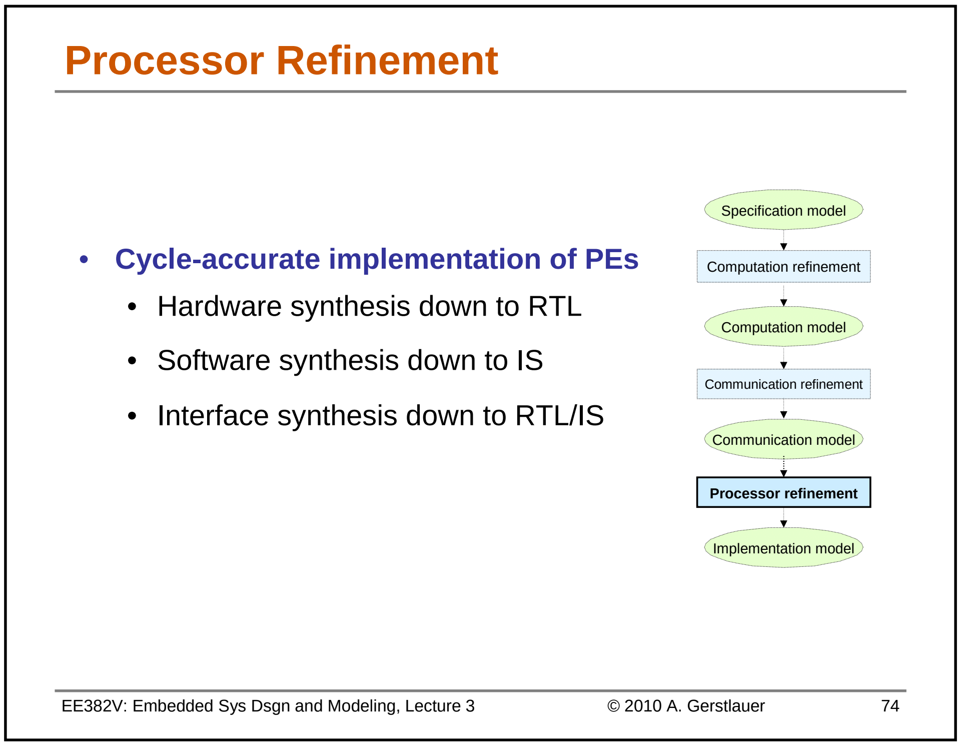
➢ **Transaction-level model (TLM)**

➢ **Pin-accurate model (PAM)**

  ➢ Bus cycle-accurate model (BCAM)

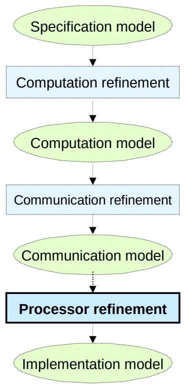


EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 73



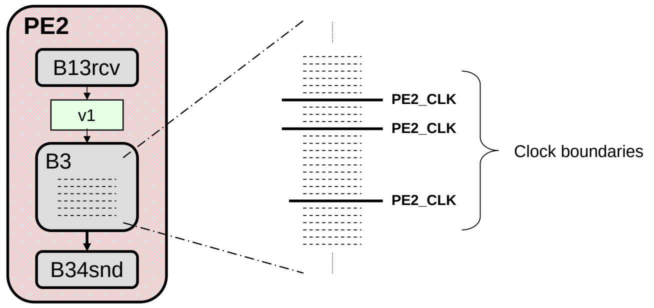
Processor Refinement

- **Cycle-accurate implementation of PEs**
  - Hardware synthesis down to RTL
  - Software synthesis down to IS
  - Interface synthesis down to RTL/IS



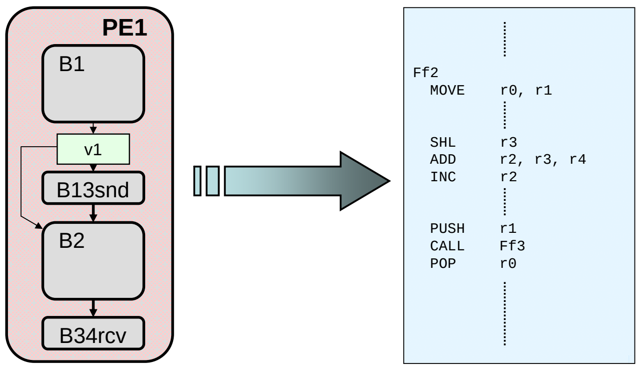
EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 74

