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

Assignment 5

Posted: November 9, 2007 (week 7)
Due: November 16, 2007 (week 8)

Tasks: Part 1: Creating Behaviors in C Code
Part 2: Pointer Elimination

Instructions: (by Pramod Chandraiah)

Part1 - Converting Statements to Behavior

In the previous assignment, we converted functions into behaviors. In this assignment, we will encapsulate a set of C statements into behaviors.

Encapsulating Statements in Behaviors

```
 behavior B ( in int p1, in int p2,  out int result)
{
    void main( )
     {
         int i1, a, b[10], s, *pa;
         a = p1+p2;
         s = p1-p2;
         pa = &s;
         …..
         result = f1(a, b[i1], pa);
         …..
     }
     int f1( int w, int x, int *p)
     { *p  = w+x+*p;
         return *p;
     }
};
```

(a) Original model (Model 1)

```
behavior  B_child1( in int p1, in  int p2, out int a, out int s) {
     void main()
     {  a = p1+p2;
        s = p1-p2;
     }
};

 behavior B (in int p1, in int p2, out int  result) {
    int a, s;
   //Instantiate child behavior here
    I_B_child1(p1, p2, a, s);
    void main( )
    {
         int i1, b[10], *pa;
         I_B_child1.main();
         pa = &s;
         …..
        result = f1(a, b[i1], pa);
         …..
    }
} ;
```

(b) Statements encapsulated in behavior (Model 2)

The idea behind encapsulating statements into behavior is same as that of encapsulating functions, i.e. to create a new computation block with explicit interface.
As shown in the figure above, encapsulating statements into a behavior involves creating a behavior body, creating the instance of the newly created behavior, and replacing the statements with the call to the instance. Note that, creating the behavior body requires analyzing expressions in the statements, determining their access type, and determining the port list. Creating the instance requires creating the port map list.

Initial Setup

The initial setup files are in the tar file `mp3_v2.tar.gz.` Untar this file using the command:

```
gtar xvzf mp3_v2.tar.gz
```

A directory by name mp3_v2 will be created. Change into this directory

```
cd mp3_v2
```

You will see following files:

`mp3decoder_2.sc` – Complete source of a floating point MP3 decoder (same as the one provided for previous assignment but with two new behaviors).

`Makefile` – To compile and test the sources.

`mp3_fixpt.sc` – Complete source of fix-point MP3 decoder (This is needed for part-2 of the assignment)

`huffman.c, huffman.h` – Needed for fix point MP3 decoder. You can ignore these files as their inclusion and compilation is taken care in the `Makefile`.

There are 3 directories (`reference/, reference-fix/, testStream/`) that contain test streams and reference output. You don't have to worry about these 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. (ignore `"Can't step back"` and "`read length less than max`" messages)

Given MP3 Code

The MP3 code is a basic SpecC code. It has 7 behaviors: Stimulus [at line 3179], Monitor [at line 3147], DUT [at line 3095], MP3Decoder [at line 3041], B_decodeMP3 [at line 2952], B_do_layer3 [at line 2770] and behavior Main [at line 3221]. Use text editor to view the source code and find these behaviors. We want to introduce more behaviors in B_do_layer3. In the next section, we describe the changes you have to do to encapsulate statements into behavior. Note that the line numbers we specify refer to the unmodified original source file (`mp3decoder_2.sc`).

Task 1: Encapsulate Statements in lines 2792-2823 in behavior **`B_do_layer3`**.

1. We will give a set of instructions to encapsulate these statements into 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. `do_layer3()` is a function in behavior `B_do_layer3`. This function is defined at line 2776. We will encapsulate the statements from line 2792 to line 2823 (inclusive).
   First we have to create the behavior body. Create a new empty behavior (say at line 2769) with the signature:

```
behavior Bchild1_B_do_layer3(
    inout int stereo,
    in struct frame *fr,
    inout int sfreq,
    inout int single,
    inout int stereo1,
    inout int ms_stereo,
    inout int i_stereo,
    inout int granules,
    in struct III_sideinfo sideinfo)
    {
};
```

Note that this signature is obtained by analyzing the lines 2792 – 2823 and determining the variables and their access types.

