EECS 222: Embedded System Modeling Lecture 5

Rainer Dömer

doemer@uci.edu

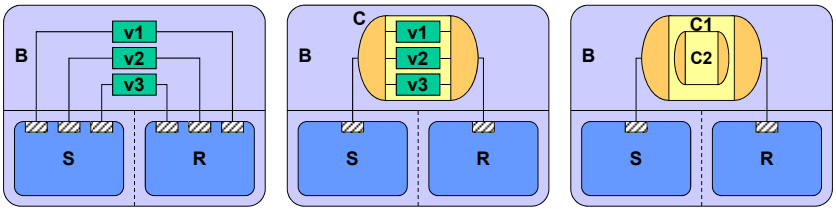
The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 5: Overview

- Review
  - Communication and synchronization in SpecC
  - Assignment 2: Producer-consumer example in SpecC
- SpecC Standard Channels
  - Synchronization
  - Communication
- SpecC Tools
  - Compiler and simulator
  - SIR tools
  - Debugging and tracing

The SpecC Language

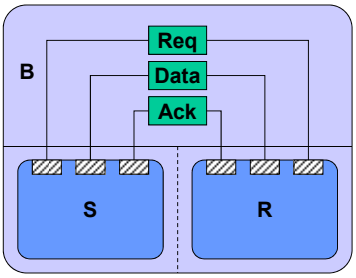
- Communication and synchronization
  - via shared variable
  - via channel with interfaces
  - via hierarchical channels



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*>;

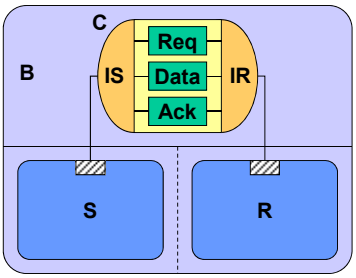



```
behavior S(out event Req,
           out float Data,
           in  event Ack)
{
 float X;
 void main(void)
 { ...
   Data = X;
   notify Req;
   wait Ack;
   ...
 }
};
```

```
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);
};
```

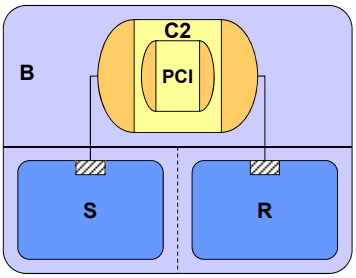
```
channel C
    implements IS, IR
{
 event Req;
 float Data;
 event Ack;

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

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

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


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);
};
```

```
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 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 5 (c) 2020 R. Doemer 7

The SpecC Language

- Summary
  - True superset of ANSI-C
    - ANSI-C plus extensions for HW-design
  - Support of all concepts needed in system design
    - Structural hierarchy
    - Behavioral hierarchy
    - State transitions
    - Exception handling
    - Communication
    - Synchronization
    - Timing
    - Library support
    - Persistent annotation
    - Register Transfer Level (RTL) modeling *(discussed later)*

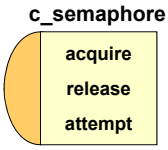
EECS222: Embedded System Modeling, Lecture 5 (c) 2020 R. Doemer 8

SpecC Standard Channels

- SpecC Standard Channel Library
  - introduced with SpecC Language Version 2.0
  - safe and consistent modeling of common channels with standard interfaces
- Standard Synchronization Channels
  - Mutually exclusive execution
    - Semaphore, mutex, critical section
  - Dependent execution
    - Token
    - Handshake, barrier
- Standard Communication Channels
  - Typed and type-less message passing
    - Double handshake
    - Queue

SpecC Standard Channels

- Standard Synchronization Channels
  - Mutually exclusive execution
    - Semaphore

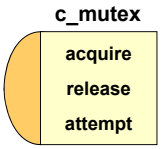



```
interface i_semaphore
{
  void acquire(void);
  void release(void);
  bool attempt(void);
};
```

```
channel c_semaphore(
     in const unsigned long c)
  implements i_semaphore;
```

SpecC Standard Channels

- Standard Synchronization Channels
  - Mutually exclusive execution
    - Semaphore
    - Mutex



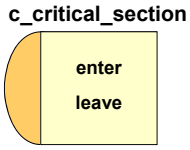

```
interface i_semaphore
{
  void acquire(void);
  void release(void);
  bool attempt(void);
};
```

```
channel c_mutex
  implements i_semaphore;
```

```
channel c_semaphore(
    in const unsigned long c)
  implements i_semaphore;
```


SpecC Standard Channels

- Standard Synchronization Channels
  - Mutually exclusive execution
    - Critical section

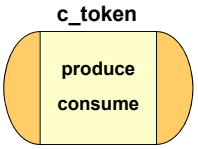



```
interface i_critical_section
{
  void enter(void);
  void leave(void);
};
```

