Efficient Modeling of Embedded Systems using Computer-Aided Recoding

Rainer Dömer

doemer@uci.edu

With contributions by P. Chandraiah

Center for Embedded Computer Systems

University of California, Irvine

UCIrvine
University of California, Irvine

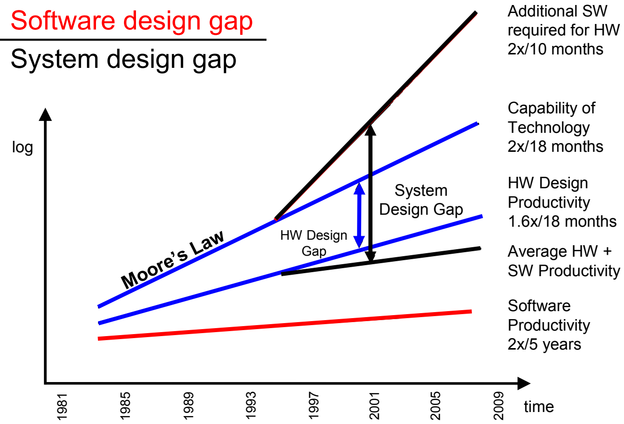
Outline

- Embedded System Design
- Computer-Aided Recoding
- Recoding Transformations
  - Creating structural hierarchy
  - Exposing potential parallelism
  - Creating explicit communication
  - Pointer recoding
- Interactive Source Recoder
- Experiments and Results
- Conclusions

Embedded System Design

- Productivity Gap

  Hardware design gap
  + Software design gap
  = System design gap



(source: *"Hardware-dependent Software"*, Ecker et al., 2009)

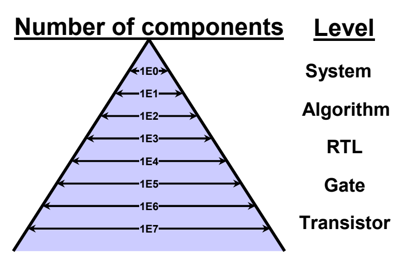
Tutorial at IESS 2009 R. Doemer 3

Embedded System Design

- How can we overcome the productivity gap?

  International Technology Roadmap for Semiconductors (ITRS) 2004:
  higher-level abstraction and specification is the first promising solution

- System Level Design
  - Unified HW and SW design
  - Higher level of abstraction
    - Fewer, more complex components
    - Maintain system overview
      - Without overwhelming details
    - Compose a system of algorithms
  - System Level Design Languages
    - SpecC [Gajski et. al, 2000]
    - SystemC [Groetker et. al, 2002]

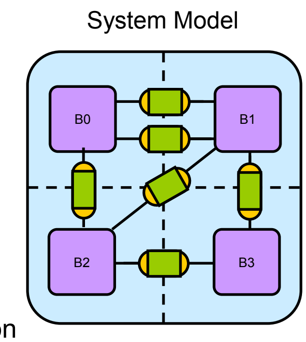


Source: "System Design: A Practical Guide with SpecC", 2001

Tutorial at IESS 2009 R. Doemer 4

Embedded System Design

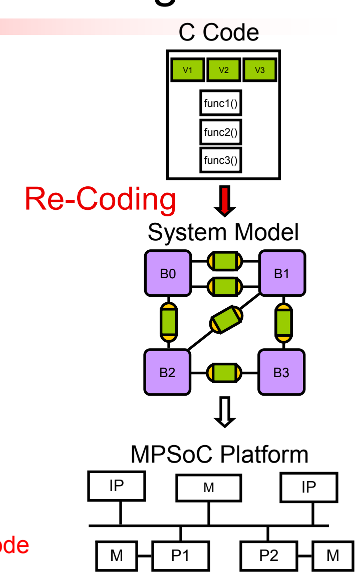
- System Level Modeling
  - Abstract description of a complete system
  - Hardware + Software
- Key Concepts in System Modeling
  - Explicit Structure
    - Block diagram structure
    - Connectivity through ports
  - Explicit Hierarchy
    - System composed of components
  - Explicit Concurrency
    - Potential for parallel execution
    - Potential for pipelined execution
  - Explicit Communication and Computation
    - Channels and Interfaces
    - Behaviors / Modules



