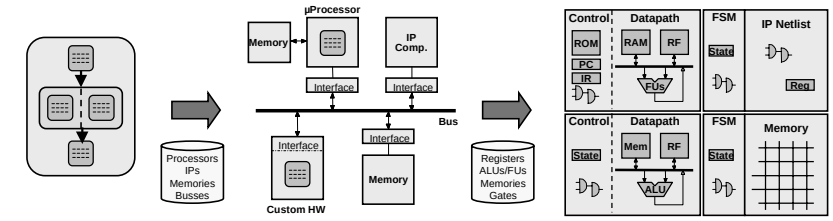
The SpecC Methodology

Andreas Gerstlauer
Center for Embedded Computer Systems
University of California, Irvine
http://www.cecs.uci.edu/~SpecC/

Outline

- **System design**
- **SpecC design methodology**
- **Specification model**
- **Architecture model**
- **Communication model**
- **Implementation model**
- **Summary & Conclusions**

System-On-Chip Design

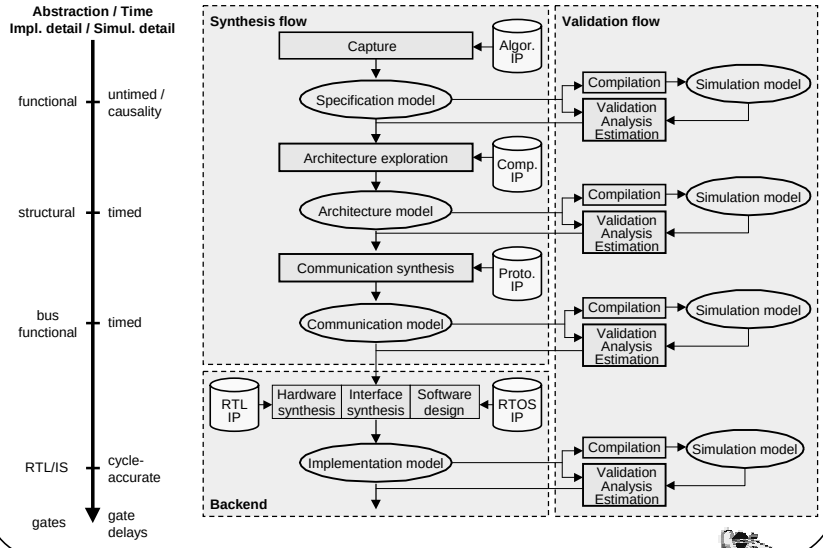


Specification + constraints | **System architecture + estimates** | **RTL Implementation + results**

- Specification to architecture to implementation
- Behavior to structure
  1. System level: system specification to system architecture
  2. RT level: component behavior to component microarchitecture

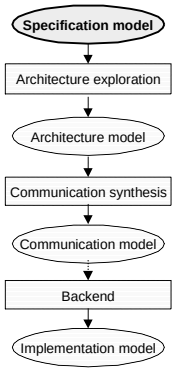
SpecC Methodology



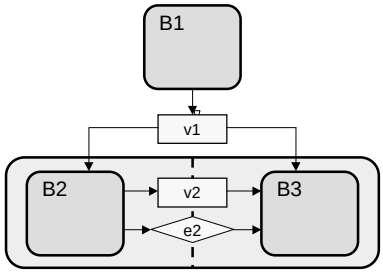
  

Specification Model

- **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



Leaf behaviors:

```
behavior B1( out int v1) {
  void main(void) {
     …
     v1 = …
};

behavior B2( in  int   v1,
             out int   v2,
             out event e2) {
   void main(void) {
      …
      v2 = f2( v1, … );
      notify( v2 );
      …
    }
};

behavior B3( in int   v1,
             in int   v2,
             in event e2) {
    void main(void) {
       …
       wait( e2 );
       f3( v1, v2, …);
       …
    }
};
```

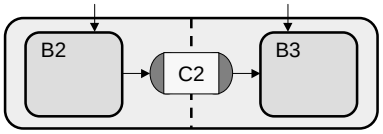
Design hierarchy:

```
behavior Design() {
  int v1;
  B1   b1  ( v1 );
  B2B3 b2b3( v1 );

  void main(void) {
     b1.main();
     b2b3.main();
  }
};
```

```
behavior B2B3( in int v1 ) {
   int   v2;
   event e2;
   B2 b2( v1, v2, e2 );
   B3 b2( v1, v2, e2 );

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


Specification Model Example (2)

- **Message-passing communication**
  - Abstract communication
  - Encapsulate communication



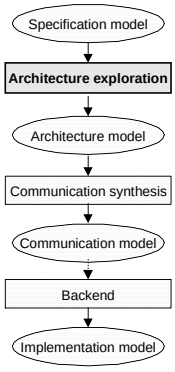
Blocking, unbuffered message-passing channel:

```
interface Isend {
  void send( void *d, int size );
};
interface Irecv {
  void recv( void *d, int size );
};

channel ChMP() implements ISend, IRecv {
  void send( void *d, int size ) { … }
  void recv( void *d, int size ) { … }
};
```

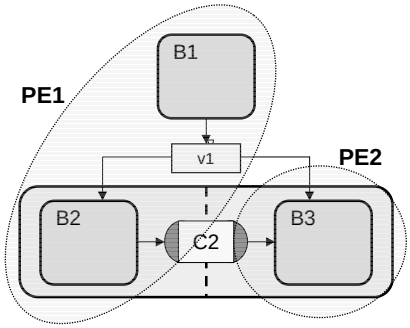
```
behavior B2( in int v1,
             ISend  c2) {
  void main(void) {
    …
    v2 = f2( v1, … );
    c2.send( &v2, sizeof(v2) );
    …
  }
};

behavior B3( in int v1,
             IRecv  c2) {
  void main(void) {
    …
    c2.recv( &v2, sizeof(v2) );
    f3( v1, v2, …);
    …
  }
};

behavior B2B3( in int v1 ) {
   ChMP c2();
   B2   b2( v1, c2 );
   B3   b3( v1, c2 );
   void main(void) {
     par {
       b2.main(); b3.main();
     }
}};
```


Architecture Exploration

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



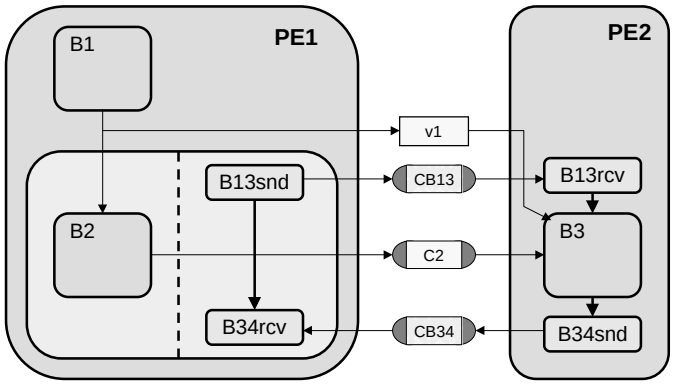
Allocation, Behavior Partitioning



- Allocate PEs
- Partition behaviors
- Globalize communication

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

Model after Behavior Partitioning



Synchronization behaviors:

```
behavior BSnd( ISend ch ) {
  void main(void) {
    ch.send( 0, 0 );
}};
```

```
behavior BRcv( IRecv ch ) {
  void main(void) {
    ch.recv( 0, 0 );
}};
```

Model after Behavior Partitioning (2)

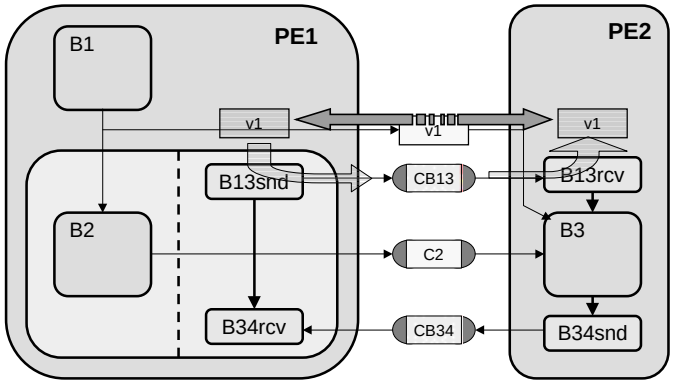
```
behavior PE1( int v1,
              ISend cb13,
              ISend c2,
              IRecv cb34) {
  B1   b1  ( v1 );
  B2B3 b2b3( v1, cb13, c2, cb34 );

  void main(void) {
    b1.main();
    b2b3.main();
}};
behavior B2B3( in int v1,
               ISend cb13,
               ISend c2,
               IRecv cb34) {
  B2     b2    ( v1, c2 );
  B3stub b3stub( cb13, cb34 );

  void main(void) {
    par {
      b2.main(); b3stub.main();
}}};
behavior B3stub( ISend cb13,
                 IRecv cb34 ) {
  BSnd b13snd( cb13 );
  BRcv b34rcv( cb34 );

  void main(void) {
    b13snd.main();
    b34rcv.main();
}};
```

```
behavior PE2( in int v1,
              IRecv cb13,
              IRecv c2,
              ISend cb34) {
  BRcv b13rcv( cb13 );
  B3   b3    ( v1, c2 );
  BSnd b34snd( cb34 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
}};
```

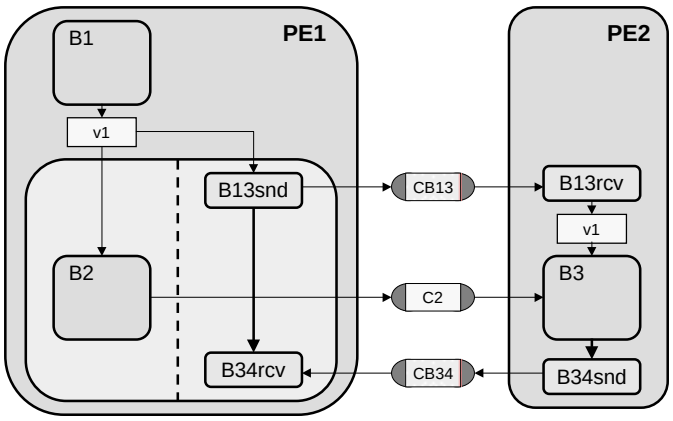
```
behavior Design() {
  int v1;
  ChMP cb13(), c2(), cb34();

  PE1 pe1( v1, cb13, c2, cb34 );
  PE2 pe2( v1, cb13, c2, cb34 );

  void main(void) {
    par {
      pe1.main();
      pe2.main();
    }
  }
};
```


Variable Partitioning

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



Message-Passing Model




```
behavior B13Snd( in int v1, ISend ch ) {
  void main(void) {
    ch.send( &v1, sizeof(v1) );
}};
```

```
behavior B13Rcv( IRecv ch, out int v1 ) {
  void main(void) {
    ch.recv( &v1, sizeof(v1) );
}};
```


Message-Passing Model (2)