Hardware Synthesis



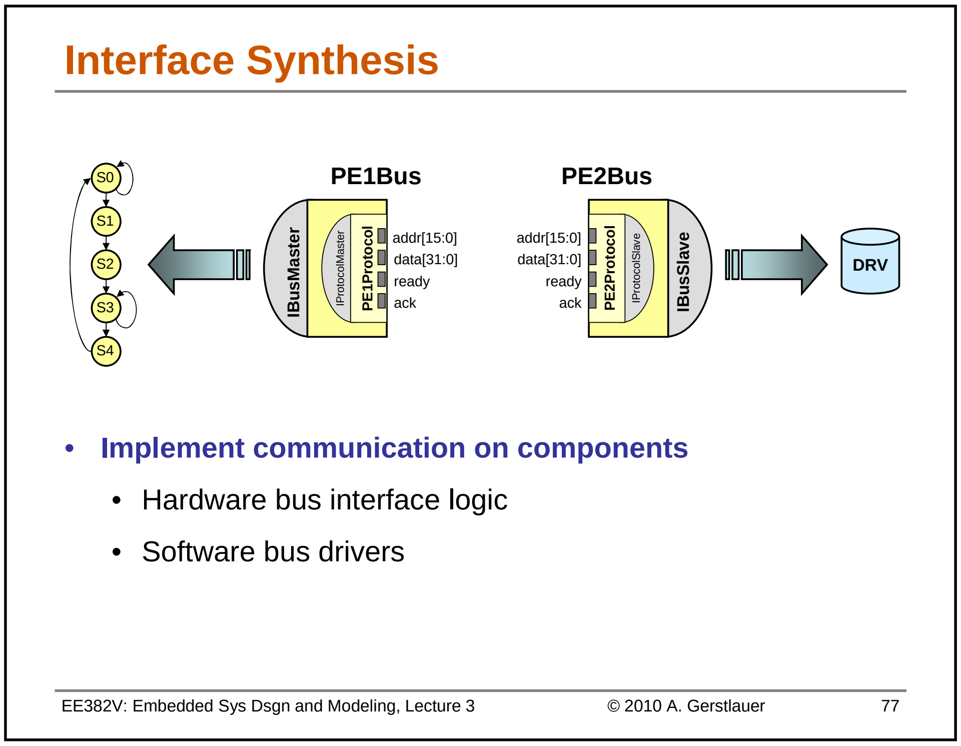
- **Schedule operations into clock cycles**
  - Define clock boundaries in leaf behavior C code
  - Create FSMD model from scheduled C code
    - Controller + datapath

Software Synthesis

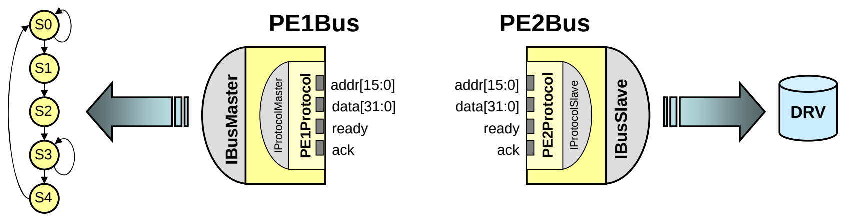


```
Ff2
  MOVE    r0, r1

  SHL     r3
  ADD     r2, r3, r4
  INC     r2

  PUSH    r1
  CALL    Ff3
  POP     r0
```

- **Implement behavior on processor instruction-set**
  - Code generation
  - Compilation