Tutorial at IESS 2009 R. Doemer 5

Computer-Aided Recoding

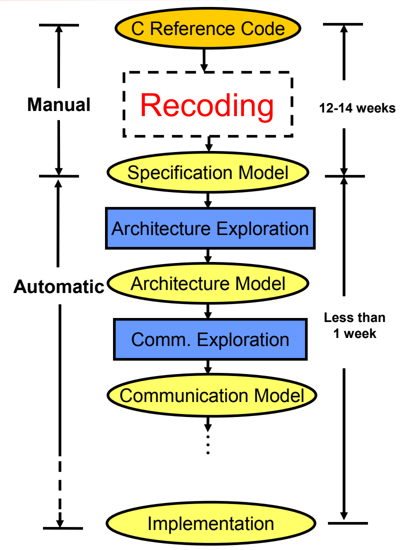
- Embedded System Design Flow
  - Input: System model
  - Output: MPSoC platform
- Actual Starting Point
  - C reference code
  - Flat, unstructured, sequential
  - Insufficient for system exploration
- Need: System Model
  - System-Level Description Language (SLDL)
  - Well-structured
    - Explicit computation, explicit communication
    - Potential parallelism explicitly exposed
  - Analyzable, synthesizable, verifiable
- Research: Automatic *Re-Coding*
  - How to get from flat and sequential C code to a flexible and parallel system model?



Tutorial at IESS 2009 R. Doemer 6

Motivation

- Extend of Automation
  - Refinement-based design flow
  - Automatic
    - Specification model down to implementation
    - Example: SCE (mostly automatic)
    - MP3 decoder: less than 1 week
  - Manual
    - C reference code to SpecC specification model
    - Source code transformations
    - MP3 decoder: 12-14 weeks!
- Automation Gap
  - 90% of overall design time is spent on re-coding!
- Proposal: Automatic Recoding

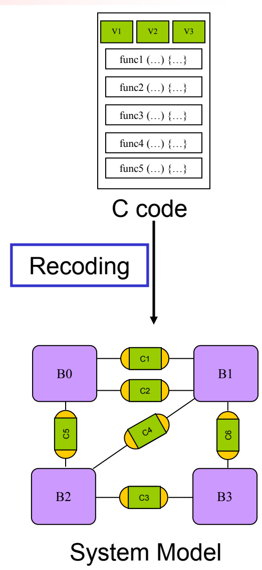


Source: *System Design: A Practical Guide with SpecC*

Tutorial at IESS 2009 R. Doemer 7

Problem Definition

- How to get from flat, sequential C code to a flexible, parallel system model?
- Recoding
  - Create structural hierarchy
  - Partition code and data
    - Expose concurrency (parallelize/pipeline)
  - Expose communication
  - Eliminate pointers
  - Make the code compliant to the design tools, …
- Our approach
  - Computer-Aided Recoding
    - Interactive source code transformations



Tutorial at IESS 2009 R. Doemer 8

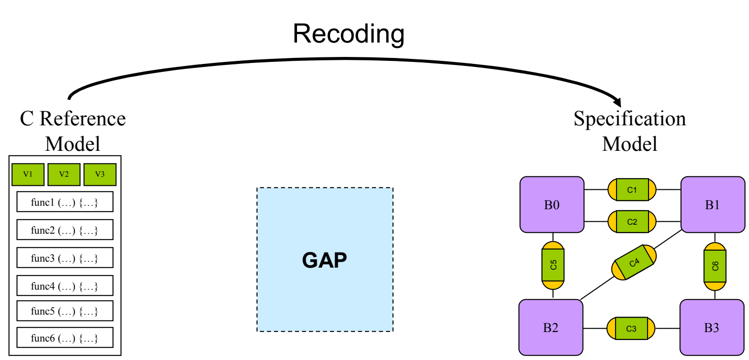
Computer-Aided Recoding

- Complete Automation is Infeasible!
  - Today's parallelizing compilers are largely ineffective
    - Heterogeneous architectures
    - Complexity of embedded applications
    - Hard problems (eliminating pointers, exposing parallelism, etc.)
  - Modeling requires understanding of the application
  - Recoding is not a monolithic transformation
    - Multiple transformations in application-specific order
- ➢ Interactive Approach
  - "Designer-in-the-loop"
  - Designer can utilize application knowledge
- *Designer-controlled* Transformations
  - Designer makes decisions
  - Tool automatically transforms the source code

