EECS 222: Embedded System Modeling Lecture 4

Rainer Dömer

doemer@uci.edu

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

Lecture 4: Overview

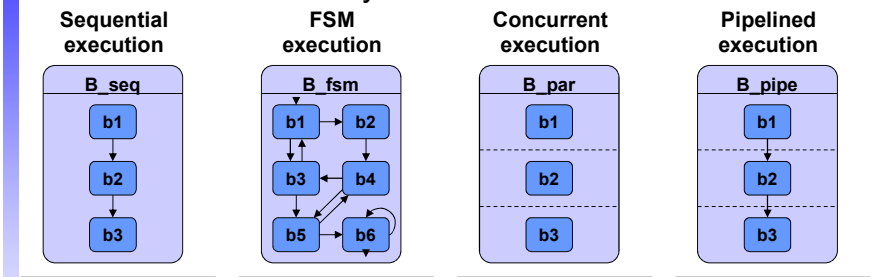
- Review
  - Behavioral hierarchy in SpecC
- Introduction to the SpecC Language (Part 2)
  - State transitions
  - Exception handling
  - Communication
  - Synchronization
- Homework Assignment 2
  - Setup the SpecC compiler and simulator
  - Run simple examples
  - Create a producer-consumer example

SpecC Language Overview

- Lecture 3
  - Foundation, types
  - Structural hierarchy
  - Behavioral hierarchy
- Lecture 4
  - State transitions
  - Exception handling
  - Communication and synchronization
- Lecture 5
  - Timing
  - Library support and persistent annotation
- Lecture 12 (tentative)
  - Register Transfer Level (RTL) support

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:{…}
         …}
  }
};
```

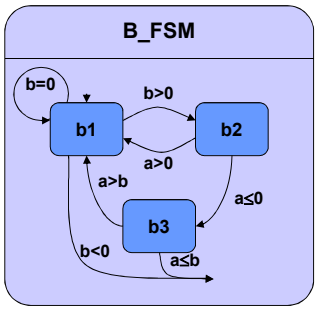
```
behavior B_par
{
 B b1, b2, b3;

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

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

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

The SpecC Language

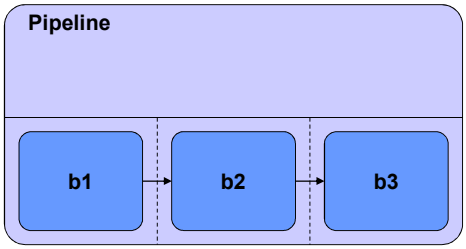
- Finite State Machine (FSM)
  - Explicit state transitions
    - triple < *current_state, condition, next_state* >
    - **fsm** { <*current_state*> : { **if** <*condition*> **goto** <*next_state*> } … }
  - Moore-type FSM
  - Mealy-type FSM

```
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

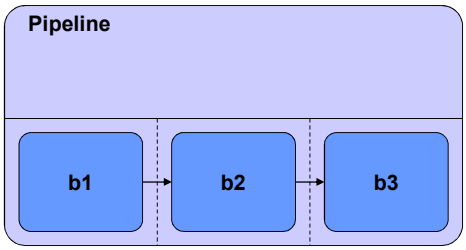
- Pipeline
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };




```
behavior Pipeline
{




 Stage1 b1;
 Stage2 b2;
 Stage3 b3;

 void main(void)
 {

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

The SpecC Language

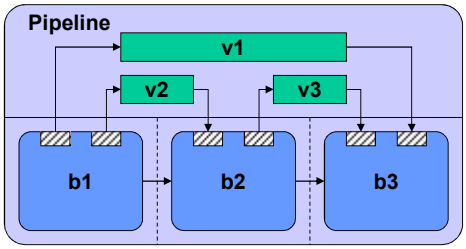
- Pipeline
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }




```
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

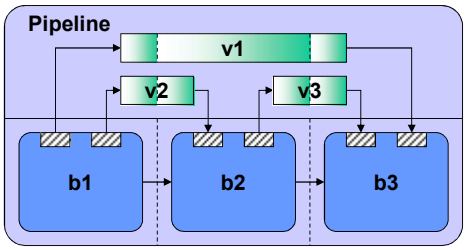
- Pipeline
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }
  - Support for automatic buffering




