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

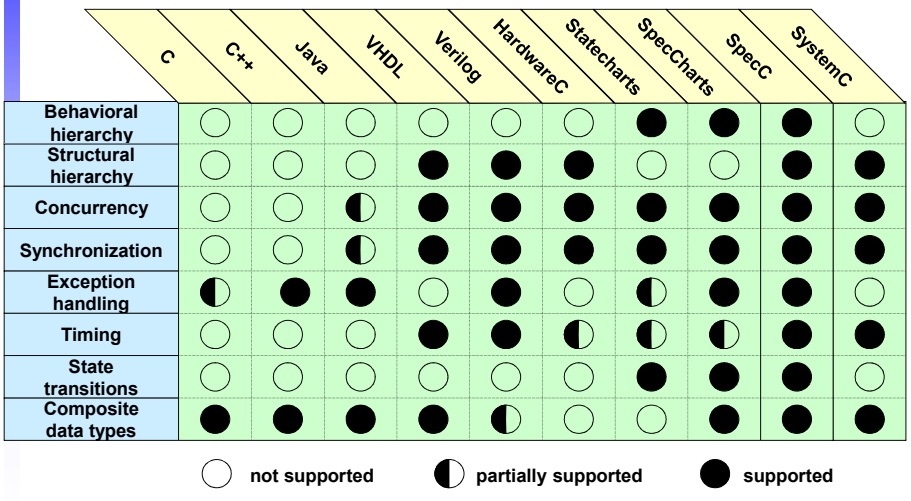
- Review
  - SLDL goals and requirements
- Introduction to the SpecC Language (Part 1)
  - Foundation
  - Types
  - Structural hierarchy
  - Behavioral hierarchy
- Discussion: Homework Assignment 1
  - Project setup, study of application

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

System-Level Description Languages

- Requirements supported by existing languages



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

- 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

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;
}
```

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;
  }
};
```


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;`

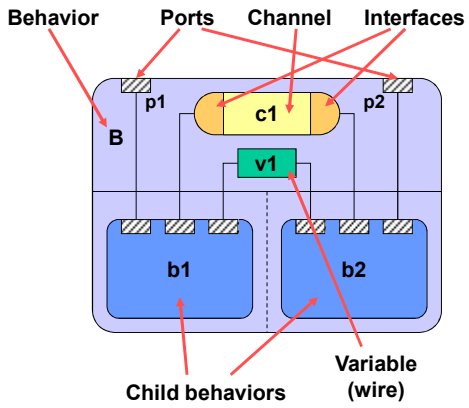
The SpecC Language

- 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

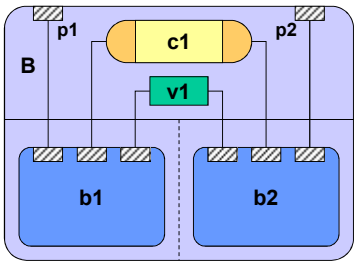
- Basic structure
  - Top behavior
  - Child behaviors
  - Channels
  - Interfaces
  - Variables (wires)
  - Ports



The SpecC Language

- Structural hierarchy

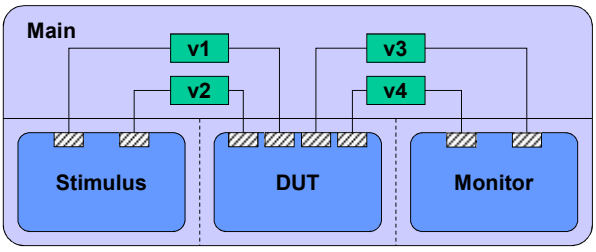
```
interface I1
{
  bit[63:0] Read(void);
  void Write(bit[63:0]);
};

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();`

The SpecC Language

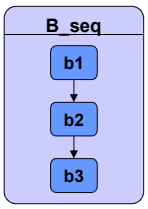
- Typical test bench
  - Top-level behavior: `Main`
  - Stimulus provides test vectors
  - Design under test (DUT) represents the target SoC
  - Monitor observes and checks outputs



The SpecC Language

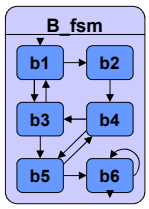
- Behavioral hierarchy

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:{…}
         …}
  }
};
```

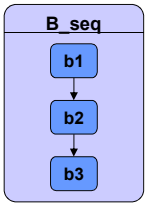
Concurrent execution

Pipelined execution

The SpecC Language

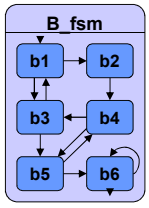
- Behavioral hierarchy

Sequential execution



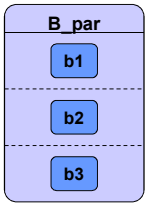

```
behavior B_seq
{
 B b1, b2, b3;

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

FSM execution



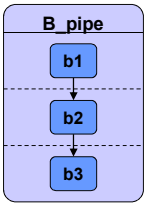

```
behavior B_fsm
{
 B b1, b2, b3,
   b4, b5, b6;
 void main(void)
 { fsm { b1:{…}
         b2:{…}
         …}
  }
};
```

Concurrent execution




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

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

Pipelined execution

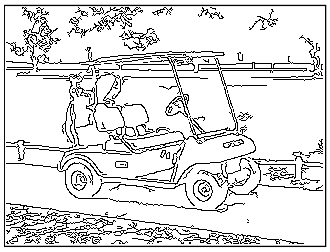



```
behavior B_pipe
{
 B b1, b2, b3;

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

EECS 222 Project

- Application Example: Canny Edge Detector
  - Embedded system model for image processing: Automatic Edge Detection in a Digital Camera

golfcart.pgm



golfcart.pgm_s_0.60_l_0.30_h_0.80.pgm

  - Application Source and Documentation:
    - http://marathon.csee.usf.edu/edge/edge_detection.html
    - http://en.wikipedia.org/wiki/Canny_edge_detector

Discussion: Homework Assignment 1

- Administration
  - EECS Department Linux Servers
    - **`crystalcove.eecs.uci.edu`**, and others
    - Linux environment (CentOS 6.8)
    - Access via secure shell protocol (SSH)
  - Accounts
    - User ID same as your UCInetID
    - Password same as your EEE password
  - Login and make yourself familiar with
    - Command-line tools and GUI tools (which need X client)
    - Text editing and C/C++ programming
    - Image processing tools

Discussion: Homework Assignment 1

- Task: Introduction to Application Example
  - Canny Edge Detector
  - Algorithm for edge detection in digital images
- Steps
  1. Setup your Linux programming environment
  2. Download, adjust, and compile the application C code with the GNU C compiler (gcc)
  3. Study the application
- Deliverables
  - None at this time (preparation for following assignments)
- Due
  - By next week: April 10, 2017, 12pm (noon!)