

Automatic Re-Coding of Reference Code into Structured and Analyzable Models

Pramod Chandraiah and Rainer Doemer

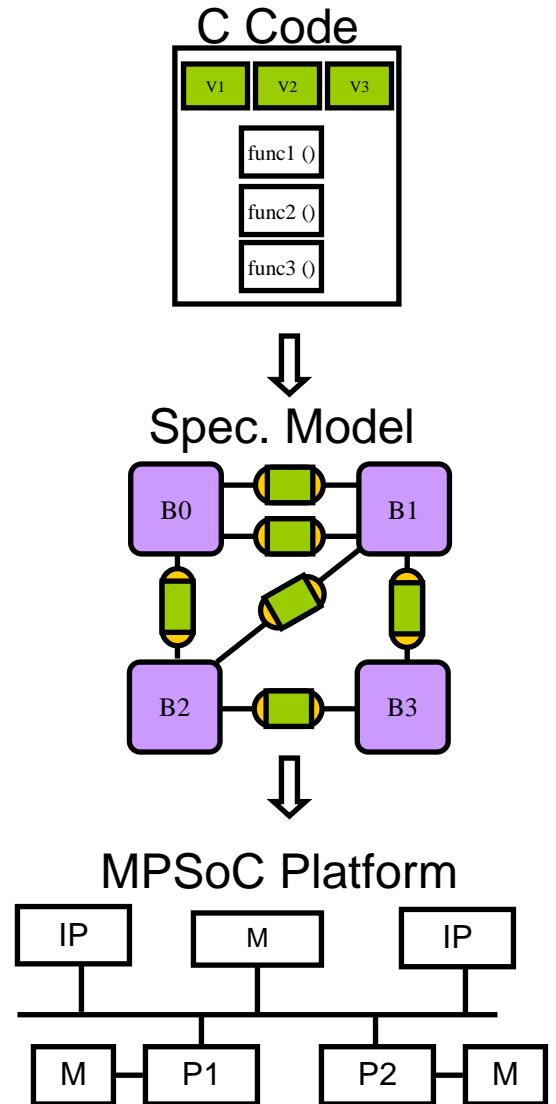
Center for Embedded Computer Systems

University of California, Irvine



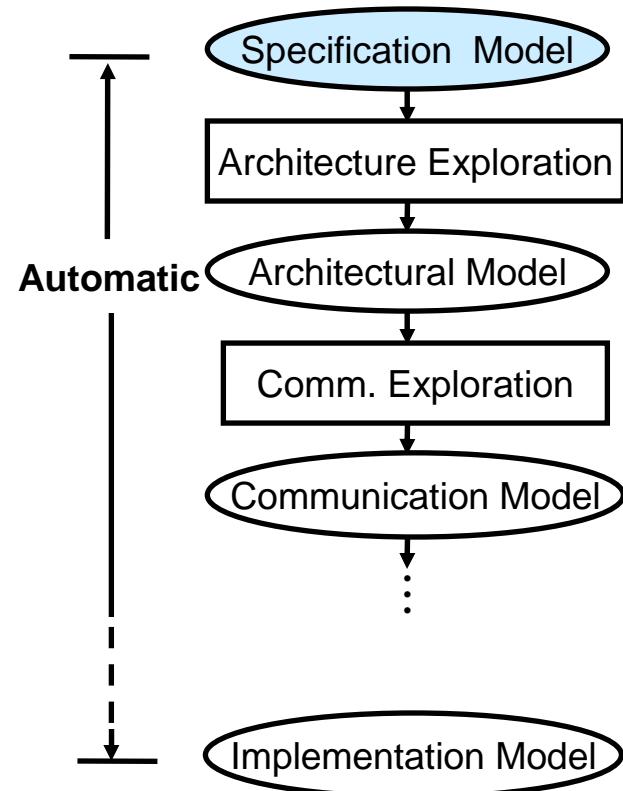
Introduction

- System level design
 - Specification to implementation
 - Needs system model
- Specification model
 - Starting point of the design
 - Structured and Analyzable model
 - Explicit computation & communication
 - Analyzable, synthesizable, verifiable
- C reference code
 - Actual starting point of design
 - Ambiguous
 - Non-definite computation & communication
 - Insufficient for Architectural Exploration
- Re-Coding C code to specification model
 - Creating structural hierarchy
 - Creating Analyzable interface for behavioral blocks