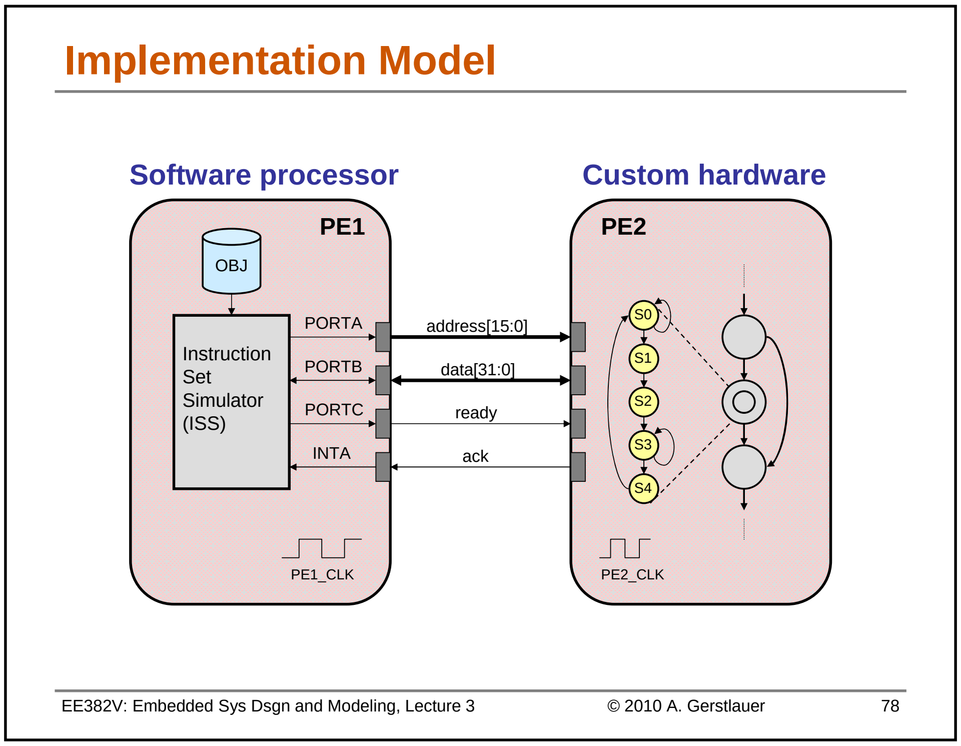


Interface Synthesis

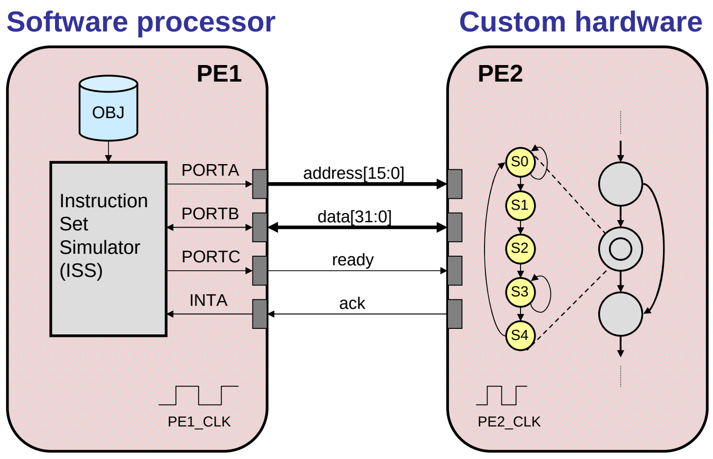


- **Implement communication on components**
  - Hardware bus interface logic
  - Software bus drivers

EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 77



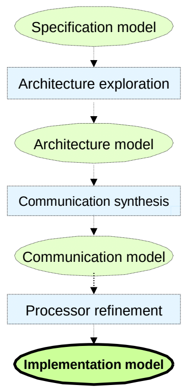
Implementation Model



EE382V: Embedded Sys Dsgn and Modeling, Lecture 3 © 2010 A. Gerstlauer 78

Implementation Model

- **Cycle-accurate system description**
  - RTL description of hardware
    - Behavioral/structural FSMD view
  - Object code for processors
    - Instruction-set co-simulation
  - Clocked bus communication
    - Bus interface timing based on PE clock



Lecture 3: Summary

- **SpecC language**
  - True superset of ANSI-C
    - ANSI-C plus extensions for HW-design
  - Support of all concepts needed in system design
    - Orthogonal, executable, synthesizable
  - Standardization and adoption
    - SpecC Technology Open Consortium (STOC), industry & academia
- **SpecC tools**
  - Compilation, validation, simulation
- **SpecC methodology**
  - Four levels of abstraction
    - Specification, Architecture, Communication, Implementation
  - Three refinement steps
    - Computation, communication and backend synthesis