```
behavior PE1( ISend cb13,
              ISend c2,
              IRecv cb34) {
  int v1;
  B1   b1  ( v1 );
  B2B3 b2b3( v1, cb13, c2, cb34 );

  void main(void) {
    b1.main();
    b2b3.main();
}};
behavior B2B3( in int v1,
               ISend cb13,
               ISend c2,
               IRecv cb34) {
  B2     b2    ( v1, c2 );
  B3stub b3stub( v1, cb13, cb34 );

  void main(void) {
    par {
      b2.main(); b3stub.main();
}}};
behavior B3stub( in int v1,
                 ISend cb13,
                 IRecv cb34 ) {
  B13Snd b13snd( v1, cb13 );
  BRcv   b34rcv( cb34 );

  void main(void) {
    b13snd.main();
    b34rcv.main();
}};
```

```
behavior PE2( IRecv cb13,
              IRecv c2,
              ISend cb34) {
  int v1;
  B13Rcv b13rcv( cb13, v1 );
  B3     b3    ( v1, c2 );
  BSnd   b34snd( cb34 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
}};
```

```
behavior Design() {
  ChMP cb13(), c2(), cb34();

  PE1 pe1( cb13, c2, cb34 );
  PE2 pe2( cb13, c2, cb34 );

  void main(void) {
    par {
      pe1.main();
      pe2.main();
    }
  }
};
```

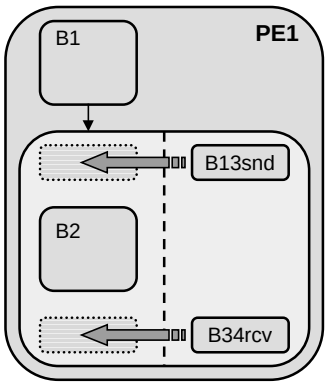
Estimated Execution Time

- **Estimate execution time**
  - Target execution delay
  - Timing budget
- **Annotate leaf behaviors**
  - Logical simulation time
  - Synthesis constraints
- **Granularity**
  - Behavior level
  - Basic block level

```
behavior B2( in int v1,
             ISend  c2)
{
  void main(void) {
    ...
    waitfor( delay );
    if( … ) {
      ...
      waitfor( delay );
    }
    else {
      ...
      waitfor( delay );
    }
    ...
    waitfor( delay );
    c2.send( … );
    ...
    waitfor( delay );
    while( …) {
      ...
      waitfor( delay );
    }
    ...
    waitfor( delay );
  }
};
```


Scheduling

- **Serialize behavior execution on components**

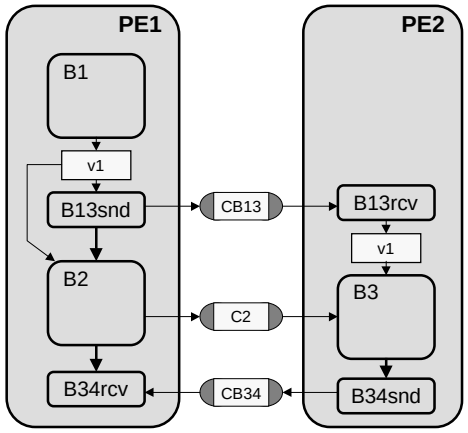


- Static scheduling
  - Fixed behavior execution order
  - Flattened behavior hierarchy
- Dynamic scheduling
  - Pool of tasks
  - Scheduler, abstracted OS

Model after Scheduling

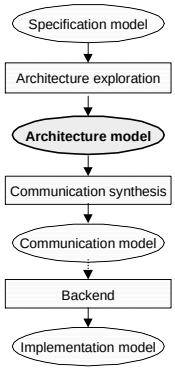
Statically scheduled PE1:

```
behavior PE1( ISend cb13,
              ISend c2,
              IRecv cb34)
{
  int v1;

  B1     b1    ( v1 );
  B13Snd b13snd( v1, cb13);
  B2     b2    ( v1, c2 );
  BRcv   b34rcv( cb34 );

  void main(void)
  {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};
```



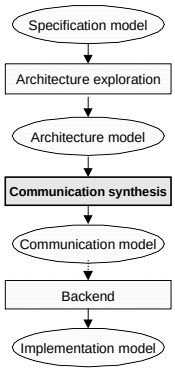
Architecture Model

- **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

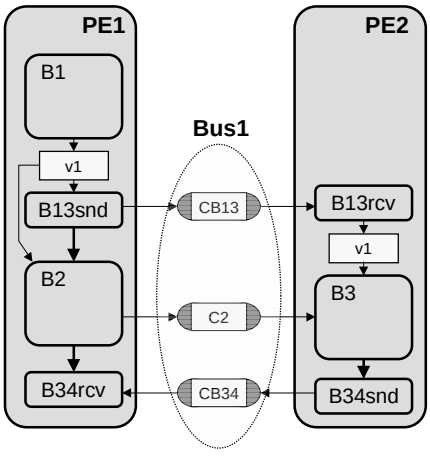


Communication Synthesis

- **Bus allocation / protocol selection**
- **Channel partitioning**
- **Protocol, transducer insertion**
- **Inlining**



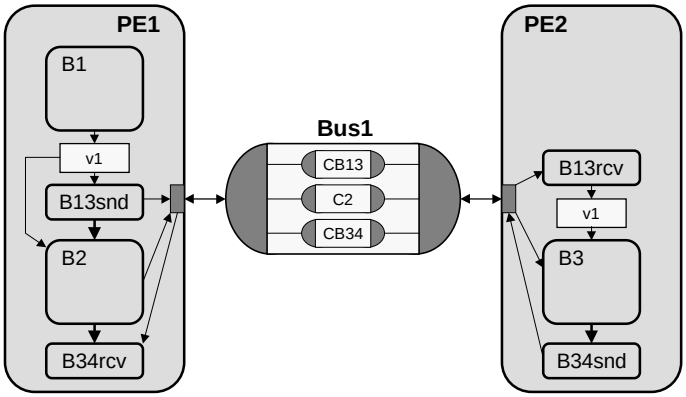
Bus Allocation / Channel Partitioning



- Allocate busses
- Partition channels
- Update communication

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

Model after Channel Partitioning



Model after Channel Partitioning (2)

Bus channel:

```
typedef enum { CB13, C2, CB34 } BusAddr;

interface IBus {
  void send( BusAddr addr, void* d, int size );
  void recv( BusAddr addr, void* d, int size );
};

channel Bus1() implements IBus
{
  ChMP cb13(), c2(), cb34();

  void send( BusAddr addr, void* d, int size )
  {
    switch( addr ) {
      case CB13: return cb13.send( d, size );
      case C2:   return c2.send( d, size );
      case CB34: return cb34.send( d, size );
    }
  }
  void recv( BusAddr addr, void* d, int size )
  {
    switch( addr ) {
      case CB13: return cb13.recv( d, size );
      case C2:   return c2.recv( d, size );
      case CB34: return cb34.recv( d, size );
    }
  }
};
```

Leaf behaviors:

```
behavior B2( in int v1, IBus bus ) {
  void main(void) {
    …
    bus.send( C2, … );
    …
}};
behavior B3( in int v1, IBus bus ) {
  void main(void) {
    …
    bus.recv( C2, …);
    …
}};
behavior B13Snd( in int v1, IBus bus ) {
  void main(void) {
    bus.send( CB13, &v1, sizeof(v1) );
}};
behavior B13Rcv( IBus bus, out int v1 ) {
  void main(void) {
    bus.recv( CB13, &v1, sizeof(v1) );
}};
behavior B34Snd( IBus bus ) {
  void main(void) {
    bus.send( CB34, 0, 0 );
}};
behavior B13Rcv( IBus bus ) {
  void main(void) {
    bus.recv( CB34, 0, 0 );
}};
```

Model after Channel Partitioning (3)

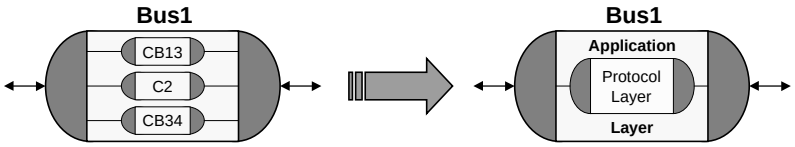
Components:

```
behavior PE1( IBus bus1)
{
  int v1;
  B1     b1    ( v1 );
  B13Snd b13snd( v1, bus1);
  B2     b2    ( v1, bus1 );
  B34Rcv b34rcv( bus1 );

  void main(void) {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};

behavior PE2( IBus bus1)
{
  int v1;
  B13Rcv b13rcv( bus1, v1 );
  B3     b3    ( v1, bus1 );
  B34Snd b34snd( bus1 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```

Design top level:

```
behavior Design()
{
  Bus1 bus1();

  PE1 pe1( bus1 );
  PE2 pe2( bus1 );

  void main(void) {
    par {
      pe1.main();
      pe2.main();
    }
  }
};
```

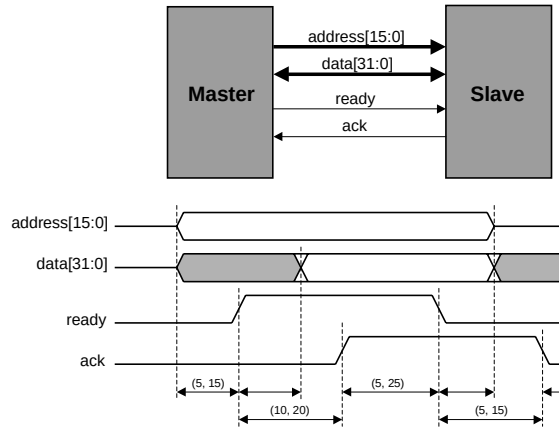
Protocol Insertion



- **Insert protocol layer**
  - Protocol channel
- **Create application layer**
  - Implement message-passing over bus protocol
- **Replace bus channel**
  - Hierarchical bus channel

Protocol Example

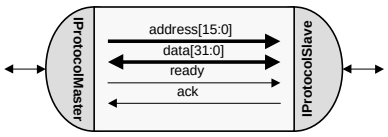
- **Double handshake protocol**



Protocol Layer

- **Protocol channel**
  - Encapsulate wires
  - Abstract bus transfers
  - Implement protocol
  - Bus timing

DoubleHandshakeProtocol




```
interface IProtocolMaster {
  void masterWrite( bit[15:0] a, bit[31:0] d );
  void masterRead( bit[15:0] a, bit[31:0] *d );
};
interface IProtocolSlave {
  void slaveWrite( bit[15:0] a, bit[31:0] d );
  void slaveRead( bit[15:0] a, bit[31:0] *d );
};

channel DoubleHandshakeProtocol()
  implements IProtocolMaster, IProtocolSlave
{
  bit[15:0] address;
  bit[31:0] data;
  Signal    ready();
  Signal    ack();

  …
};
```