Tutorial at IESS 2009 R. Doemer 9

Overcoming the Specification Gap

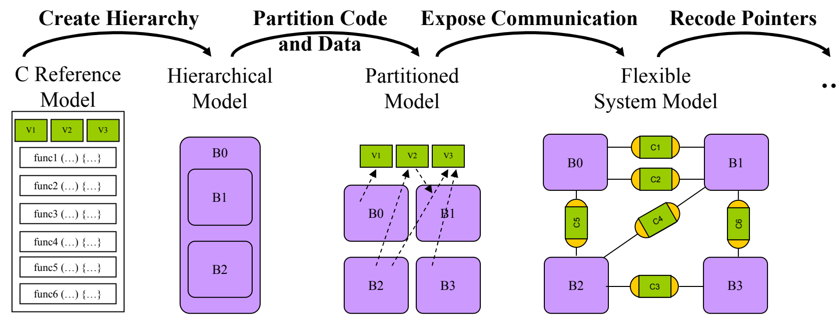
- Recoding Transformations



Tutorial at IESS 2009 R. Doemer 10

Overcoming the Specification Gap

- Recoding Transformations
  - Creating structural hierarchy [ASPDAC'08]
  - Code and data partitioning [DAC'07]
  - Creating explicit communication [ASPDAC'07]
  - Recode pointers [ISSS/CODES'07]

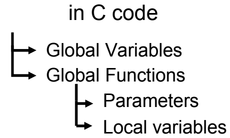


Tutorial at IESS 2009 R. Doemer 11

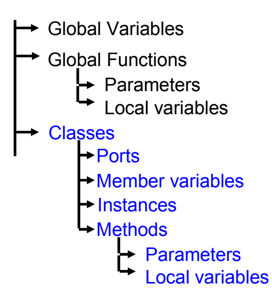
Creating Structural Hierarchy

- Goals
  - Separation of computation and communication
  - Explicit structure
  - Static connectivity (to enable/simplify analysis!)
- Modeling Hierarchy
  - Component blocks
    - Ports, data direction
  - Component instantiation
    - Port map, connectivity
- Describing Hierarchy
  - C code
    - Global scope
    - Local scope
  - SLDLs
    - Global scope
    - Local scope
    - Class scope

Syntactical hierarchy in C code



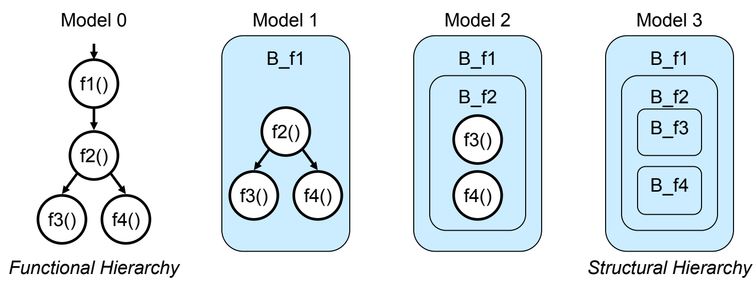
Syntactical hierarchy in SLDL code



Tutorial at IESS 2009 R. Doemer 12

Creating Structural Hierarchy

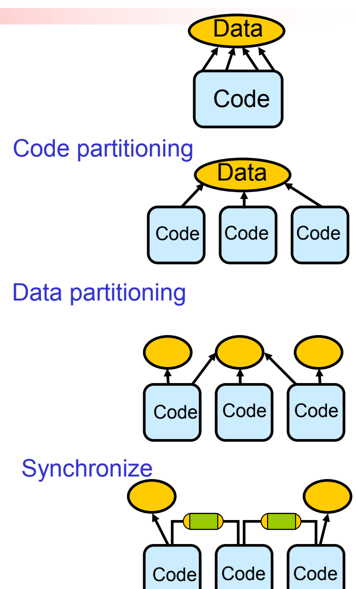
- Recoding
  - Convert functional hierarchy into structural hierarchy
  - Step-wise model transformation
  - Hierarchical encapsulation
    - Utilize given function call tree
    - Convert each function into a behavior
    - Start with root (i.e. `main()` function)
    - Continue step by step down to leafs



Tutorial at IESS 2009 R. Doemer 13

Exposing Potential Parallelism

- Desirable model features
  - Enable parallel execution
  - Allow mapping to different PEs
- Recoding tasks
  - Partition code
  - Partition data
  - Synchronize dependents
- Recoding transformations
  1. Loop splitting
  2. Cumulative Access Type analysis
  3. Partitioning of vector dependents
  4. Synchronizing dependent variables
  - [DAC'07, TCAD'08]

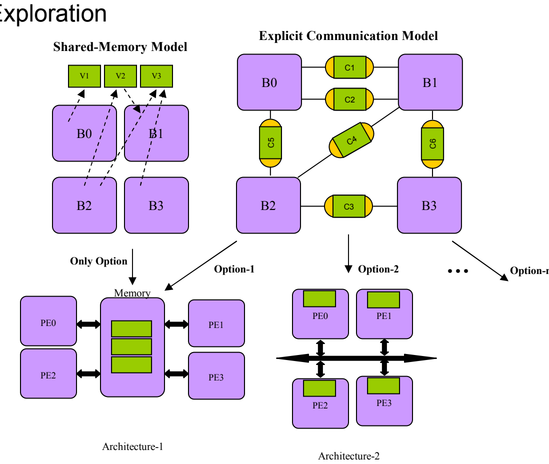


Tutorial at IESS 2009 R. Doemer 14

Exposing Communication

- Why create explicit communication?

