EECS 222A: System-on-Chip Description and Modeling Lecture 2

Rainer Dömer

doemer@uci.edu

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

Lecture 2: Overview

- The SpecC Language
  - Foundation
  - Types
  - Structural and behavioral hierarchy
  - Concurrency
  - State transitions
  - Exception handling
  - Communication
  - Synchronization
  - Timing
  - (RTL) *to be addressed separately later
- Homework Assignment 1
  - Administration, Tasks

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>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 5

The SpecC Language

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

```
/* HelloWorld.c */

#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

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

```
// HelloWorld.sc

#include <stdio.h>

behavior Main
{
  int main(void)
  {
    printf("Hello World!\n");
    return 0;
  }
};
```

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 6

The SpecC Language

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

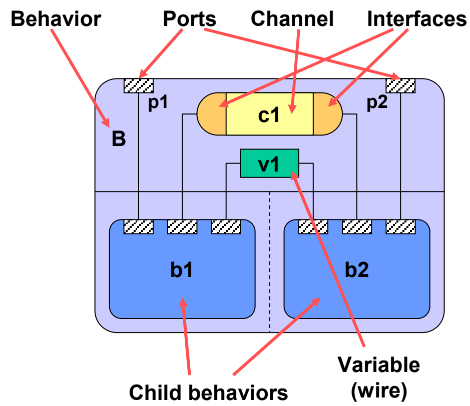
The SpecC Language

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

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

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

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

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

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

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

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

 return r;
}
```

The SpecC Language

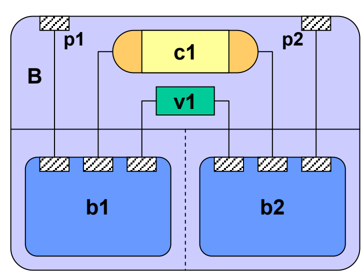
- Basic structure
  - Top behavior
  - Child behaviors
  - Channels
  - Interfaces
  - Variables (wires)
  - Ports



The SpecC Language

- Basic structure

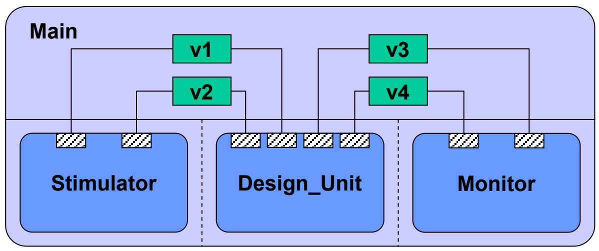
```
interface I1
{
  bit[63:0] Read(void);
  void Write(bit[63:0]);
};

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

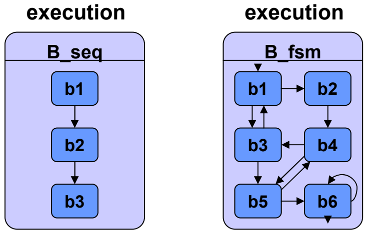


EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 11

The SpecC Language

- Behavioral hierarchy

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



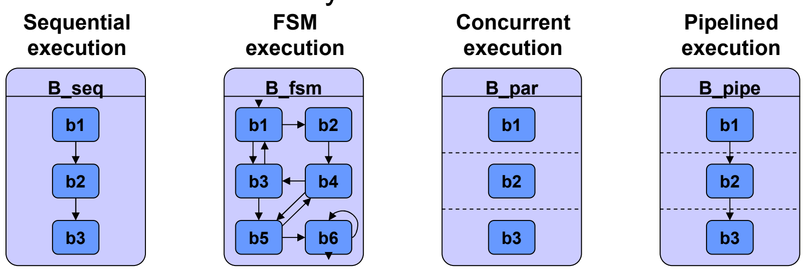

```
behavior B_seq
{
 B b1, b2, b3;

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

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

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 12

The SpecC Language

- Behavioral hierarchy




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

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

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

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

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

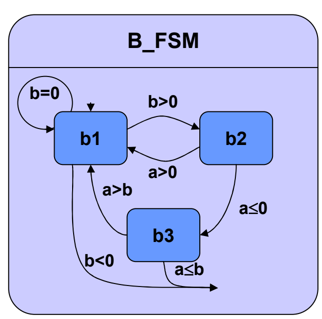
```
behavior B_pipe
{
 B b1, b2, b3;

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

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 13

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

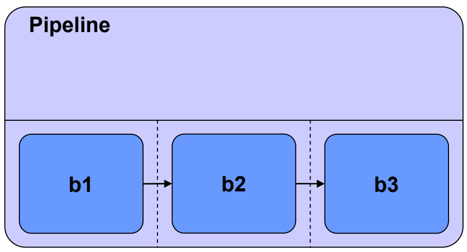
```
behavior B_FSM(in int a, in int b)
{
 B b1, b2, b3;