3. In the behavior `B_decodeMP3` create an empty main function `void main (void){}`
4. Copy the C statements (2792-2823) into the `main( )` of the `Bchild1_B_do_layer3`.
5. Save the file and do: `make` and `make test` and check if the tests run fine.
6. **Note the time T1.**

7. Now we have to create the instance of this newly created behavior in the parent behavior. First, we have to move any local variables accessed by the statements into parent behavior's scope.
8. Move variables, `stereo, stereo1, single, i_stereo, ms_stereo, sfreq, sideinfo, granules` into parent behavior `B_do_layer3`. These variables are declared at the beginning of the function `do_layer3`. After moving change the name to: `R_stereo, R_stereo1, R_single, R_i_stereo, R_ms_stereo, R_sfreq, R_sideinfo, R_granules` so that there are no naming clashes.

9. Create an instance of the new behavior in the parent behavior `B_do_layer3` after the above variable declaration and before the `main` body.
   ```
   Bchild1_B_do_layer3 I_Bchild1_B_do_layer3(R_stereo, fr, R_sfreq,
   R_single, R_stereo1, R_ms_stereo, R_i_stereo, R_granules,
   R_sideinfo);
   ```

   The portmap needed for creating the instance is obtained from the previous analysis of the statements function call.
10. Now that you have renamed some of the relocated variables, you have to change the references to the old names with the new names. There are 7 references to variable `sideinfo` in the function `do_layer3( ),` change them to use the new name `R_sideinfo.` There is 1 reference to `granules,` change it to `R_granules.` Similarly, rename the access to other variables `sfreq, stereo, stereo1, ms_stereo, i_stereo, granules`.
11. Replace the statements with the call to the instance
    ```
    I_B_decodeMP3.main ()
    ```
12. Save the file and do: `make` and `make test` and check if the tests run fine.
13. **Make a note of the time T2**

Task 2: Encapsulate statements in lines 2863 to 2887 in behavior **`B_do_layer3`**

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 statements from line 2863 to 2887 (both lines inclusive). The lines refer to the original final `mp3decoder_2.sc` and are located in the function `do_layer3 ( )` which is a member of the behavior `B_do_layer3`. These will be approximately located at lines 2896 to 2921 in the file that was saved in step-11 in previous section.
3. First create the behavior signature with an empty body. Use the behavior name `B_child2do_layer3`. To derive the signature, analyze the statements and determine the variables and their accesses. To determine the direction of the ports (`IN, OUT, INOUT`) you have to analyze the program and determine the accesses. It is recommended to determine these directions. However, if it is too complicated for you, you can skip this and not specify any port direction. By default these ports will be treated as `INOUT`.
4. Copy the statements into the `main` of the `B_child2do_layer3` behavior, 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_child2do_layer3` in the parent behavior `B_do_layer3`. If necessary, move the variables into the scope of the behavior. Replace the statements with the call to the newly created instance.
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

At the end, please report the times **T0, T1, T2, T3, T4** and **T5.**

Part2 - Pointer Replacement

As we know, pointers in the C code create ambiguity and make the code unanalyzable and unsynthesizable by automatic tools. In this part, you will see how to replace indirect variable access through pointers with direct variable access.

Pointer re-coding example

```
1.  int a[50], ab[50][16];
2.  int v1, v2, x, y;
3.  int *p1,*p2, *p3, *p4, (*p5)[16], p6;

4.  p1 = &x;
5.  *p1 = y+1;
6.  if(condition) p2 = &v1;
7.  else p2 = &v2;
8.  *p2 = 5;
9.  p3 = &ab[40][10];
10. *p3 = 100;
11. p4 = a;
12. p4++;
13. *p4++ = 1;
14. p5 = &ab[5];
15. p6 = p4+v1;
```

(a) Code with pointers

```
1.  int a[50], ab[50][16];
2.  int v1, v2, x, y;
3.  int ip3, ip4, ip5, ip6;

4.  //Nothing here
5.  x =y+1;
6.  if(condition) p2 = &v1;
7.  else p2 = &v2;
8.  *p2 = 5;
9.  ip3 =10;
10. ab[40][ip3] = 100;
11. ip4 = 0;
12. ip4++;
13. a[ip4++] = 1;
14. ip5 = 5;
15. ip6 = ip4+v1;
```

(b) Code with p1, p3, p4, p5, p6 substituted

Pointer re-coding is a 2 step process. First, you have to determine the variable to which the pointer points to and second, replace the pointer accesses with the direct access to the target variable. The figure above shows some examples of pointer recoding. Note that pointers to the scalar variables are completely removed, and in place of pointer to the arrays, integer variables that act as indices into the array are created (e.g. line3). Expressions of the pointers that point to arrays are replaced with the array access expression formed by the actual variable the newly created index variable (e.g. line 10). Pointer `p2` is not recoded as it could point to more than 1 variable.