Double Handshake Protocol Layer

Master Interface:

```
void masterWrite( bit[15:0] a, bit[31:0] d ) {
  do {
    t1: address = a;
        data    = d;
        waitfor( 5 );    // estimated delay
    t2: ready.set( 1 );
        ack.waituntil( 1 );
    t3: waitfor( 10 );   // estimated delay
    t4: ready.set( 0 );
        ack.waituntil( 0 );
  } timing {             // timing constraints
    range( t1; t2; 5; 15 );
    range( t3; t4; 10; 25 );
}}
void masterRead( bit[15:0] a, bit[31:0] *d ) {
  do {
    t1: address = a;
        waitfor( 5 );    // estimated delay
    t2: ready.set( 1 );
        ack.waituntil( 1 );
    t3: *d = data;
        waitfor( 15 );   // estimated delay
    t4: ready.set( 0 );
        ack.waituntil( 0 );
  } timing {             // timing constraints
    range( t1; t2; 5; 15 );
    range( t3; t4; 10; 25 );
}}
```

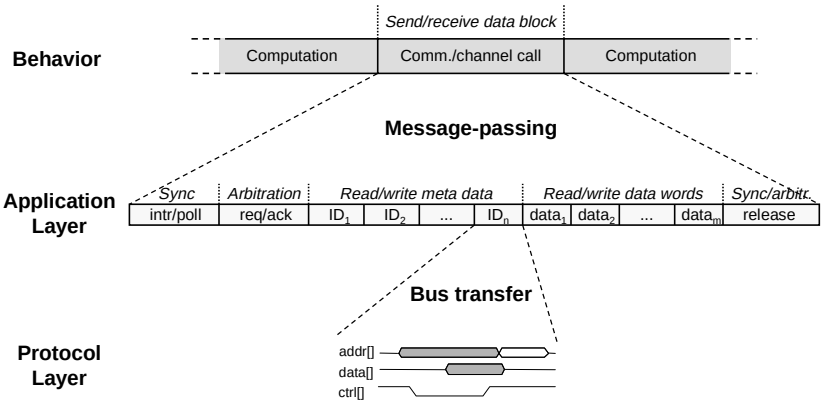
Slave Interface:

```
void slaveWrite( bit[15:0] a, bit[31:0] d ) {
  do {
    t1: ready.waituntil( 1 );
    t2: if( a != address ) goto t1;
        data = d;
        waitfor( 12 );   // estimated delay
    t3: ack.set( 1 );
        ready.waituntil( 0 );
    t4: waitfor( 7 );    // estimated delay
    t5: ack.set( 0 );
  } timing {             // timing constraint
    range( t2; t3; 10; 20 );
    range( t4; t5; 5; 15 );
}}
void slaveRead( bit[15:0] a, bit[31:0] *d ) {
  do {
    t1: ready.waituntil( 1 );
    t2: if( a != address ) goto t1;
        *d = data;
        waitfor( 12 );   // estimated delay
    t3: ack.set( 1 );
        ready.waituntil( 0 );
    t4: waitfor( 7 );    // estimated delay
    t5: ack.set( 0 );
  } timing {             // timing constraint
    range( t2; t3; 10; 20 );
    range( t4; t5; 5; 15 );
}}
```

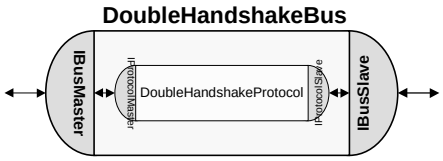
Application Layer

- **Implement abstract message-passing over protocol**



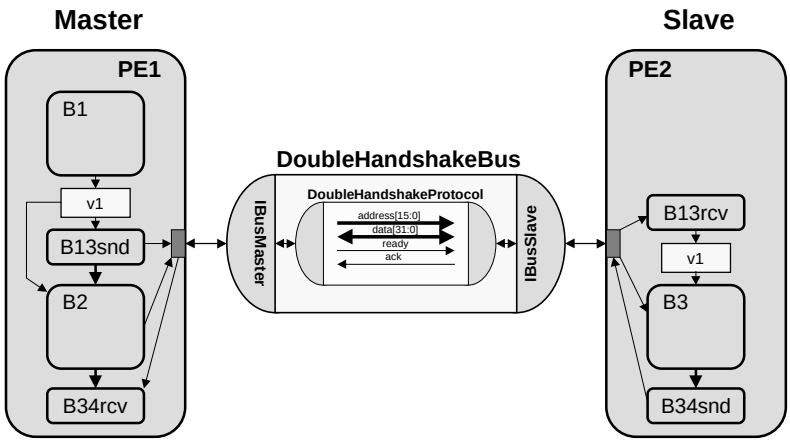
  

Example Application Layer




```
channel DoubleHandshakeBus() implements IBusMaster, IBusSlave {
  DoubleHandshakeProtocol protocol();

  void masterSend( int addr, void* d, int size ) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.masterWrite( addr, (long)*d );
  }}
  void masterRecv( int addr, void* d, int size ) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.masterRead( addr, (long*)d );
  }}

  void slaveSend( int addr, void* d, int size ) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.slaveWrite( addr, (long)*d );
  }}
  void slaveRecv( int addr, void* d, int size ) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.slaveRead( addr, (long*)d );
  }}
};
```


Model after Protocol Insertion



Model after Protocol Insertion (2)

Components:

```
behavior PE1( IBusMaster bus1)
{
  int v1;
  B1     b1    ( v1 );
  B13Snd b13snd( v1, bus1);
  B2     b2    ( v1, bus1 );
  B34Rcv b34rcv( bus1 );

  void main(void) {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};

behavior PE2( IBusSlave bus1)
{
  int v1;
  B13Rcv b13rcv( bus1, v1 );
  B3     b3    ( v1, bus1 );
  B34Snd b34snd( bus1 );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```

Top level:

```
behavior Design()
{
  DoubleHandshakeBus bus1();