Motivation

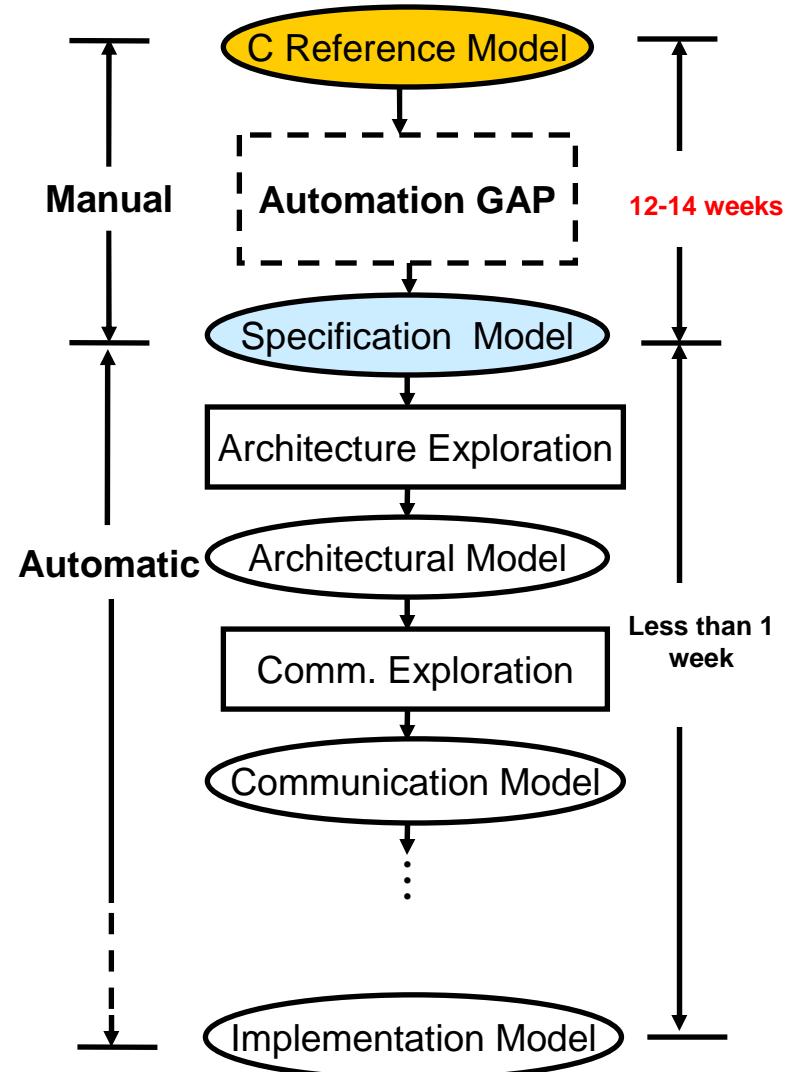
- SpecC design methodology
 - Starts from Specification Model
 - Architectural exploration
 - Communication exploration
 - ...
- Architectural exploration
 - Code partitioning
 - Grouping/Re-grouping behaviors
 - Mapping behaviors to PEs
- Specification Model
 - Behaviors are the basic unit of computation
 - All the explorations (partitioning and mapping) happen at the granularity specified by the behaviors



Source: *System Design: A practical guide with SpecC*

Motivation

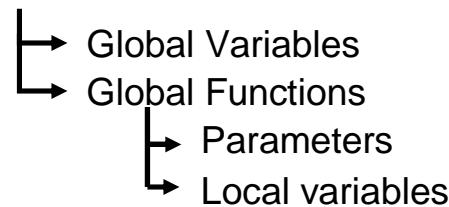
- Absence of structural hierarchy limit the effectiveness of system design tools
 - Architecture exploration tools
 - Code partitioning and mapping
 - Analyzability
 - System level performance analysis
- Manual re-coding of C into SoC model in SLDL is time consuming
- **Proposed Solution: Automatic behavior creation**
 - Enables automatic architectural exploration