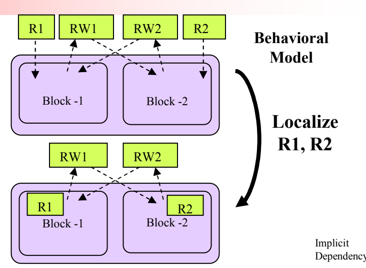
- Quality of Communication Exploration
  - Number of explorations
  - Extent of automation
  - Time
- Shared-Memory Model
  - Global variables limit the number of possible automatic explorations
- Explicit Communication Model
  - Enables automatic exploration of more design alternatives



Tutorial at IESS 2009 R. Doemer 15

Exposing Communication: 1. Localize

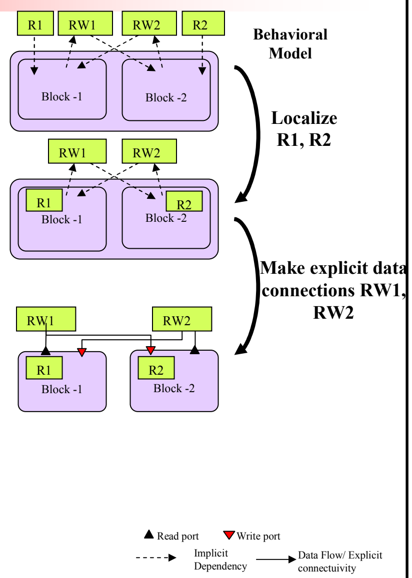
- Localize global variables to partitions
  - To enable multiple explorations
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If only one behavior is accessing it, migrate the variable into this behavior



Tutorial at IESS 2009 R. Doemer 16

Exposing Communication: 2. Expose

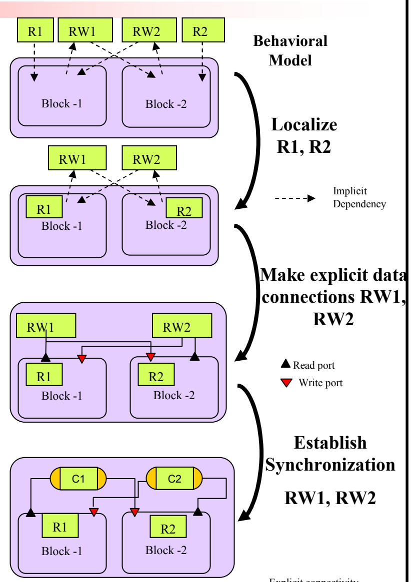
- Localize global variables to common parent and provide explicit access
  - Simplifies subsequent analysis of models
- Procedure
  - Find the global variable
  - Determine the functions and behaviors accessing it
  - If multiple behaviors are accessing it, find the lowest common parent
  - Migrate the variable to the parent
  - Provide access to the variable by recursively inserting ports in behaviors



Tutorial at IESS 2009 R. Doemer 17

Exposing Communication: 3. Synchronize

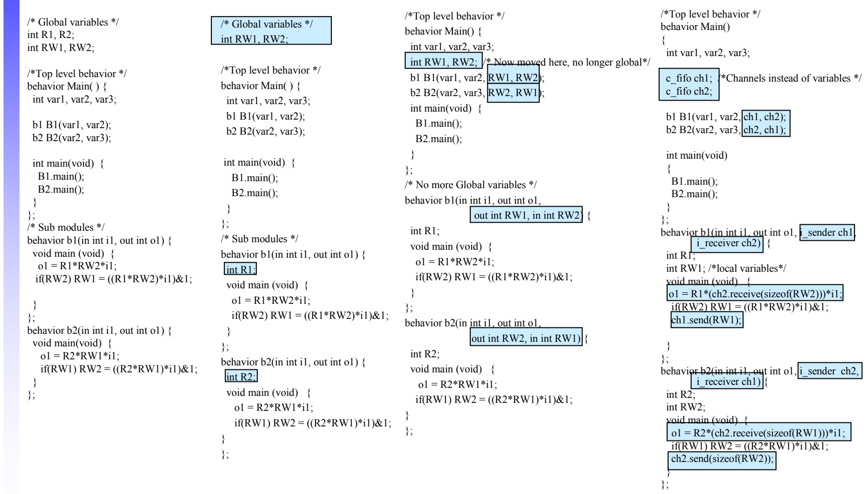
- Use message passing channels instead of variables
  - Defines synchronization scheme
  - Guides exploration tools
- Procedure
  - Create a typed synchronization channel
  - Replace the ports corresponding to the original variable with the channel interface type
  - Modify each access to the variable to call the appropriate interface function of the channel
    - read() / receive()
    - write() / send()



Tutorial at IESS 2009 R. Doemer 18

Exposing Communication: Example Code

- Transformations require significant code modification!



