Embedded System Design and Modeling

EE382V, Fall 2008

Lecture Notes 3

The SpecC System-Level Design Language

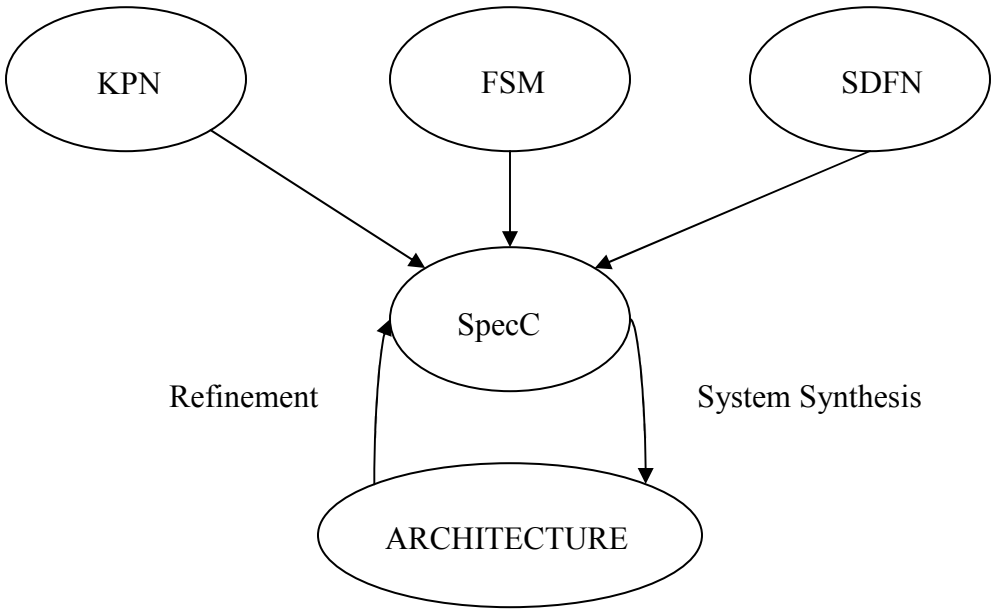
Dates: Sep 9&11, 2008
Scribe: Mahesh Prabhu

Languages:

Any language would have characters, words and syntax. Syntax being synonymous with grammar i.e. rules for forming legal "statements". Syntax identifies the structure of legal combination of the alphabet set and Semantics refers to the meaning that the legal structure conveys.

For English:
Alphabet consists of characters: 'a', 'b', 'c' …
Words: {"and", "or"}
Rules: Grammar
C Language:
Alphabet: letters, numbers, ';', …
Grammar: {"if", "switch"}
Rules: BNF

A high level model of a system is usually a combination of models of computation, such as Kahn process networks, finite state machines and synchronous data flow models. These models need to be combined into a single, common model. SpecC is a language which is powerful enough to capture the semantics of the 3 mentioned models.



In fact it is possible to capture computational models like FSM and KPN in languages like JAVA. For example in JAVA the processes in KPN can be represented by threads which run in parallel. To represent the queues we can have a "buffer" class. However these languages are not powerful enough to capture all the necessary semantics. Sometimes libraries help in filling the gap. For example we cannot processes in C but with the help of pthreads library we can model parallel processes in KPN. However libraries are not always there to fulfill the shortcomings of the language.

Discrete Event Semantics:

This represents everything as streams of events of a particular object. An event is a tuple (value,tag). For example, in the case of a signal which is initialized to 0, changes its value at time t_1 to 1 and then again flips back to 0 at time t_2, then we have the following events $(1,t_1)$ and $(0,t_2)$ where $t_1 < t_2$. Total order implies for all t_1, t_2 either $t_1 < t_2$ or $t_2 < t_1$. In cases of system with partial order it is not possible to order all events.

Orthogonality in languages: Keep language clean and simple so that most of the data types work with most of the operators. Remove superfluous items to make the language minimal. For example in C, the operators + and * work with both the basic data types float and int.

Hierarchy (or composition) in models:

Structural Hierarchy:

This indicates how two blocks are connected to each other, i.e the emphasis is on the connectivity between the blocks.

Behavioral Hierarchy:

This indicates how the blocks behave relative to each other. For example we can have 5 blocks which run in parallel or we can have 5 blocks which run sequentially one after the other.

State-charts: are hierarchical concurrent FSMs.

Computation and Communication:

Consider the producer consumer problem captured using KPN.

Process Produce:
…
send (data);
… (some computation)
wait (acknowledge)
…

```
Process Consume:
…
wait (data);
if (data is Ok)
    send (acknowledge);
…
```

We see that the communication specifics are interleaved with the communication specifics and the SpecC model permits the separation of the communication part (channel) from the computation (behavior).

Q: How does software/hardware partitioning feature in the separation of communication from computation?
Hardware/Software partitioning is a separate step in the design process. Both the communication and the computation parts will have parts carved out to be hardware and the rest will go into software. The separation of communication from computation must be done at the system level of abstraction where we haven't yet partitioned the system into hardware and software.

Segregation of computation and communication also helps design exploration, where changes in communication scheme do not affect the computation and vice versa. IPs also can be used to plug into the communication or the computation part, which are usually sold as one of the two.

