

EECS 222: Embedded System Modeling Lecture 3

Rainer Dömer

doemer@uci.edu

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

Lecture 3: Overview

- System-Level Description Languages
 - Goals
 - Requirements
- Introduction to the SpecC Language (Part 1)
 - Foundation, types
 - Structural hierarchy
 - Behavioral hierarchy
 - Exception handling

System-Level Description Languages

- **Goals and Requirements**
 - Formality
 - Formal syntax and semantics
 - Executability
 - Validation through simulation
 - Synthesizability
 - Implementation in HW and/or SW
 - Support for IP reuse
 - Modularity
 - Hierarchical composition
 - Separation of concepts
 - Completeness
 - Support for all concepts found in embedded systems
 - Orthogonality
 - Orthogonal constructs for orthogonal concepts
 - Simplicity
 - Minimality

EECS222: Embedded System Modeling, Lecture 3 (c) 2020 R. Doemer 3

System-Level Description Languages

- **Requirements supported by existing languages**

	C	C++	Java	VHDL	Verilog	HardwareC	Statecharts	SpecCharts	SpecC	SystemC
Behavioral hierarchy	○	○	○	○	○	○	●	●	●	○
Structural hierarchy	○	○	○	●	●	●	○	○	●	●
Concurrency	○	○	◐	●	●	●	●	●	●	●
Synchronization	○	○	◐	●	●	●	●	●	●	●
Exception handling	◐	●	●	○	●	○	◐	◐	●	○
Timing	○	○	○	●	●	◐	◐	◐	●	●
State transitions	○	○	○	○	○	○	●	●	●	○
Composite data types	●	●	●	●	◐	○	○	●	●	●

○ not supported ◐ partially supported ● supported

EECS222: Embedded System Modeling, Lecture 3 (c) 2020 R. Doemer 4

SpecC Language Overview

- Part 1: Today
 - Foundation, types
 - Structural hierarchy
 - Behavioral hierarchy
 - Exception handling
- Part 2:
 - Communication and synchronization
 - Timing
 - Library support
 - Persistent annotation
- Part 3:
 - Register Transfer Level (RTL) support

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
 - Is not just a class library (as SystemC)
 - Relies on dedicated compiler and static analysis

EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

7

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;  
}
```

EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

8

The SpecC Language

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

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

```
/* HelloWorld.c */
#include <stdio.h>

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

```
// HelloWorld.sc
#include <stdio.h>

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

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;
}
    
```

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
11

The SpecC Language

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

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
12

The SpecC Language

- Structural hierarchy

```

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()*;

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
13

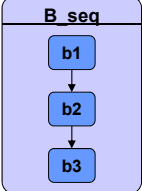
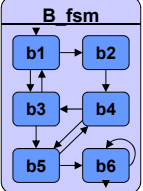
The SpecC Language

- Typical test bench
 - Top-level behavior: Main
 - Stimulus provides test vectors
 - Design under test (DUT) represents the target SoC
 - Monitor observes and checks outputs

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
14

The SpecC Language

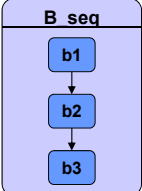
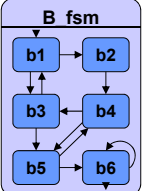
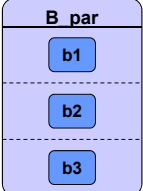
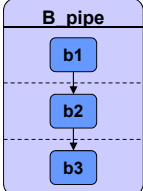
- Behavioral hierarchy

Sequential execution	FSM execution	Concurrent execution	Pipelined execution
			
<pre>behavior B_seq { B b1, b2, b3; void main(void) { b1; b2; b3; } };</pre>	<pre>behavior B_fsm { B b1, b2, b3, b4, b5, b6; void main(void) { fsm { b1: {...} b2: {...} ... } } };</pre>		

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
15

The SpecC Language

- Behavioral hierarchy

Sequential execution	FSM execution	Concurrent execution	Pipelined execution
			
<pre>behavior B_seq { B b1, b2, b3; void main(void) { b1; b2; b3; } };</pre>	<pre>behavior B_fsm { B b1, b2, b3, b4, b5, b6; void main(void) { fsm { b1: {...} b2: {...} ... } } };</pre>	<pre>behavior B_par { B b1, b2, b3; void main(void) { par { b1; b2; b3; } } };</pre>	<pre>behavior B_pipe { B b1, b2, b3; void main(void) { pipe { b1; b2; b3; } } };</pre>

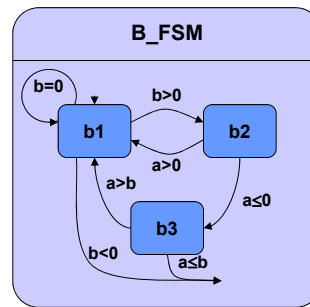
EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
16

The SpecC Language

- Finite State Machine (FSM)
 - Explicit state transitions
 - triple $\langle \text{current_state}, \text{condition}, \text{next_state} \rangle$
 - `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; }
        }
  };
}
```