(a) Model-1: Original Model (b) Model-2: After Localization (c) Model-3: Exposed connectivity (d) Model-4: Synchronized Model

Pointer Recoding

- Pointer ambiguities limit the effectiveness of system design tools
  - Architecture exploration tools
    - Analyzability
  - High level synthesis tools
    - Synthesizability
  - Verification and validation tools
    - Verifiability

⇨ Pointers pose a problem for MPSoC Design

- Proposed Solution: Pointer re-coding
  - Enables design tools which otherwise cannot handle pointers
  - Aids program comprehension

⇨ Resolves some of the critical pointers in the specification

Pointer Recoding

- What is pointer re-coding?
  - Replacing indirect pointer accesses with direct variable accesses

```
int x, y;
int *p1;
…
p1 = &x;
*p1 = y+1;
```

➡

```
int x, y;
//p1 removed
…
//Nothing here
x =y+1;
```

Simple Example

- What do we need for pointer re-coding?
  - Basic Idea: Pointer Analysis + Replacement
    - We use existing pointer analysis
    - We contribute automatic pointer replacement

Tutorial at IESS 2009 R. Doemer 21

Pointer Recoding: Pointer Analysis

- 2 types of pointer analyses exist
  - Points-to analysis
    - Determines the memory location a pointer points to
  - Alias analysis
    - Determines if two pointer expressions point to the same location
- Points-to analysis
  - In general, not solvable [4,5,6]
  - Most algorithms trade-off between precision and run-time
    - Flow sensitivity ([1] vs [2])
    - Context sensitivity ([1] vs [3])
- Our Points-to analysis
  - Andersen's algorithm [1]
    - Flow-insensitive and Context-insensitive
  - Operates on an Abstract Syntax Tree representation of the program

```
1.  int a[50], x;
2.  int *p1,*p2, *p3;
3.  …
4.  if(x) p1 = &v1;
5.  else p1 = &v2;

6.  p2 = &x;
7.  *p2 = y+1;

8.  p3 = a;
9.  p3++;
10.*p3++ = 1;
```

Points-to List

p1 → v1, v2
p2 → x
p3 → a[ ]

Tutorial at IESS 2009 R. Doemer 22

Pointer Recoding: Limitations

- Not all pointers can be recoded
- Depends on how a pointer is used
  - Pointer as *value*
    - Absolute value of the pointer is used
    - Eg. *p1*
  - Pointer as *alias*
    - Pointer could point to more than one variable
    - Eg. *p2*
  - Pointer as *address*
    - When pointer is dereferenced
    - Eg. *p3*
  - Pointer as *offset*
    - When the pointer points to an array and is manipulated using pointer arithmetic
    - Eg. *P4, initial offset is 2*
- We recode only pointers that are used as *address/offset*

```
1. int a[50];
2. int *p1,*p2, *p3, *p4;
3.  …
4. if(p1) p2 = &v1;
5. else p2 = &v2;

6. p3 = &x;
7. *p3 = y+1;

8. p4 = &a[2];
9. p4++;
10.*p4++ = 1;
```

Tutorial at IESS 2009 R. Doemer 23

Pointer Recoding: Example

- Re-coding pointers to scalars
  - Indirect access to the scalar is replaced with direct access
- Re-coding pointers to arrays
  - Pointer to an array (*p2*) is replaced with an index variable (*ip2*)
  - Pointer arithmetic is replaced with equivalent arithmetic of the index variable (*ip2+=2)*
  - Pointer access is replaced with array access (*a[ip2])*

```
int x, y;
int *p1;
…
p1 = &x;
…
*p1 = y+1;
```

➡

```
int x, y;
//p1 removed
…
//Nothing here
…
x =y+1;
```

Recoding pointer to scalar

```
int a[50]
int *p2;
…
p2 = a;
p2+=2;
*p2++ = 1;
```

➡

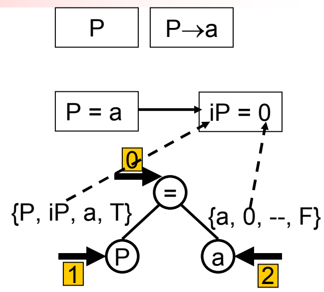
```
int a[50];
int ip2;
…
ip2 = 0;
ip2+=2;
a[ip2++] = 1;
```