SpecC:

New basic types:

Bit Vector: same as in verilog. Has the operators bit slice and the concatenation operator defined on it. The concatenation of two bit vectors will result in a bit vector whose size is the sum of the two vectors being concatenated.
Event type: this type does not have any values and is used for exception handling and synchronization. Whenever a signal changes its value an event happens due to that change.
Buffered type: this is a data type for capturing clocked design elements like flip flops.
Signal type: this is used for the representation of wires in the design.

Q: Why no high impedance (captured in Verilog as 'Z') is there in SpecC?
SpecC has a 4-value logic data type (bit4) but for capturing the design at the system level, such a low level signal value such as 'Z' would not be required.

Behavioral Hierarchy:

In SpecC the hierarchy defined should be clean. C code used to define behavior should not be intermingled with SpecC constructs used to define hierarchy. C code should be used only at the leaves of the hierarchy.

For example, the following code would be deemed unclean:

```
behavior SomeBehave() {
        B b1();
        void main( void) {
                if( k < 5) {
                        b1.main();              // this should not be done as tools would not
                                                // support this
                }
        }
}
```

The default behavioral hierarchy in SpecC is sequential execution. So a hierarchy without "fsm", "par" or "pipe" would be a sequential hierarchy.

Pipelined execution is different from concurrent execution, even though it is a type of "concurrent" execution. This semantic identifies pipelined parts as different within the system which would help other tools down the flow like logic synthesis tools to synthesize the relevant logic elements.

Consider the following bit of SpecC code:

```
void main(void) {
        int I;
        pipe( I = 0; I < 10; i++) {
                B1;
                B2;
                B3;
        }
}
```

This loop would execute as follows:

```
    B1
    B1 B2
    B1 B2 B3 (this would repeat 8 times)
        B3 B3
            B4
```

So each behavior would execute 10 times but totally there would be 12 loops.

The piped variables help define the simulation semantics for the communication between behaviors in different stages of a pipeline. When data is being passed in between such pipelined behaviors, piped variables should always be used. The semantics define the variables values written in the current piped loop iteration to be visible to the behaviors at the fan-outs at the next piped loop iteration. In case piped variables are not used for the communication then the behavior is undefined.

Exception handling:

Abortion: These are used in cases where the occurrence of an event must stop execution of some existing behavior and start executing some other behavior without the need to return to the original behavior which was stopped.

Interrupts: These are used in cases where the occurrence of an event should interrupt the current behavior and execute a "handler" behavior and at the end execution it must return to the original behavior which was interrupted.

Two exceptions can happen at the same time, the first exception specified gets the preference to be handled. Exceptions are not buffered, if an exception is being handled then the occurrence of another exception is ignored unless if is handled explicitly in nested fashion. For example:

```
behavior B2(in event e1, in event e2)
{
        B b, i1, i2;
        void main(void)
        {
                try { b; }
                interrupt (e1) {
                        try {e1;}
                        interrupt (e2) {i2;}
                }
                interrupt (e2) {i2;}
        }
}
```

Communication:

Wait, notify and notify-one are built in constructs in SpecC which can be used for synchronization for communication between behaviors.

wait – wait for any one of the events specified in the event-list to happen.
notify – trigger an event for each of the events specified in the event-list and wake up all threads waiting for those events
notifyone – trigger an event for each of the events specified in the event-list but wake up only one (randomly picked) thread waiting for each event.

Interfaces and Channels:

An "interface" defines only the external structure of the communication mechanism. A "channel" defines the internal behavior of the communication mechanism. There can be multiple channel implementations for the same interface.

SpecC Channel Library:

Barrier: This is used when there are 'N' threads that need to synchronize at some point. While creating the barrier the number 'N' is specified. Then using the call barrier () each of the threads can suspend execution. A resume of the execution happens when all of the 'N' threads have suspended.

Token: this is an interface which is used for behaviors which act as both producers and consumers.

Queue:

Following is an example of a queue usage:

```
…
int x = 10;
c1.send ( &x, sizeof(x));
…
```

In the above piece of code C1 is a queue channel object. The first argument is the pointer to the beginning of the data to be stored in the queue, and the second argument is the number of bytes to be stored. The read can similarly happen, the pointer to where the data to be stored is passed along with the number of bytes to be stored.

Queues with given data types can also be defined using the queue defined in c_typed_queue.sh as follows:-

```
#include <c_typed_queue.sh>
DEFINE_C_TYPED_QUEUE(myint, int)

…
c_myint_queue c1(5/*depth of the queue*/);
int x;
c1.send(x);
c1.receive(&x);
…
```

Handshake:
The receive() call is blocking i.e. it waits for a send () to happen. The send() call is not blocking, and if a send() happens when there is no corresponding receive() then the event is buffered until picked up (as opposed to a plain SpecC event, which would be lost in this case). If send() is called before the event is picked up, the even gets overwritten, i.e. one of the two send() is lost.

Double handshake:
This is similar to handshake with data but the send() also blocking.

Timing:

Waitfor: this is used to introduce delays; the delay values specified have no particular time units and just represent simulated time.