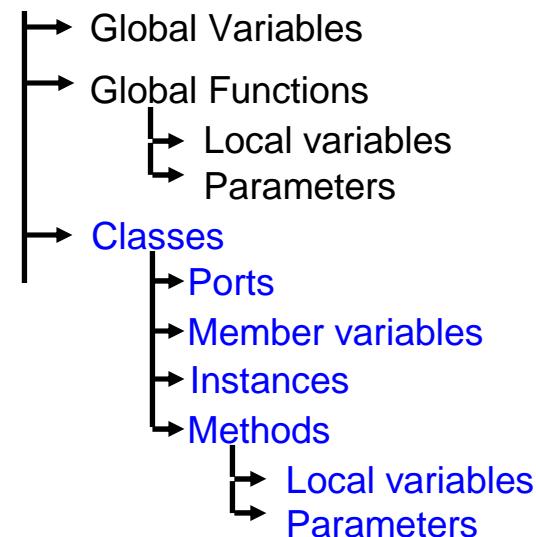
Source: *System Design: A practical guide with SpecC*

Advantages of behaviors

- Explicit interface
 - Ports
 - Name, type and direction
 - Port map
 - Mapping known at compile time
- Additional level of Scope
 - C has only 2 levels
 - Global
 - Local
 - Behaviors in SLDL provide 3 levels
 - Global
 - Local
 - Class



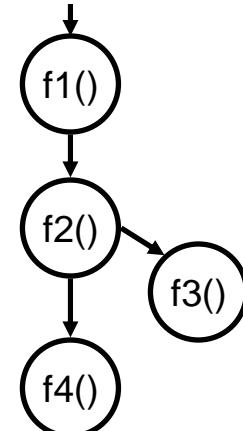
Syntactical hierarchy in C code



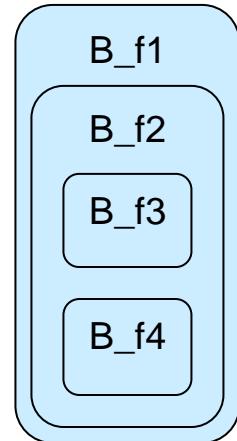
Syntactical hierarchy in SLDL code

Creating Structural Hierarchy

- Use the partial structure available in the form of functions
- Encapsulating C code blocks
 - Determining the statically analyzable interface
 - Re-coding to encapsulate the code in a behavior
 - Instantiation of behavior



Function call hierarchy



Structural hierarchy

Analyzable Interface

- Accumulated accesses to variable in the code block
 - In case of functions, function parameters, return variables
 - In case of statements, variables in the code block, excluding the local variables within the block
- Cumulative Access Types (CATs)
 - Read ($a, i, b[10]$)
 - Write (c)
 - Read-Write (s)
- Safe assumption made about arrays
 - As in case of array b as the value of i is not statically known

```
int func( int w, int x, int *p)
{ *p = w+x+*p}

pointer = &s
c= func (a, b[i], pointer)
```

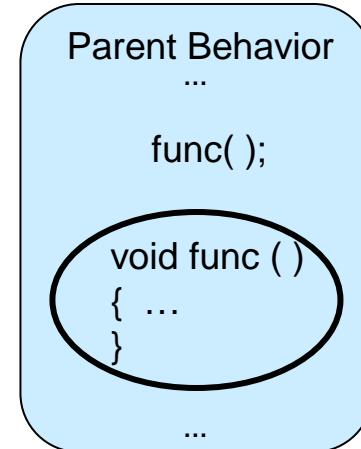
Code Snippet

$I: \{R: a, R: b[10], R: i, RW: s, W: c\}$

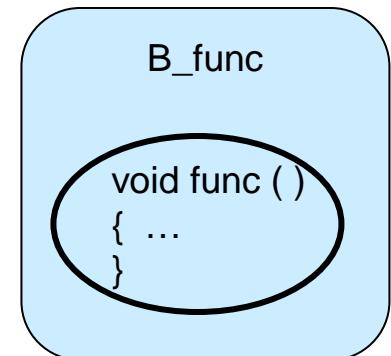
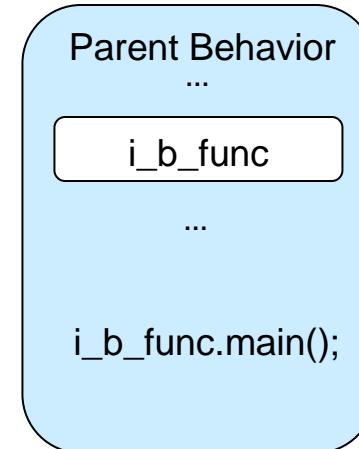
Interface of $func()$