```
channel c_critical_section
  implements i_critical_section;
```

SpecC Standard Channels

- Standard Synchronization Channels
  - Dependent execution
    - Token




```
interface i_consumer
{
  void consume(unsigned long n);
};
```

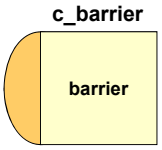
```
interface i_producer
{
  void produce(unsigned long n);
};
```

```
interface i_token
{
  void consume(unsigned long n);
  void produce(unsigned long n);
};
```

```
channel c_token
  implements i_consumer,
             i_producer,
             i_token;
```


SpecC Standard Channels

- Standard Synchronization Channels
  - Dependent execution
    - Barrier

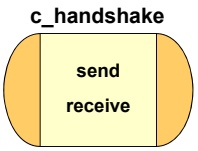



```
interface i_barrier
{
  void barrier(void);
};
```

```
channel c_barrier(
    in unsigned long n)
  implements i_barrier;
```

SpecC Standard Channels

- Standard Synchronization Channels
  - Dependent execution
    - Handshake



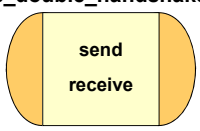

```
interface i_receive
{
  void receive(void);
};
```

```
interface i_send
{
  void send(void);
};
```

```
channel c_handshake
  implements i_receive,
             i_send;
```


SpecC Standard Channels

- Standard Communication Channels
  - Type-less message passing
    - Double handshake




```
interface i_receiver
{
  void receive(void          *d,
               unsigned long l);
};
```

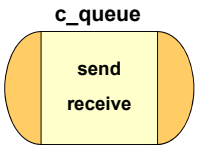
```
interface i_sender
{
  void send(void          *d,
            unsigned long l);
};
```

```
interface i_tranceiver
{
  void receive(void *d, unsigned long l);
  void send(void *d, unsigned long l);
};
```

```
channel c_double_handshake
  implements i_receiver,
             i_sender;
```

SpecC Standard Channels

- Standard Communication Channels
  - Type-less message passing
    - Double handshake
    - Queue




```
interface i_receiver
{
  void receive(void          *d,
               unsigned long l);
};
```

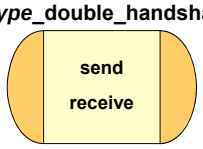
```
interface i_sender
{
  void send(void          *d,
            unsigned long l);
};
```

```
interface i_tranceiver
{
  void receive(void *d, unsigned long l);
  void send(void *d, unsigned long l);
};
```

```
channel c_queue(
     in const unsigned long s)
  implements i_receiver,
             i_sender,
             i_tranceiver;
```


SpecC Standard Channels

- Standard Communication Channels
  - Typed message passing
    - Double handshake




```
interface i_type_receiver
{
  void receive(type *d);
};
```

```
interface i_type_sender
{
  void send(type d);
};
```

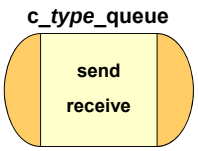
```
interface i_type_tranceiver
{
  void receive(type *d);
  void send(type d);
};
```

```
channel c_type_double_handshake
  implements i_type_receiver,
             i_type_sender;
```

- *type* can be any basic or composite SpecC type

SpecC Standard Channels

- Standard Communication Channels
  - Typed message passing
    - Double handshake
    - Queue




```
interface i_type_receiver
{
  void receive(type *d);
};
```

```
interface i_type_sender
{
  void send(type d);
};
```

```
interface i_type_tranceiver
{
  void receive(type *d);
  void send(type d);
};
```

```
channel c_type_queue(
      in const unsigned long s)
  implements i_type_receiver,
             i_type_sender;
```

- *type* can be any basic or composite SpecC type

SpecC Standard Channels

- Using Standard Channels
  - Import synchronization channels
    - semaphore channel `import "c_semaphore";`
    - mutex channel `import "c_mutex";`
    - critical section `import "c_critical_section";`
    - token `import "c_token";`
    - barrier `import "c_barrier";`
    - handshake `import "c_handshake";`
  - Import type-less communication channels
    - double handshake `import "c_double_handshake";`
    - queue `import "c_queue";`
  - Include typed communication channels
    - double handshake `#include <c_typed_double_handshake>`
    - queue `#include <c_typed_queue>`

SpecC Standard Channels

- Using Typed Communication Channels
  - Include channel header file (from **$SPECC/inc/**)
    - double handshake **#include <c_typed_double_handshake>**
    - queue **#include <c_typed_queue>**
  - Example:

```
#include <c_typed_queue.sh>
struct packet { int a, b, c; };
DEFINE_I_TYPED_SENDER(packet, struct packet)
DEFINE_I_TYPED_RECEIVER(packet, struct packet)
DEFINE_C_TYPED_QUEUE(packet, struct packet)
behavior Sender(i_packet_sender Port)
{ void main(void)
    { struct packet Data = { 1, 2, 3 };
      // ...
      Port.send(Data);
      // ...
    }
};
```

Example source code available here:
~eecs222/public/queue.sc

SpecC Tools

- Compilation and Simulation
  - **scc DesignName -sc2out -vv -ww**
  - **./DesignName**
  - Header file **sim.sh**
    - Access to simulation time
      - macros **PICO_SEC**, **NANO_SEC**, **MICRO_SEC**, **MILLI_SEC**, **SEC**
      - typedef **sim_time**, **sim_delta**, **sim_time_string**
      - function **now()**, **delta()**
      - conversion functions **time2str()**, **str2time()**
    - Handling of bit vectors
      - conversion functions **bit2str()**, **ubit2str()**, **str2bit()**, **str2ubit()**
    - Handling of long-long values
      - conversion functions **ll2str()**, **ull2str()**, **str2ll()**, **str2ull()**

SpecC Tools

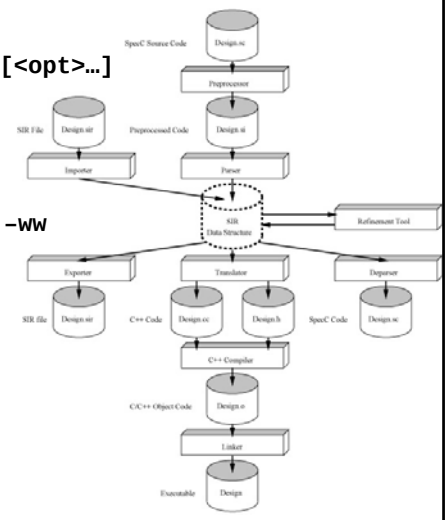
- SpecC Simulation Time
  - Example: Print the current simulation time

```
#include <sim.sh>
...
sim_time t;
sim_delta d;
sim_time_string buffer;
...
t = now();  d = delta();
printf("Time is now %s pico seconds.\n", time2str(buffer, t));
printf("(delta count is %s)\n", time2str(buffer, d);
waitfor 42000 NANO_SEC;
printf("Time is now %s pico seconds.\n", time2str(buffer, t));
printf("Time is now %s nano seconds.\n",
            time2str(buffer, t/(1 NANO_SEC)));
...
```


SpecC Tools

- SpecC Compiler
  - Command line interface
  - Usage: `scc <design> [<cmd>] [<opt>…]`
  - Help: `scc -h`
    `man scc`
  - Example:

```
% scc HelloWorld -sc2out -v -ww
scc: SpecC Compiler V 2.2.1
(c)2012 CECS, UC Irvine
Preprocessing...
Parsing...
Translating...
Compiling...
Linking…
Done.
```



SpecC Tools

- SIR Tools
  - Tools working with SpecC Internal Representation (SIR) files
  - Example:

```
% scc Adder -sc2sir -o Adder.sir
% sir_list -t Adder.sir
behavior ADD8
behavior AND2
behavior FA
behavior HA
behavior Main
behavior XOR2
% sir_tree -bt Adder.sir FA
behavior FA
|------ HA ha1
|       |------ AND2 and1
|       \------ XOR2 xor1
|------ HA ha2
|       |------ AND2 and1
|       \------ XOR2 xor1
\------ OR2 or1
```


SpecC Tools

- Debugging
  - **`scc DesignName –sc2out –vv –ww –g -G`**
    `gdb ./DesignName`
    `ddd ./DesignName`
  - Header file **`sim.sh`**
    - Access to simulation engine state
      - functions **`ready_queue()`**, **`running_queue()`**, etc.
      - functions **`_print_ready_queue()`**, **`_print_running_queue()`**, etc.
      - function **`_print_process_states()`**
      - function **`_print_simulator_state()`**
    - Access to current instance
      - functions **`active_class()`**, **`active_instance()`**
      - functions **`current_class()`**, **`current_instance()`**
      - functions **`print_active_path()`**, **`print_current_path()`**
      - ...

SpecC Tools

- Tracing
  - **`scc DesignName –sc2out –vv –ww -Tvcds`**
    `./DesignName`
    `gtkwave DesignName.vcd`
  - Trace instructions in file **`DesignName.do`**
  - Trace log in file **`DesignName.vcd`**
  - Waveform display **`gtkwave`**
    - available as **`/opt/gtkwave/bin/gtkwave`**
- Documentation:
  - E. Johnson, A. Gerstlauer, R. Dömer:
    "Efficient Debugging and Tracing of System Level Designs",
    CECS Technical Report 06-08, May 2006.
  - http://www.cecs.uci.edu/~doemer/publications/CECS_TR_06_08.pdf