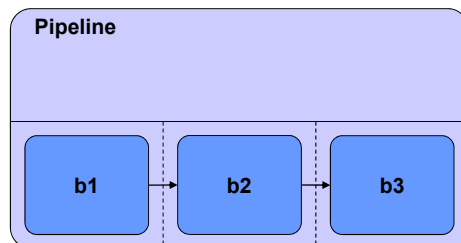
EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

17

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;
    }
  };
}
```

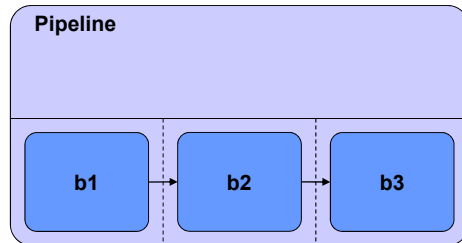
EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

18

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;
        }
    }
};
```

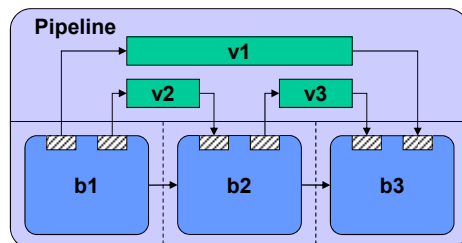
EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

19

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;
        }
    }
};
```

EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

20

The SpecC Language

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

The diagram shows a pipeline with three stages labeled b1, b2, and b3. Above each stage is a piped variable: v1 above b1, v2 above b2, and v3 above b3. Arrows indicate the flow of data from b1 to b2 to b3. The variables v1, v2, and v3 are shown as green bars with arrows pointing to their respective stages.

```
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;
    }
  }
};
```

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
21

The SpecC Language

- Exception handling
 - Abortion
 - Interrupt

The diagram shows a behavior B1 with a try block b and two trap blocks a1 and a2. Events e1 and e2 are shown as inputs to the try block. Arrows indicate the flow from b to a1 and b to a2.

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

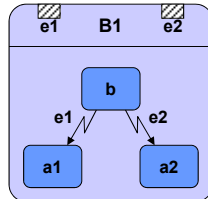
  void main(void)
  {
    try { b; }
    trap (e1) { a1; }
    trap (e2) { a2; }
  }
};
```

EECS222: Embedded System Modeling, Lecture 3
(c) 2020 R. Doemer
22

The SpecC Language

- Exception handling

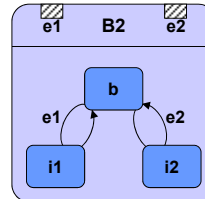
- Abortion



```
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; }
  }
};
```

EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

23

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 8 of Lecture 4 for reference
 - 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
 - January 22, 2020, 6pm

EECS222: Embedded System Modeling, Lecture 3

(c) 2020 R. Doemer

24