Encapsulation and Instantiation

- Function encapsulation
 - Create port list from the interface
 - Read access results in *in* port
 - Write access results in *out* port
 - *RW* access results in *inout* port
 - Return variable becomes an *out* port
 - Create behavior body
- Behavior Instantiation
 - Create port map
 - From function arguments
 - Return value
 - Replace function call with instance call



Initial structure with global function



Global function encapsulated in behavior
(syntactical structure)

Re-coding Complexities

- Variables used in the port map must be made available in the class scope
 - Example, *a*, *b*, *i1*, *s*
- Need to handle the difference in semantics of functions and behaviors
 - Especially, Call-by-value semantics
 - Call-by-value parameter can only be an *in* port

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

Initial code with function *f1()*

```
behavior B (in int p1, in int p2, out int result) {
    int a, b[10], i1, s;

    I_B_f1(a, b, i1, s, result);
    void main( )
    {
        int *pa;
        a = p1+p2;
        s = p1-p2;
        pa = &s;
        ....
        I_B_f1.main();
        ....
    }
};
```

f1() replaced with instance call

Re-coding Complexities

- Implicit function return value
 - Example, function call in the conditional expression of an *if* statement
 - Explicit return value is introduced
- Evaluation of expressions in function arguments
 - Example, *w+x* must be either evaluated first or inside the behavior

```
1. int func (int, int, int);  
2. /*...*/  
3. if (func (w+x, y, z))  
4. { /* ... */  
5. }
```

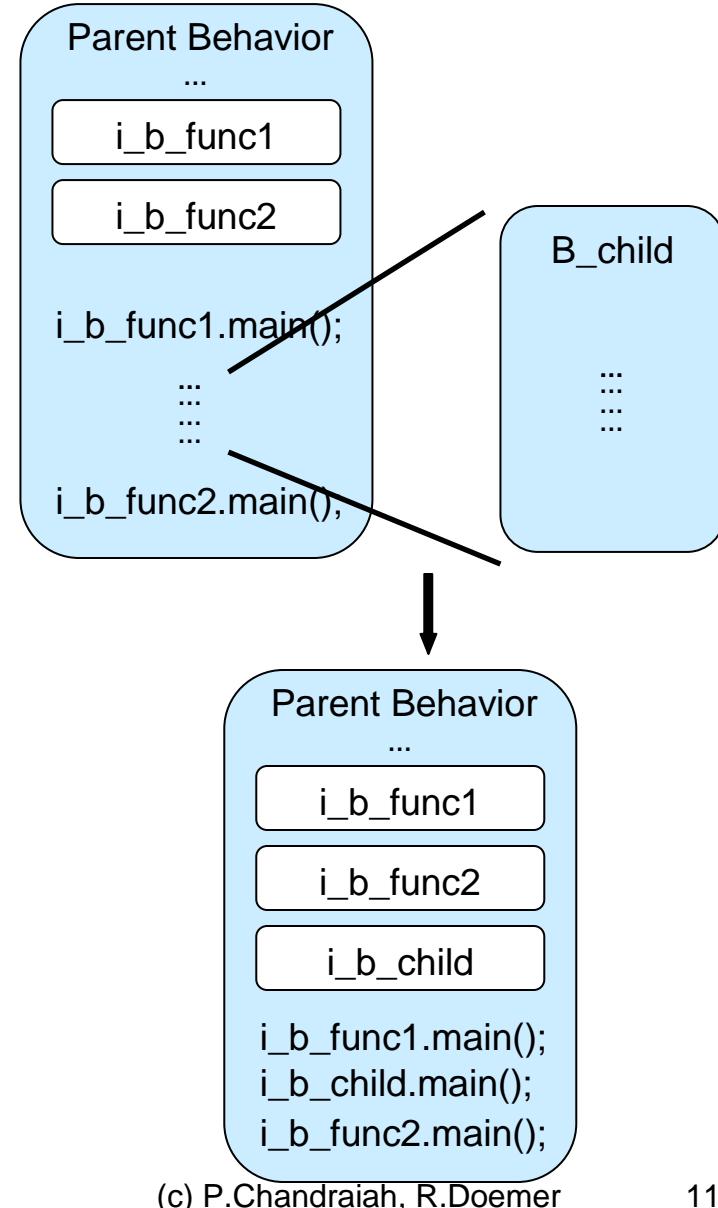
Initial code with function *func()*

```
1. behavior B_func (in int, in int, in int, out int);  
2. /*...*/  
3. int wx, retval;  
4. B_func I_B_func (wx, y, z, retval); //Instance  
5. /*...*/  
6. wx = w+x;  
7. I_B_func.main();  
8. if (retval)  
9. { /* ... */  
10. }
```