  PE1 pe1( bus1 );
  PE2 pe2( bus1 );

  void main(void) {
    par {
      pe1.main();
      pe2.main();
    }
  }
};
```

Model after Protocol Insertion (3)

PE1 leaf behaviors:

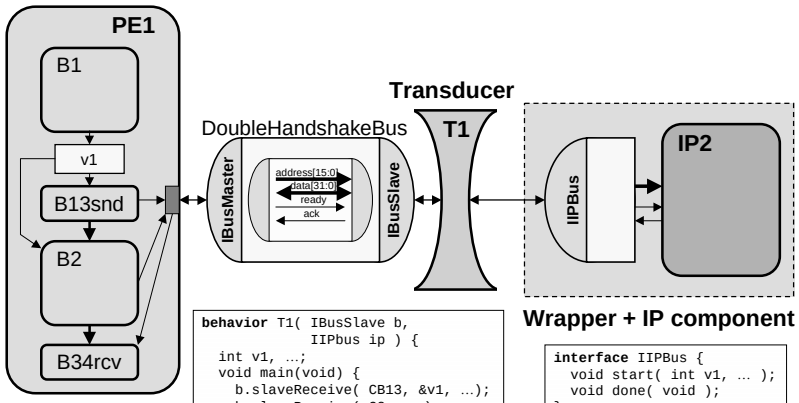
```
// PE1
behavior B2( in int v1, IBusMaster bus )
{
  void main(void) {
    …
    bus.masterSend( C2, … );
    …
  }
};

behavior B13Snd( in int v1, IBusMaster bus )
{
  void main(void) {
    bus.masterSend( CB13, &v1, sizeof(v1) );
  }
};

behavior B13Rcv( IBusMaster bus )
{
  void main(void) {
    bus.masterRecv( CB34, 0, 0 );
  }
};
```

PE2 leaf behaviors:

```
// PE2
behavior B3( in int v1, IBusSlave bus )
{
  void main(void) {
    …
    bus.slaveRecv( C2, …);
    …
  }
};

behavior B13Rcv( IBusSlave bus, out int v1 )
{
  void main(void) {
    bus.slaveRecv( CB13, &v1, sizeof(v1) );
  }
};

behavior B34Snd( IBusSlave bus )
{
  void main(void) {
    bus.slaveSend( CB34, 0, 0 );
  }
};
```

Transducer Insertion

- **IP component with fixed protocol**

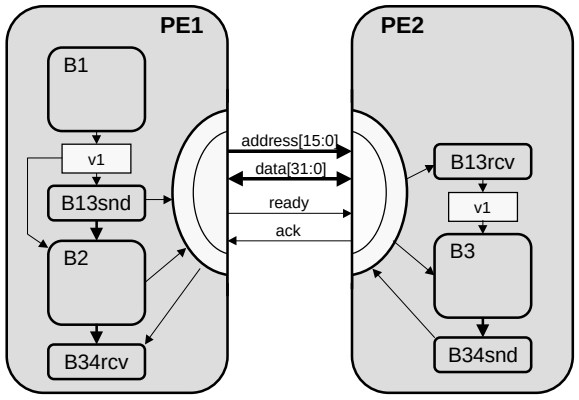



```
behavior T1( IBusSlave b,
             IIPbus ip ) {
  int v1, …;
  void main(void) {
    b.slaveReceive( CB13, &v1, …);
    b.slaveReceive( C2, … );
    ip.start( v1, … );
    ip.done();
    b.slaveSend( CB34, 0, 0 );
}};
```

```
interface IIPBus {
  void start( int v1, … );
  void done( void );
};
```


Inlining

- **Move communication functionality into components**




```
behavior Design()
{
  // wires
  bit[15:0] addr;
  bit[31:0] data;
  Signal ready(),
  Signal ack();

  PE1 pe1( addr,
           data,
           ready,
           ack );

  PE2 pe2( addr,
           data,
           ready,
           ack );

  void main(void)
  {
    par {
      pe1.main();
      pe2.main();
    }
  }
};
```

Model after Inlining (PE1)

```
channel PE1BusInterface( out bit[15:0] addr.
                             bit[31:0] data,
                             OSignal   ready,
                             ISignal   ack )
  implements IProtocolMaster
{
  void masterWrite( bit[15:0] a, bit[31:0] d ) {
    …
  }
  void masterRead( bit[15:0] a, bit[31:0] *d ) {
    …
}};

channel PE1BusDriver( out bit[15:0] addr.
                          bit[31:0] data,
                          OSignal   ready,
                          ISignal   ack )
  implements IBusMaster
{
  PE1BusInterface protocol( addr, data, ready, ack );

  void masterSend(int addr, void* d, int size) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.masterWrite( addr, (long)*d );
      waitfor( …);
  }}
  void masterReceive(int addr, void* d, int size) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.masterRead( addr, (long*)d );
      waitfor( …);
}}};
```

```
behavior PE1( out bit[15:0] addr.
                  bit[31:0] data,
                  OSignal   ready,
                  ISignal ack )
{
  PE1BusDriver bus( addr, data,
                    ready, ack );

  int v1;

  B1     b1    ( v1 );
  B13Snd b13snd( v1, bus );
  B2     b2    ( v1, bus );
  B34Rcv b34rcv( bus );

  void main(void) {
    b1.main();
    b13snd.main();
    b2.main();
    b34rcv.main();
  }
};
```

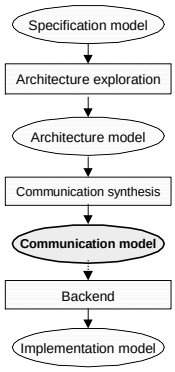

Model after Inlining (PE2)