Recoding pointer to array

Tutorial at IESS 2009 R. Doemer 24

Pointer Recoding: Algorithm (1)

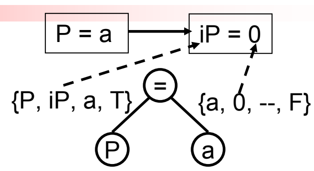
- Input:
  - Pointer to be recoded (*P*)
  - Points-to information (*P→a*)
  - AST of the input program (*P=a*)
- Algorithm
  - Recursively process each node of the AST in Depth First manner
  - Each recursive-call returns 4-tuple
    1. Unmodified original expression (*P*)
    2. Index variable expression (*iP*) or offset expression (*0*)
    3. Target variable (*a*)
    4. Boolean indicating positive pointer match (*True/False*)
  - The results are propagated upwards through the AST
  - Recoding decision is made at the parent node that has the global picture
- Output
  - Recoded AST (*iP=0*)

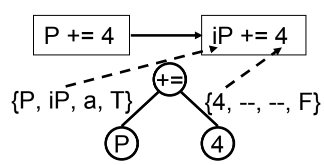


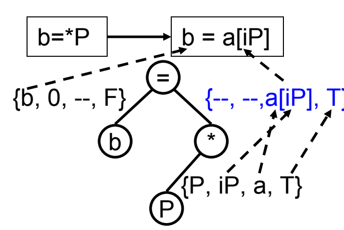
Tutorial at IESS 2009 R. Doemer 25

Pointer Recoding: Algorithm (2)

- Recoding decision depends on the expression type
  - Pointer Initialization
    - Replace with index variable initialization
  - Pointer arithmetic
    - Replace with index variable arithmetic
  - Pointer dereferencing
    - Replace with array access expression or just the target scalar
  - etc. (see [ISSS+CODES'07])



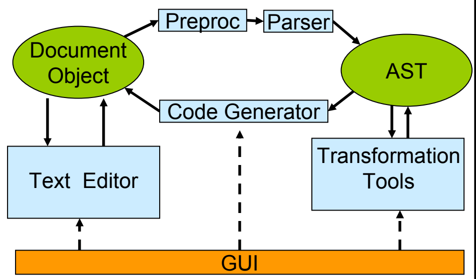




Tutorial at IESS 2009 R. Doemer 26

Interactive Source Recoder

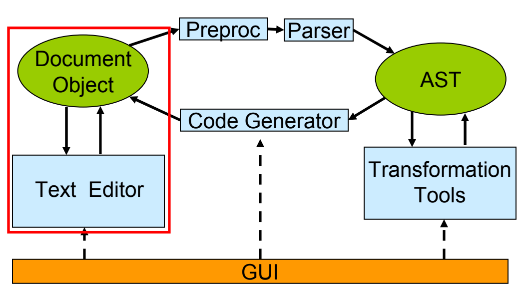
- Implementation
  - Integrated Development Environment (IDE)
- *Cute* tool is a union of
  - Text editor
  - Abstract Syntax Tree (AST)
  - Parser
  - Transformations
  - Code generator



Tutorial at IESS 2009 R. Doemer 27

Interactive Source Recoder

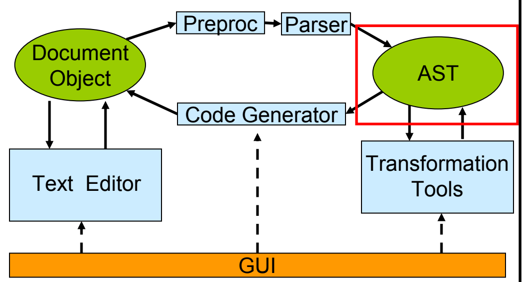
- Text editor
  - Interface to the designer
  - Basic and advanced source-code editing
    - C/C++/SpecC
  - Document object
    - Based on Andrew text editor [8]

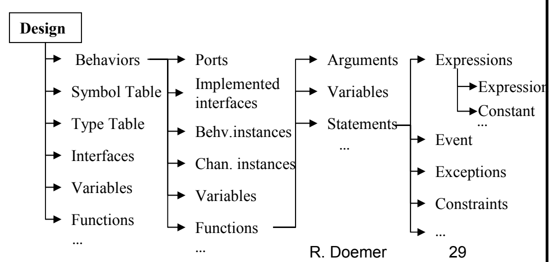


Tutorial at IESS 2009 R. Doemer 28

Interactive Source Recoder

- Text editor
- Abstract Syntax Tree
  - Captures the structure of the design model
  - Used by transformation tools
  - Complete coverage
    - C and SLDLs
    - Correspondence with document object
  - Can re-generate code in its original form