Code after replacing *func()* with behavior

Encapsulating Statements

- Encapsulating functions works on the function calls
- The statements that exist between function calls must be encapsulated, as well



Establishing connectivity

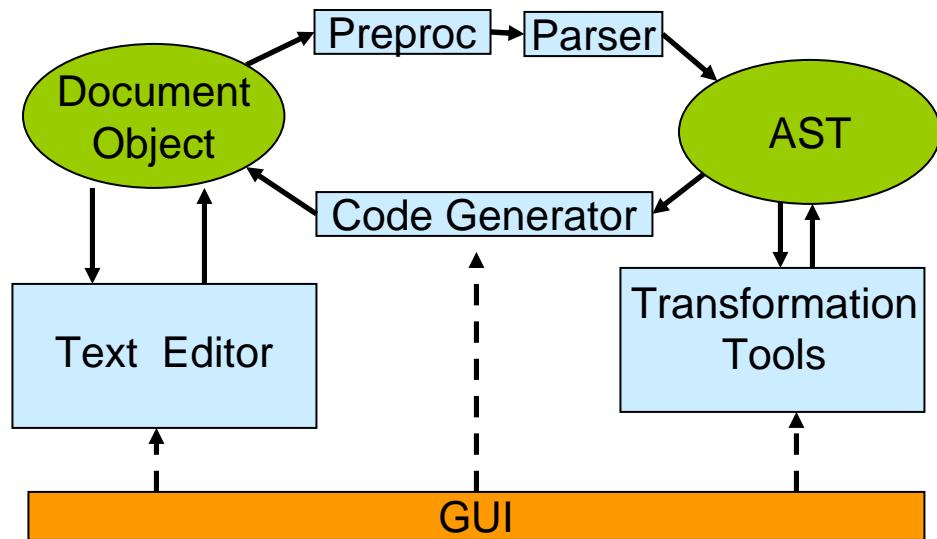
- After encapsulating functions, global variables must be localized to the behavioral partitions
- Procedure
 - Finding the global variable
 - Determining the functions and behaviors accessing it
 - Migrate it to the lowest most common parent behavior
 - Providing access to the new localized variables by inserting ports

Restrictions

- Encapsulating functions applies only to internal functions
- Encapsulating statements not allowed in presence of *goto* statements