```
channel PE2BusInterface( in bit[15:0] addr.
                            bit[31:0] data,
                            ISignal   ready,
                            OSignal   ack )
  implements IProtocolSlave
{
  void slaveWrite( bit[15:0] a, bit[31:0] d ) {
    …
  }
  void slaveRead( bit[15:0] a, bit[31:0] *d ) {
    …
}};

channel PE2BusDriver( in bit[15:0] addr.
                         bit[31:0] data,
                         ISignal   ready,
                         OSignal   ack )
  implements IBusSlave
{
  PE2BusInterface protocol( addr, data, ready, ack );

  void slaveSend(int addr, void* d, int size) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.slaveWrite( addr, (long)*d );
      waitfor( …);
  }}
  void slaveReceive(int addr, void* d, int size) {
    for( ; size > 0; size -= 4, ((long*)d)++ ) {
      protocol.slaveRead( addr, (long*)d );
      waitfor( …);
}}};
```

```
behavior PE2( in bit[15:0] addr.
                 bit[31:0] data,
                 ISignal   ready,
                 OSignal   ack )
{
  PE2BusDriver bus( addr, data,
                    ready, ack );

  int v1;

  B13Rcv b13rcv( bus, v1 );
  B3     b3    ( v1, bus );
  B34Snd b34snd( bus );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```

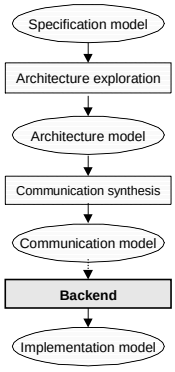
Communication Model

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

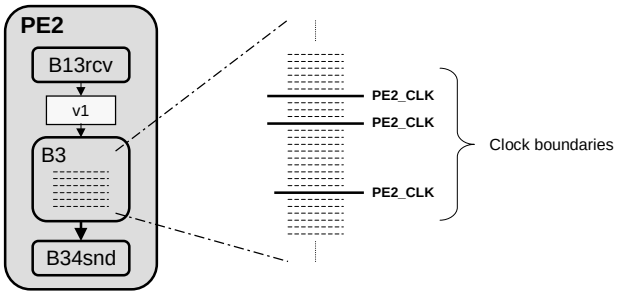


Backend

- **Clock-accurate implementation of PEs**
  - Hardware synthesis
  - Software development
  - Interface synthesis



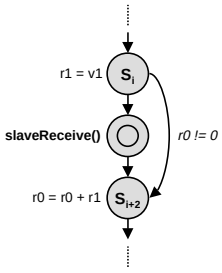
Hardware Synthesis



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

Scheduled Hardware Model

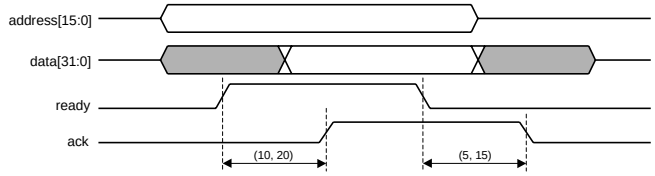
Hierarchical FSMD:



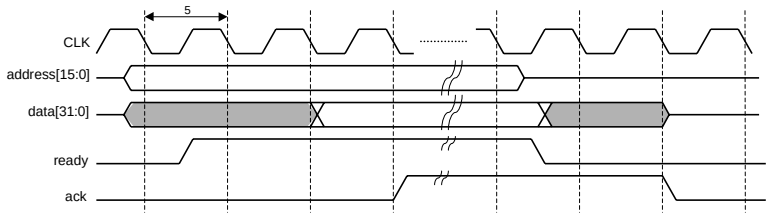

```
behavior B3( in int v1, IBusSlave bus )
{
  void main(void) {
    enum { S0, S1, S2, …, Sn } state = S0;

    while( state != Sn )
    {
      waitfor( PE2_CLK );  // wait for clock edge

      switch( state ) {
        …
        case Si:
          r1 = v1;  // memory read
          if( r0 )
            state = Si+1;
          else
            state = Sj+2;
          break;
        case Si+1:  // receive message
          bus.slaveReceive( C2, … );
          state = Si+2;
          break;
        case Sj+2:
          r0 += r1; // ALU operation
          state = Si+3;
          break;
        …
    }}
  }
};
```