 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; }
       }
 }
};
```



EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 14

The SpecC Language

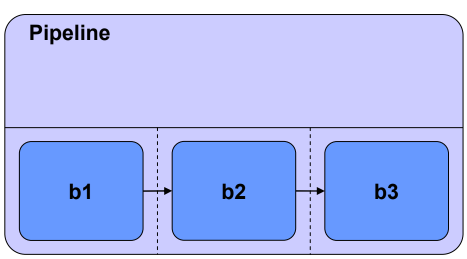
- Pipeline
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };




```
behavior Pipeline
{
 Stage1 b1;
 Stage2 b2;
 Stage3 b3;

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


The SpecC Language

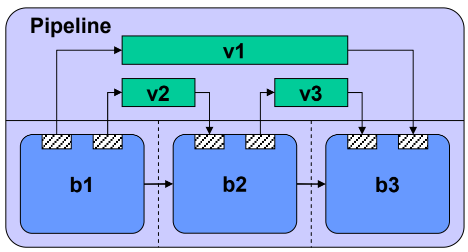
- Pipeline
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }




```
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

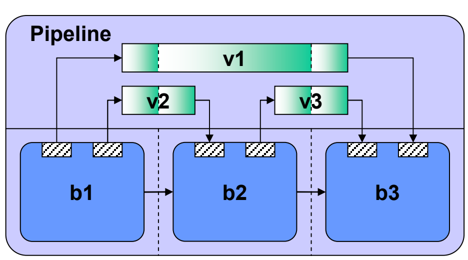



```
behavior Pipeline
{
 int v1;
 int v2;
 int v3;

 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;
        }
  }
};
```

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 17

The SpecC Language

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

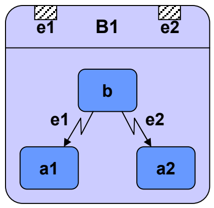



```
behavior Pipeline
{
 piped piped int v1;
 piped int v2;
 piped int v3;

 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;
        }
  }
};
```

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 18

The SpecC Language

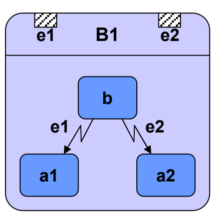
- Exception handling
  - Abortion
  - Interrupt

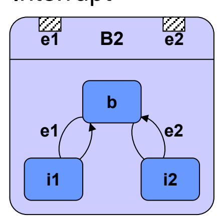



```
behavior B1(in event e1, in event e2)
{
 B b, a1, a2;

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


The SpecC Language

- Exception handling
  - Abortion
  - Interrupt



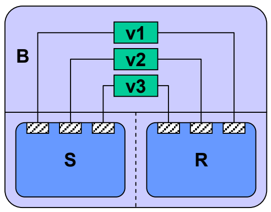



```
behavior B1(in event e1, in event e2)
{
 B b, a1, a2;

 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

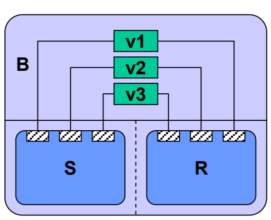
- Communication
  - via shared variable



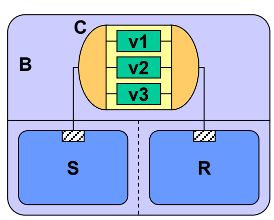
Shared memory

The SpecC Language

- Communication
  - via shared variable
  - via virtual channel



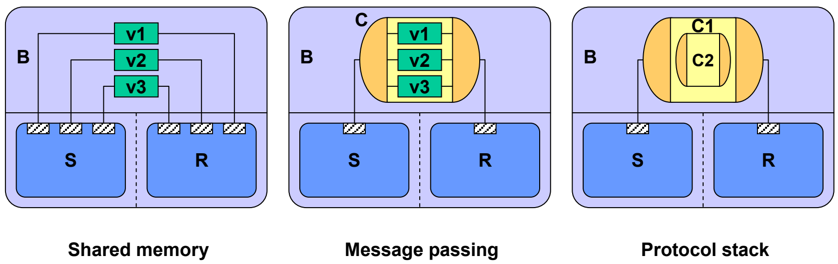
Shared memory



Message passing

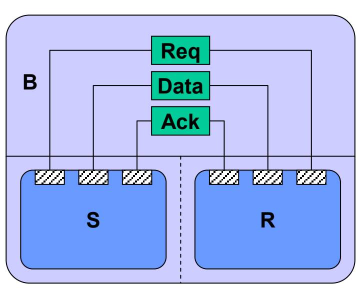
The SpecC Language

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



The SpecC Language

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

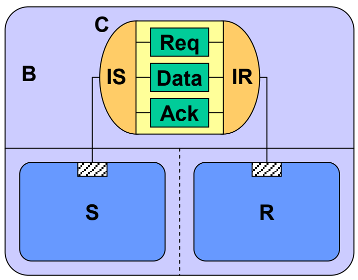