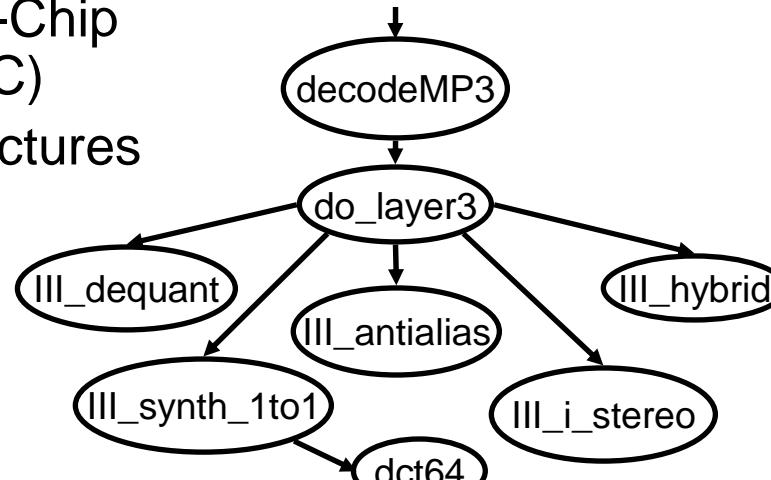
Interactive Source Re-coder

- Controlled interactive approach
- Automatic programming
- It's a union of
 - Textual Editor
 - Abstract Syntax Tree
 - Preprocessor/Parser
 - Transformation and analysis functions
 - Code generator
- Main Characteristics
 - Computes points-to information with a click
 - Re-codes pointers with a click
 - Applies Source-to-Source transformations on-the-fly

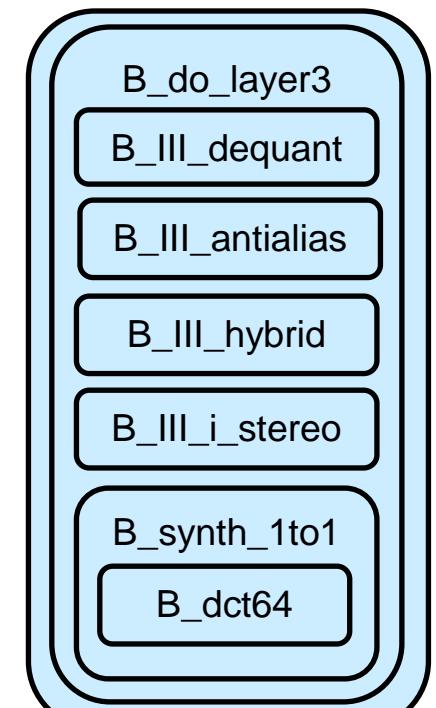


Experiments

- Specification model of MP3 Decoder
 - Input C code had 30 functions
 - 43 behaviors were introduced
- Design Explorations
 - Using System-on-Chip Environment (SEC)
 - 6 different architectures



Partial function hierarchy in MP3 code



Structural hierarchy

Gains

- Design examples
 - JPEG, MP3, GSM
- Introduced behaviors
 - Manual
 - Source Re-Coder
- Manual time
 - Created manually by different designers
- Re-coding time
 - Time to implement the transformations using Source Re-coder
 - In the order of minutes
- **Significant productivity gains**

Properties	JPEG	Float-MP3	Fix-MP3	GSM
Lines of C code	1K	3K	10K	10K
C Functions	32	30	67	163
Lines of SpecC code	1.6K	7K	13K	7K
Behaviors created	28	43	54	70
Re-Coding time	≈ 30 mins	≈ 35 mins	≈ 40 mins	≈ 50 mins
Manual time	1.5 weeks	3 weeks	2 weeks	4 weeks
Productivity gain	120	205	120	192

Source Re-coder

CUTE: /home/pramodc/work/project/cars/cute1/cute_spec/temp/mp3_v3/mp3decoder.sc

File Edit View Search Project Tools Options Language Extra Window Help

Transformation and Analysis tools

Side Panel

mp3decoder.sc

```
2755 behavior MP3Decoder (
2756   .i_receiver:filebuf,
2757   .i_sender:send_data)
2758   {
2759     .int:size;
2760     .char:output[8192];
2761     .int:len;
2762     .int:offset;
2763     .int:ret;
2764     .struct:mpstr:mp;
2765     .unsigned:char*:conv16to8;
2766
2767     void:main()
2768     {
2769       →Init();
2770       →Initcc();
2771       →Init_pnts();
2772       →InitMP3(&mp);
2773       →while(1)
2774       →{
2775         .filebuf.receive(buf, 16384);
2776         .len = bytesRead;
2777         .ret = decodeMP3(&mp, buf, len, output, 8192, &size);
2778         →while(ret == 0)
2779         →{
2780           if(((int)size / 256) * 256 != size)
```

Function 2 Behavior

ret = decodeMP3(...)