Initial Setup

You will use the same set up as the part-1 of this assignment, except that you will use fix-point MP3 decoder implementation which is contained in the single source file `mp3_fixpt.sc.`
As usual, 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 (note that commands are different from before)

```
make clean
make all_fix
make test_fix
```

Given Fix-point MP3 Code

This is a fix-point MP3 implementation in SpecC. It has many behaviors, including `Mad_Stimulus` [at line 13681], `Mad_Monitor` [at line 13533], `MP3Decoder` [at line 13717] and behavior `Main` [at line 13746]. Use text editor to view the source code and find these behaviors. We will replace couple of pointers in the behavior `Calc_sample` located at line 13177. In the next section, we describe the changes you have to do to replace a pointer. Note that the line numbers we refer to are the unmodified lines in the original source file (`mp3_fixpt.sc`).

Task 3: Replace pointer **`fe`** in behavior **`Calc_sample`**

1. We will now give a set of instructions to replace the pointer expressions. We would also like to measure the time it takes to perform this conversion.
   Please make a note of the start time T0.
2. Pointer `int (*fe)[8]` is declared at line 13202 in the `main` body of `Calc_sample` behavior and this pointer points to multi-dimensional array `filter[2][2][16][8]` which is a port of the behavior. More specifically, `fe` points to the dimension `(filter)[0][phase & 1].`
3. We have to replace all accesses to `fe` with the direct access to the array variable `filter`. As a first step, create an integer variable which acts as an index variable (`i_fe`) into the array in place of the pointer `fe`. You can remove the pointer declaration.
4. Replace the pointer initialization statement (`fe = &(filter)[0][phase & 1][0];`) at line 13214 with the initialization of the index variable `i_fe`. (`i_fe = 0`).
5. Replace the expressions involving `fe` with the direct access to `filter.` For example, replace the expression at line 13229 which is :
   ```
   ((lo) += (( *fe)[0]) * (ptr[0]));
   ((lo) += (( *fe)[1]) * (ptr[14]));
   ```
   with
   ```
   ((lo) += (((filter)[0][phase & 1][i_fe])[0]) * (ptr[0]));
   ((lo) += (((filter)[0][phase & 1][i_fe])[1]) * (ptr[14]));
   ```
   and so on…
6. Similarly, replace all the other expression involving `fe` in the `main()`body.
7. Arithmetic Expressions involving `fe` such as the one at line 13242:
   ```
   ++fe;
   ```
   is replaced with
   ```
   ++i_fe;
   ```
8. Save the file and do: `make all_fix` and `make test_fix` and check if the tests run fine.
9. **Make a note of the time T2**

Task 4: Replace pointer **`fx`**, **`fo`** and **`Dptr`** in behavior **`Calc_sample`**

1. We will note give complete instructions to recode these pointers. These pointers can be recoded on similar lines as pointer `fe.`
2. Pointer `fx` is declared at line 13204 and its initialization is at line 13215. Note that this pointer points to dimension `(filter)[0][ ~phase & 1]`.
3. Replace this pointer and make a note of the start and the end time to perform the task **(T3, T4)**

4. Pointer `fo` is declared at line 13203 and its initialization is at line 13216. Note that this pointer points to dimension `(filter)[1][ ~phase & 1]`.
5. Replace this pointer and make a note of the start and the end time to perform the task **(T5, T6)**

6. Pointer `Dptr` is declared at line 13205 and its initialization is at line 13217. Note that this pointer points to array `D`.
7. Replace this pointer and make a note of the start and the end time to perform the task **(T6, T7)**

At the end, please report the times **T0** trough **T7.**

Deliverables:

- 1-paragraph description about each of the tasks above
  (i.e. how far you got, what were the problems, how did you solve it)
- Please also report the times for part 1 (T0 to T5, in minutes), and for part 2 (T0 to T7, in minutes).

Due: Week 8 (Nov 16, 2007)

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