EECS 222A
System-on-Chip Description and Modeling
Fall 2007

Assignment 4

Posted: November 2, 2007 (week 6)
Due: November 9, 2007 (week 7)

Task: Creating Behaviors in C Code

Instructions: (by Pramod Chandraiah)

As you might know by this time, behaviors in SpecC act as a basic unit of computation. Because of their explicit syntax, compared to functions and plain C statements, the analysis and the refinement tools can easily analyze and conduct refinement tasks, such as partitioning and mapping.
In this document, we briefly describe how to encapsulate functions and C statements into behaviors, followed by precise directions to create behaviors in an MP3 decoder example.

Encapsulating Functions in Behaviors

```
1.   behavior B ( in int p1, in int p2,  out int result)
2.   {
3.      void main( )
4.      {
5.         int i1, a, b[10], s, *pa;
6.         a = p1+p2;
7.         s = p1-p2;
8.         pa = &s;
9.         …..
10.        result = f1(a, b[i1], pa);
11.        …..
12.     }
13.     int f1( int w, int x, int *p)
14.     { *p  = w+x+*p;
15.        return *p;
16.     }
17.  };
```

(a) Original model (Model 1)

```
1.   behavior  B_f1( in int w, in  int x[10], in int i, inout int s, out int c) {
2.       void main()
3.       {  c= func(w, x[i], &s);
4.       }
5.       int f1( int w, int x, int *p)
6.       { *p  = w+x+*p;
7.            return *p;
8.       }
9.   };
10.  behavior B (in int p1, in int p2, out int     result) {
11.     int a, b[10], i1, s;
12.    //Instantiate child behavior here
13.    I_B_f 1(a, b, i1, s, result);
14.    void main( )
15.    {
16.       int *pa;
17.       a = p1+p2;
18.       s = p1-p2;
19.       pa = &s;
20.       …..
21.      I_ B_f.main();
22.       …..
23.    }
24.  } ;
```

(b) Function encapsulated in behavior (Model 2)

The explicit port list that defines the interface of the behaviors is what makes behaviors easily analyzable. As shown in the figure above, encapsulating a function into a behavior involves creating a behavior body, creating the instance of the newly created behavior and replacing the function call with the call to the instance. Note that, creating the behavior body requires first determining the port list. Creating the instance requires first creating the port map list.

Initial Setup

The initial setup files are in the tar file /home/doemer/EECS222A_F07/`mp3_v1.tar.gz.`
Untar this file using the command:

```
gtar xvzf mp3_v1.tar.gz
```

A directory by name mp3_v1 will be created. Change into this directory

```
cd mp3_v1
```

The entire MP3 source code is in single file `mp3decoder.sc`. There is a `Makefile` to compile and test the decoder for a set of MP3 streams. There are two directories `reference/` and `testStream/` which you don't have to worry about at this stage.

You need to set the path for the SpecC compiler. Source the setup shell script as below:

```
source /opt/sce-20060301/bin/setup.csh
```

Now compile and test the decoder by running following commands

```
make clean
make
make test
```

The setup should compile and simulate without errors. Please just ignore any `"Can't step back"` and "`read length less than max`" messages.

Given MP3 Code

The MP3 code is a basic SpecC code. It has 4 behaviors: Stimulus [at line 5724], Monitor [at line 5692], DUT [at line 5640] and MP3Decoder [at line 2755]. Use text editor to view the source code and find these behaviors.

We want to introduce more behaviors in MP3Decoder to enable exploration. In the next section, we describe the changes you have to do to convert a function call into a behavior. Note that the line numbers we refer to are the unmodified lines in the original source file.

Task 1: Encapsulate **`decodeMP3`** function call in behavior **`MP3Decoder`**.