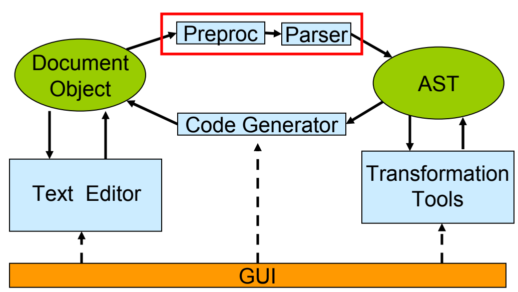




Tutorial at IESS 2009 R. Doemer 29

Interactive Source Recoder

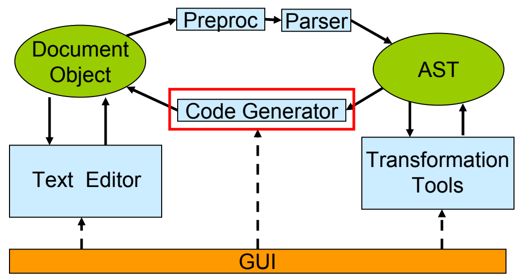
- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
  - Build AST from text
  - Keep AST in synch
  - Complement the editor
    - Color coding
    - Syntax high-lighting



Tutorial at IESS 2009 R. Doemer 30

Interactive Source Recoder

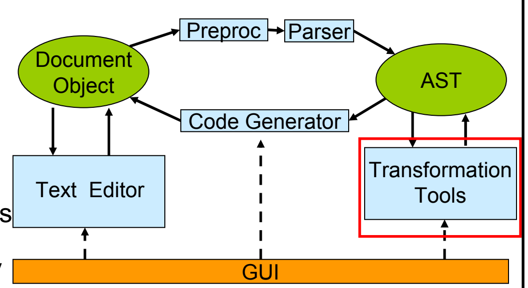
- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
- Code Generator
  - Generates SLDL source code after transformations
  - Keeps text in synch



Tutorial at IESS 2009 R. Doemer 31

Interactive Source Recoder

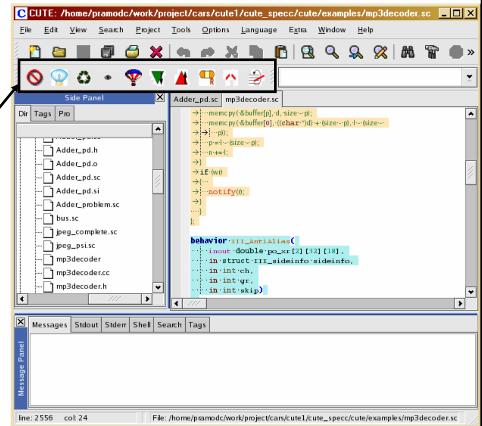
- Text editor
- Abstract Syntax Tree
- Preprocessor and Parser
- Code Generator
- Transformation tools
  - Recoding transformations
    - Code partitioning
    - Create structural hierarchy
  - Data transformations
    - Variable re-scoping
    - Data structure partitioning
  - Analysis
    - Dependency analysis
    - Pointer analysis



Tutorial at IESS 2009 R. Doemer 32

Interactive Source Recoder

- Interactive Environment
  - Scintilla + QT + AST + Transformations
- Basic editing
  - Syntax highlighting
  - Auto-completion
  - …
- Recoding Transformations
  - Dependency analysis
  - Code and data splitting
  - Variable re-scoping
  - Port insertion
  - …



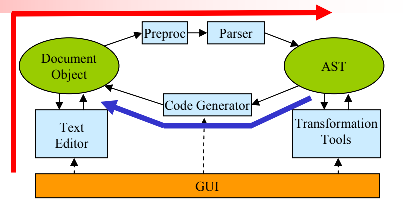
Tutorial at IESS 2009 R. Doemer 33

Experiments and Results

- We have conducted various sets of experiments
- Goals
  - Responsiveness of the “compiler in the editor”
  - Estimated Productivity Gains
    - Extrapolation based on the number of lines of code changed
  - Measured Productivity Gains
    - Class of graduate students
- Design examples
  - GSM Vocoder (voice codec in mobile phones)
  - MP3 Decoder (audio decoder, e.g. iPod)
    - Fixed-point version
    - Floating-point version
  - JPEG Encoder (image encoder, e.g. digital camera)
  - …