SpecOut Messages Stdout Stderr Shell Search Tags

Symbol is decodeMP3 – at 2777::12

Statement is ret = decodeMP3(&mp, buf, len, output, 8192, &size); – at 2777::12

Message Panel

line: 2777 col: 15 Converts function 2 behavior File: /home/pramodc/work/project/cars/cute1/cute_spec/temp/mp3_v3/mp3decoder.sc (S) T. Chandran, K. Decker

Source Re-coder

The screenshot shows the CUTE (Cute Embedded Toolkit) IDE interface. The main window displays a source code file named `mp3decoder.sc`. The code is a statechart-like script with several states and transitions. A yellow arrow points from the text "Statement 2 Behavior" to the transition labeled "Statement 2 Behavior" in the toolbar. Another yellow arrow points from the text "Code selection before invoking Statement 2 Behavior" to the selected code block in the editor. The message panel at the bottom shows the symbol `decodeMP3` and its statement definition.

```
2790 → int stereo1;
2791
2792 → stereo = fr->stereo;
2793 → sfreq = fr->sampling_frequency;
2794 → single = fr->single;
2795 → if (stereo == 1)
2796 → {
2797 →   stereo1 = 1;
2798 →   single = 0;
2799 → }
2800 → else
2801 →   if (single >= 0)
2802 →     stereo1 = 1;
2803 →   else
2804 →     stereo1 = 2;
2805 → if (fr->mode == 1)
2806 → {
2807 →   ms_stereo = fr->mode_ext & 2;
2808 →   i_stereo = fr->mode_ext & 1;
2809 → }
2810 → else
2811 →   ms_stereo = i_stereo >= 0;
2812 → if (fr->lzf)
2813 → {
2814 →   granules = 1;
2815 →   III_get_side_info_2(&sideinfo, stereo, ms_stereo, sfreq,
2816 →   single);
2817 → }
2818 → else
2819 → {
2820 →   granules = 2;
2821 →   III_get_side_info_1(&sideinfo, stereo, ms_stereo, sfreq,
2822 →   single);
2823 → }
2824 → if (lzf)
2825 →   ms_stereo = min(granules, 1);
```

Statement 2 Behavior

Code selection before invoking Statement 2 Behavior

Symbol is decodeMP3 – at 2777::12
Statement is ret = decodeMP3(&mp, buf, len, output, 8192, &size); – at 2777::12

SpecOut Messages Stdout Stderr Shell Search Tags

line: 2821 col: 1 Converts statement 2 behavior File: /home/pramodc/work/project/cars/cutel/cute_speccc/temp/mp3_v3/mp3decoder.sc

Conclusions

- SoC specification
 - Structured, Analyzable, and synthesizable
 - C Reference Models act as starting point
- C code to Structured SoC specification
 - Time-consuming and error-prone re-coding
- Creating structural hierarchy
 - Automatic creation of behaviors from C functions and statements
 - Function-call chain transformed to behavioral hierarchy
- Interactive Source Re-coder
 - Designer-in-the-loop approach to recode C into SoC specification
 - Programming using source-level transformations
 - Integrates automatic behavior creation
- Compared to manual programming, source re-coder results in significant productivity gains
- Facilitates architectural exploration by enabling easy partitioning of code
- Enables faster path to the implementation

References

- S. Abdi, J. Peng, H. Yu, D. Shin, A. Gerstlauer, R. D'omer, and D. Gajski. System-on-chip environment (SCE version 2.2.0 beta): Tutorial. Technical Report CECS-TR-03-41, CECS, University of California, Irvine, 2003
- F. Ghenassia. *Transaction-Level Modeling with SystemC : TLM Concepts and Applications for Embedded Systems*. Springer- Verlag, 2006.
- Sprint parallelizes real life applications for embedded systems. <http://www.imec.be/design/sprint/>.
- A. Pimentel, L.O.Hertzberger, P. Lieverse, and P. Wolf. Exploring embedded-systems architectures with artemis. *IEEE Transactions on Computers*, 34(1), November 2001.
- A. Jerraya, H. Tenhunen, and W. Wolf. Guest editors' introduction: Multiprocessor systems-on-chips. *Computer*, 38(7):36.40, 2005
- A. Gerstlauer, R. D'omer, J. Peng, and D. D. Gajski. *System Design: A Practical Guide with SpecC*. Kluwer Academic Publishers, 2001.
- P. Chandraiah and R. D'omer. Specification and design of an mp3 audio decoder. Technical Report CECS-TR-05-04, CECS, University of California, Irvine, 2005.