1. We will give a set of instructions to convert a function to behavior. We would also like to measure the time it takes to perform this conversion.
   Please make a note of the start time T0.
2. `decodeMP3()` is a function in behavior `MP3Decoder`. The global function is defined at line 2873 and is declared at 2724. There are 2 calls to this function, first call at line 2778 and second at line 2792. For this exercise, we are only interested in encapsulating the function call at line 2778.
3. First we have to create the behavior body. Create a new empty behavior (say at line 2754) with the signature:

```
behavior B_decodeMP3(
    in struct mpstr *mp,
    in char *input,
    in int isize,
    in char *output,
    in int osize,
    inout int *done,
    out int ret_port) {

};
```

   Note that this signature can be derived from the function signature. You could copy the function signature of decodeMP3 and modify it, or copy this signature from this document directly, or else just type it.

4. Copy the global function `decodeMP3()` into behavior B_decodeMP3 (retain the global function, don't delete it).
5. In the behavior `B_decodeMP3` create an empty main function `void main (void){}`
6. Add this function call in the empty main function
   `ret_port = decodeMP3(mp, input, isize, output, osize, done);`
7. Save the file and do: `make` and `make test` and check if the tests run fine.
8. **Note the time T1.**

9. Now we have to create the instance of this newly created behavior in the parent behavior. First, create new temporary variables in the scope of behavior MP3Decoder. We explain later the need for these variables.

```
struct mpstr *t_dummy;
char *t_dummy_0;
char *t_dummy_1;
int *t_dummy_2;
```

10. Create an instance of the new behavior in the parent behavior MP3Decoder before the main body.
    `B_decodeMP3 I_B_decodeMP3 (t_dummy, t_dummy_0, len, t_dummy_1, (8192), t_dummy_2, ret);`
    The portmap needed for creating the instance is obtained from the function call. The function call is: `ret = decodeMP3( &mp, buf, len, output, 8192, &size);`
    Since the portmaps can only be variables (not expressions), temporary variables are used to store the argument expressions. So, before we make a call to the instance, we need to initialize these temporary variables with the argument expressions.
11. Replace the first call to function `decodeMP3()` in the behavior `MP3Decoder` with the call to the instance

```
I_B_decodeMP3.main()
```

12. Add following assignment statements to initialize the temporary variables in the behavior `MP3Decoder` before the call to the instance.

```
t_dummy =  &mp;
t_dummy_0 = buf;
t_dummy_1 = output;
t_dummy_2 =  &size;
```

13. Save the file and do: `make` and `make test` and check if the tests run fine.
14. **Make a note of the time T2**

Task 2: Encapsulate **`do_layer3`** function call in behavior **`B_decodeMP3`**

1. For performing this task, unlike the previous case, we will not give the detailed directions. We will briefly outline the steps needed. You will use the source file resulting from the changes done in the previous step.
   Please make a note of the start time T3.
2. You have to encapsulate the function call `do_layer3()` which is located in the member function decodeMP3() which is a member of the behavior `B_decodeMP3`.
3. First create the behavior signature with an empty body. Use the behavior name `B_do_layer3`. To derive the signature you can use the function call signature of `do_layer3()`. To determine the direction of the ports (`IN, OUT, INOUT`) you have to analyze the program and determine the accesses. Since this is complicated you could skip this and not specify any port direction. By default these ports will be treated as `INOUT`.
4. Copy the do_layer3() function to the behavior body and add the function call to the `do_layer3()` function from the `main(),` just like the previous example.
5. Save the file and do `make` and `make test` and check if the tests run fine.
6. **Note the time T4.**

7. Now create the instance of the behavior `B_do_layer3` in the parent behavior `B_decodeMP3`. Use the temporary variables if you need for the port map. Replace the call to function `do_layer3` (which is located in the member function `B_decodeMP3::decodeMP3`) with the call to the instance. If you have used temporary variables, then introduce the necessary assignment statements before the instance call.
8. Save the file and do `make` and `make test` and check if the tests run fine.
9. **Make a note of the time T5**

Deliverables:

- 1-paragraph description about the two tasks above
  (i.e. how far you got, what were the problems, how did you solve it)
- Please also report the times T0, T1, T2, T3, T4 and T5 (in minutes).

Due: Week 7 (Nov 9, 2007)

--
Rainer Doemer (ET 444C, x4-9007, doemer@uci.edu)