Tutorial at IESS 2009 R. Doemer 34

Experiments and Results: Responsiveness



- Why measure Responsiveness ?
  - To check feasibility
- Responsiveness
  - Response to designer actions
  - Time to synch AST
    - On editing
  - Time to synch Editor
    - On transformation
  - Depends on the size of the AST
- Design examples
  - JPEG, MP3, GSM
  - << 1 sec (on a 3 GHz Linux PC)
  - File I/O overhead (20%)

Operation	Simple	JPEG	MP3	GSM
Lines of code	174	1642	7086	7492
Objects in AST	1073	5338	31763	26009
Synch AST	**0.15 secs**	**0.19 secs**	**0.68 secs**	**0.55 secs**
Synch Editor	**0.16 secs**	**0.20 secs**	**0.73 secs**	**0.59 secs**

Tutorial at IESS 2009 R. Doemer 35

Experiments and Results

- Productivity Gain
  - Creating structural hierarchy
    - Manually
      - estimation
    - Automatically
      - measured
- Results
  - Manual time
    - ➢ weeks
  - Recoding time
    - ➢ minutes

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

[ASPDAC'08]

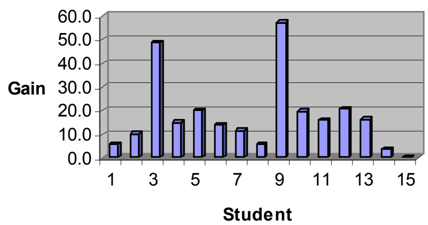
➢ Significant productivity gains!

Tutorial at IESS 2009 R. Doemer 36

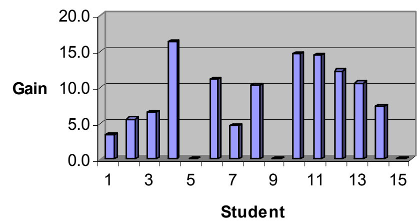
Experiments and Results: Productivity

- Measured Productivity Gains
  - Class of 15 graduate students
  - Recode an MP3 design example
    - Manually (given detailed instructions)
    - Automatically (using the Source Recoder)
- Results

Creating Structural Hierarchy



Pointer Recoding



  - Productivity factors vary, but show significant gains!

Tutorial at IESS 2009 R. Doemer 37

Conclusions

- Embedded System Design
  - Start from higher level of abstraction
  - Need flexible system models in SLDL
- Motivation
  - Automation gap between C reference and SLDL system models
  - 90% of the overall design time spent on “coding” and “re-coding”
  - Need for design automation
- Problem
  - Complete automation is difficult
- Approach
  - *Computer-Aided Recoding* using Source Recoder
  - Designer-in-the-loop
- Results
  - Significant productivity gains
- Future work
  - Research and develop more transformations
  - Improve interactive graphical environment

Tutorial at IESS 2009 R. Doemer 38

References

- [ASPDAC'07] P. Chandraiah, J. Peng, R. Dömer, *"Creating Explicit Communication in SoC Models Using Interactive Re-Coding"*, Proceedings of the Asia and South Pacific Design Automation Conference 2007, Yokohama, Japan, January 2007.
- [IESS'07] P. Chandraiah, R. Dömer, *"An Interactive Model Re-Coder for Efficient SoC Specification"*, Proceedings of the International Embedded Systems Symposium, "Embedded System Design: Topics, Techniques and Trends" (ed. A. Rettberg, M. Zanella, R. Dömer, A. Gerstlauer, F. Rammig), Springer, Irvine, California, May 2007.
- [DAC'07] P. Chandraiah, R. Dömer, *"Designer-Controlled Generation of Parallel and Flexible Heterogeneous MPSoC Specification"*, Proceedings of the Design Automation Conference 2007, San Diego, California, June 2007.
- [ISSS+CODES'07] P. Chandraiah, R. Dömer, *"Pointer Re-coding for Creating Definitive MPSoC Models"*, Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis, Salzburg, Austria, September 2007.
- [ASPDAC'08] P. Chandraiah, R. Dömer, *"Automatic Re-coding of Reference Code into Structured and Analyzable SoC Models"*, Proceedings of the Asia and South Pacific Design Automation Conference 2008, Seoul, Korea, January 2008.
- [TCAD'08] P. Chandraiah, R. Dömer, *"Code and Data Structure Partitioning for Parallel and Flexible MPSoC Specification Using Designer-Controlled Re-Coding"*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems vol. 27, no. 6, pp. 1078-1090, June 2008.
- [DATE'09] R. Leupers, A. Vajda, M. Bekooij, S. Ha, R. Dömer, A. Nohl, *"Programming MPSoC Platforms: Road Works Ahead!"*, Proceedings of Design Automation and Test in Europe, Nice, France, April 2009.