```
behavior S(out event Req,
           out float Data,
           in  event Ack)
{
 float X;
 void main(void)
 { ...
   Data = X;
   notify Req;
   wait Ack;
   ...
 }
};
```

```
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*> };




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

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

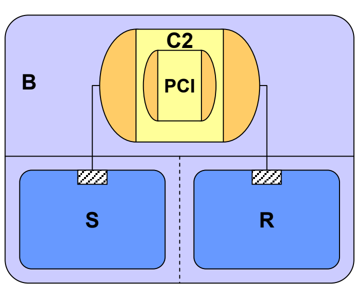
```
behavior R(IR Port)
{
 float Y;
 void main(void)
 {...
  Y=Port.Receive();
  ...
 }
};
```

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

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

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 25

The SpecC Language

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




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

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

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

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

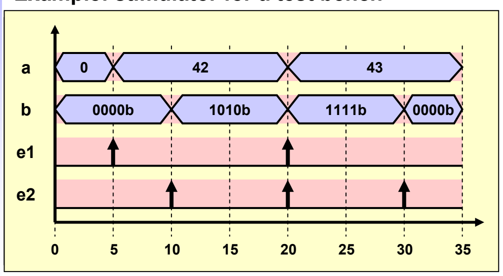
```
channel PCI
   implements PCI_IF;

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

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 26

The SpecC Language

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

Example: stimulator for a test bench



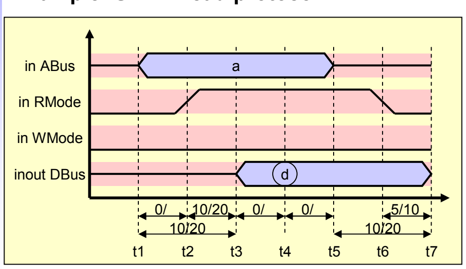

```
behavior Testbench_Driver
         (inout int a,
          inout int b,
          out event e1,
          out event e2)
{
  void main(void)
  {
   waitfor 5;
   a = 42;
   notify e1;

   waitfor 5;
   b = 1010b;
   notify e2;

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

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

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 27

The SpecC Language

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

Example: SRAM read protocol



Specification

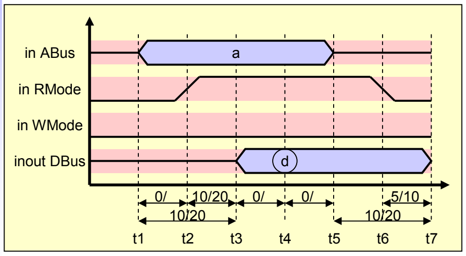
```
bit[7:0] Read_SRAM(bit[15:0] a)
{
 bit[7:0] d;

 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);
}
```

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 28

The SpecC Language

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

Example: SRAM read protocol



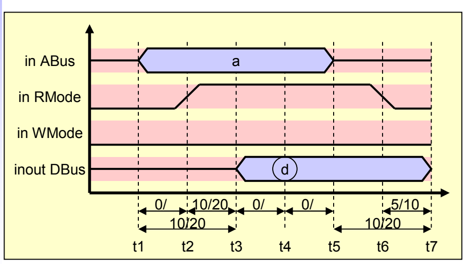
Implementation 1

```
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);
}
```

The SpecC Language

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

```
// MyDesign.sc

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

import "Interfaces/I1";
import "Channels/PCI_Bus";
import "Components/MPEG-2";

...
```


The SpecC Language

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

The SpecC Language

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

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

```
/* comment, not persistent */

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

behavior CPU(in event CLK, in event RST, ...)
{
  // local annotations
  note MinMaxClockFreq = {750*1e6, 800*1e6 };
  note CLK.IsSystemClock = true;
  note RST.IsSystemReset = true;
  ...
};
```


Summary

- SpecC language
  - True superset of ANSI-C
    - ANSI-C plus extensions for HW-design
  - Support of all concepts needed in system design
    - Structural and behavioral hierarchy
    - Concurrency
    - State transitions
    - Communication
    - Synchronization
    - Exception handling
    - Timing
    - (RTL)

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**

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 35

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”

EECS222A: SoC Description and Modeling, Lecture 2 (c) 2009 R. Doemer 36