```
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

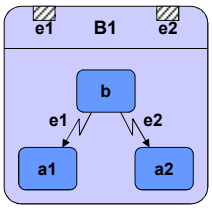
- Pipeline
  - Explicit execution in pipeline fashion
    - **pipe** { <*instance_list*> };
    - **pipe** (<*init*>; <*cond*>; <*incr*>) { … }
  - Support for automatic buffering
    - **piped** […] <*type*> <*variable_list*>;




```
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

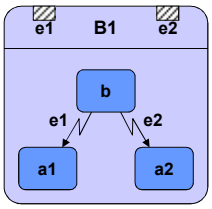
- Exception handling
  - Abortion
  - Interrupt




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

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

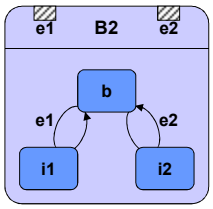
The SpecC Language

- Exception handling
  - Abortion




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

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

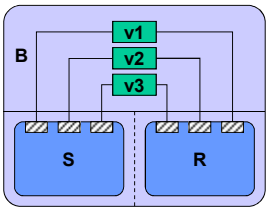
  - Interrupt




```
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

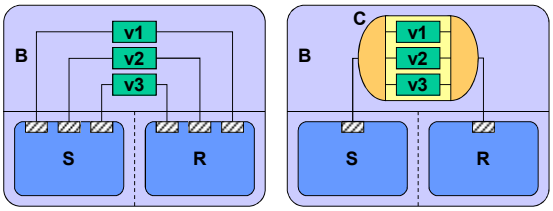
- Communication and synchronization
  - via shared variable



Shared memory

The SpecC Language

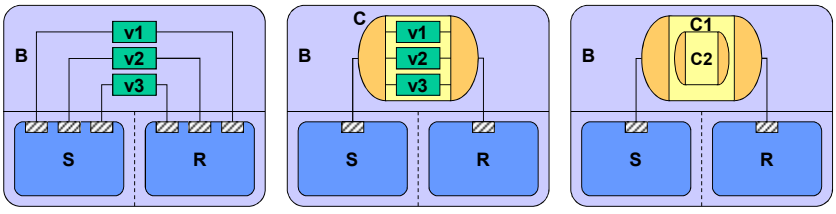
- Communication and synchronization
  - via shared variable
  - via channel with interfaces



Shared memory **Message passing**

The SpecC Language

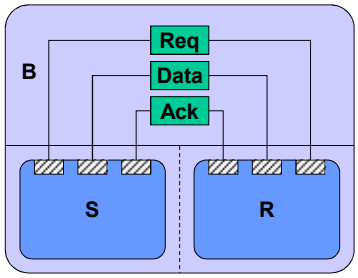
- Communication and synchronization
  - via shared variable
  - via channel with interfaces
  - via hierarchical channels



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*>;

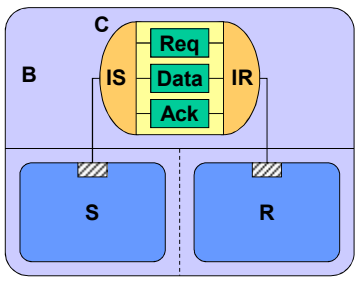



```
behavior S(out event Req,
           out float Data,
           in  event Ack)
{
 float X;
 void main(void)
 { ...
   Data = X;
   notify Req;
   wait Ack;
   ...
 }
};
```

```
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*> };




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

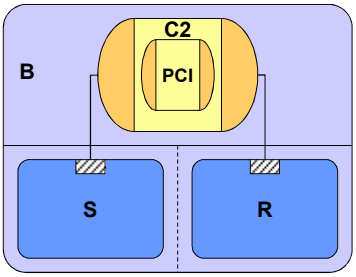
```
behavior S(IS Port)
{
 float X;
 void main(void)
 { ...
   Port.Send(X);
   ...
 }
};
```

```
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;
 }
};
```

The SpecC Language

- Hierarchical channel
  - Virtual channel implemented by standard bus protocol
    - Example: simplified 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;
 }
};
```


Homework Assignment 2

- Task: Introduction to SpecC Compiler and Simulator
- Steps
  - Setup the SpecC compiler **`scc`**
    - **`source /opt/sce/bin/setup.csh`**
  - Use **`scc`** to compile and simulate some simple examples
    - **`scc HelloWorld -vv`**
    - See **`man scc`** for the compiler manual page
  - Build and simulate a Producer-Consumer example
    - See Slide 16 (behavior **`B`** can become **`Main`**)
    - Producer **`Prod`** should send string **`"Apples and Oranges"`** character by character to the consumer **`Cons`**
    - Both print the sent/received characters to the screen
- Deliverables
  - Source and log file: **`ProdCons.sc`**, **`ProdCons.log`**
- Due
  - April 17, 2017, 12pm (noon!)