Hardware Interface Synthesis

- **Specification: timing diagram / constraints**

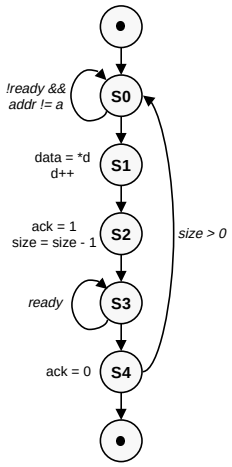


- **FSM implementation: clock cycles**

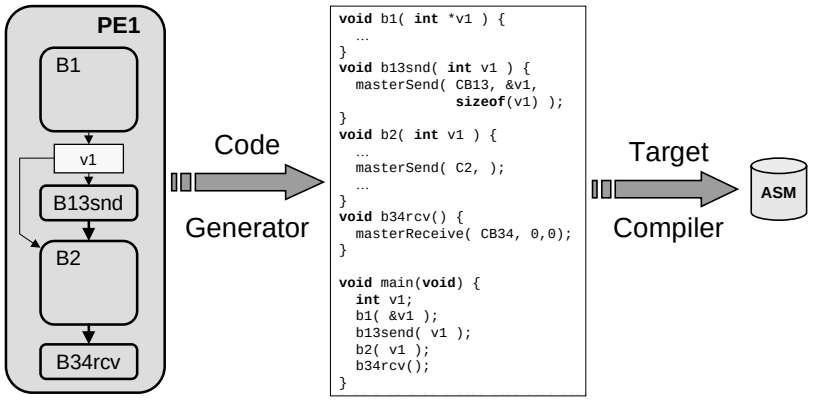


Hardware Interface FSM

```
channel HWBusDriver( in bit[15:0] addr, bit[31:0] data,
                     ISignal ready, OSignal ack )
implements IBusSlave
{
  void slaveSend( int a, void* d, int size ) {
    enum { S0, S1, S2, S3, S4, S5 } state = S0;
    while( state != S5 ) {
      waitfor( PE2_CLK );        // wait for clock edge
      switch( state ) {
          case S0:  // sample ready, address
            if( (ready.val() == 1) && (addr == a) ) state = S1;
            break;
          case S1:  // read memory, drive data bus
            data = *( ((long*)d)++ );
            state = S2;
            break;
          case S2: // raise ack, advance counter
            ack.set( 1 );
            size--;
            state = S3;
            break;
          case S3: // sample ready
            if( ready.val() == 0 ) state = S4;
            break;
          case S4: // reset ack, loop condition
            ack.set( 0 );
            if( size != 0 ) state = S0 else state = S5;
    }}
  }
  void slaveReceive( int a, void* d, int size ) { … }
};
```



Software Development



- Generate C code from SpecC component model
- Compile C code into assembly code for target processor

Software Interface Synthesis

- **Implement communication over processor bus**
  - Map bus wires to processor ports
    - address: PORTA, data: PORTB, ready: PORTC, ack: INTA
  - Generate assembly code
    - Implement bus protocol timing via processor's I/O instructions

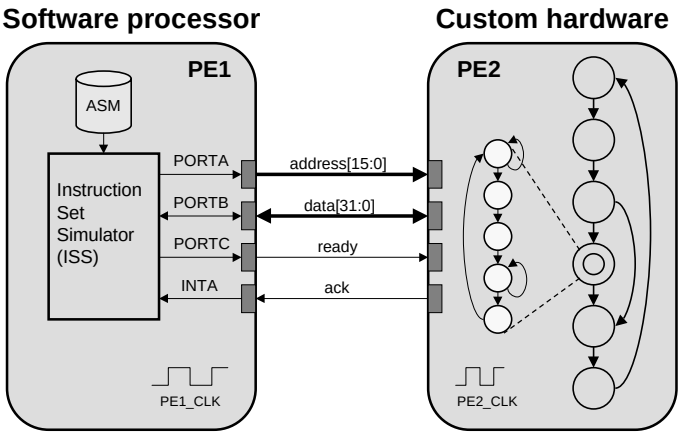
Bus driver:

```
FmasterWrite  ; protocol layer
  OUTA    r0            ; write addr
  OUTB    r1            ; write data
  OUTC    #0001         ; raise ready
  MOVE    ack_event,r2
  CALL    Fevent_wait   ; wait for ack ev.
  OUTC    #0000         ; lower ready
  RET
FmasterSend   ; application layer
  LOOP    L1END,r2      ; loop over data
  MOVE    (r6)+,r1
  CALL    FmasterWrite ; call protocol
L1END
  RET
```

Interrupt handler:

```
FintaHandler
  PUSH    r2
  MOVE    ack_event,r2
  CALL    Fevent_notify  ; notify ack ev.
  POP     r2
  RTI
```

Implementation Model



Implementation Model (2)

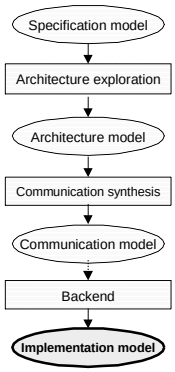
```
#include <iss.h>    // ISS API

behavior PE1( out bit[15:0] addr.
                  bit[31:0] data,
                  OSignal   ready,
                  ISignal   ack ) {
  void main(void) {
    // initialize ISS, load program
    iss.startup();
    iss.load("a.out");

    // run simulation
    for( ; ; ) {
      // drive inputs
      iss.porta = addr;
      iss.portb = data;
      iss.inta  = ack.val();

      // run processor cycle
      iss.exec();
      waitfor( PE1_CLK );

      // update outputs
      data = iss.portb;
      ready.set( iss.portc & 0x01 );
    }
  }
};
```

```
behavior PE2( in bit[15:0] addr, bit[31:0] data,
              ISignal ready, OSignal ack )
{
  // Interface FSM
  HWBusDriver bus( addr, data, ready, ack );

  int v1;                    // memory

  B13Rcv b13rcv( bus, v1 );  // FSMDs
  B3     b3    ( v1, bus );
  B34Snd b34snd( bus );

  void main(void) {
    b13rcv.main();
    b3.main();
    b34snd.main();
  }
};
```

```
behavior Design() {
  bit[15:0] address;  bit[31:0] data;
  Signal    ready(), ack();
  PE1 pe1( address, data, ready, ack );
  PE2 pe2( address, data, ready, ack );
  void main(void) {
    par {
      pe1.main(); pe2.main();
}}};
```

Implementation Model

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



Summary & Conclusions

- **SpecC system-level design methodology & language**
  - Four levels of abstraction
    - Specification model: untimed, functional
    - Architecture model: estimated, structural
    - Communication model: timed, bus-functional
    - Implementation model: cycle-accurate, RTL/IS
  - Specification to RTL
    - System synthesis
      1. Architecture exploration (map computation to components)
      2. Communication synthesis (map communication to busses)
    - Backend
      3. hardware synthesis, software compilation, interface synthesis
  - Well-defined, formal models & transformations
    - Automatic, gradual refinement
    - Executable models, testbench re-use
    - Simple verification