Advanced Modeling Techniques for Efficient Design of Embedded Systems

Rainer Dömer
doemer@uci.edu

Gunar Schirmer, Pramod Chandraiah
Center for Embedded Computer Systems
University of California, Irvine

Outline

• Embedded System Design
• Modeling and Re-Coding
• Recoding Transformations
  – Creating structural hierarchy
  – Code and data partitioning
  – Creating explicit communication
  – Pointer recoding
• Interactive Source Recoder
• Experiments and Results
• Conclusions
Embedded Systems

- System embedded into another system
  - Constraints from external input
  - Application specific
- Omnispreadent in our environment
  - In many application domains
  - In 2005 [Source Netrino]
    - Only 2% of all processors in workstations
    - Remaining 8.8 billion in embedded systems
  - Pervasive

Embedded Systems

- Product challenges
  - Often mobile
  - Performance constrained
  - Tightly coupled software and hardware
    - Specialized
    - Complex
      - E.g. Car: MB E-class
        » 55 ECUs
        » 5 busses
Embedded Systems

- Design Advantages
  - Application known at design time
  - Environment known at design time
  - Allows for customized / optimized solution
    - Improved performance
    - More functionality
    - At lower power
- Customized HW/SW
  - Application Specific Integrated Circuit (ASIC)
    - Multi-Processor System-on-Chip (MPSoC),
      - Complete embedded system integrated on a chip
      - Containing multiple processors
  - Field Programmable Gate Array (FPGA)
  - Circuit board with off-the-shelf-components

How to design a complex embedded system?

System Level Design

- How to efficiently design a complex embedded system?

  International Technology Roadmap for Semiconductors (ITRS) 2004:
  higher-level abstraction and specification is 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
    - SystemC® [Groetker et. al, 2002]
    - SpecC [Gajski et. al, 2000]
  - Use abstract models
    - Describing complete system (hardware + software)
    - Examine implementation implications ahead of time!
System Level Design

- Embedded System Design Flow
  - Input: System model
  - Output: MPSoC platform

- Actual Starting Point
  - C reference code
  - Flat, unstructured, ambiguous
  - Insufficient for system exploration

- Need: System model
  - System-Level Description Language (SLDL)
  - Well-structured
    - Explicit computation, explicit communication
  - Analyzable, synthesizable, verifiable

- Research: Automatic Re-Coding
  - How to get from flat and sequential C code to a flexible and parallel system model?

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
  - Source code transformation
  - C reference code to SpecC specification model
  - MP3 decoder: 12-14 weeks!

- Automation Gap
  - 90% of overall design time is spent on re-coding!

- Proposal: Automatic Recoding
Problem Definition

- How do we go from flat C code to a flexible specification 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
  - Designer-controlled Recoder
    - Interactive source code transformations

Overcoming the Specification Gap

- Recoding Transformations
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]

Create Hierarchy  Partition Code and Data  Expose Communication  Recode Pointers

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

(c) R. Doemer, P. Chandraiah, G. Schirmer
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 leaves

Model 0

Model 1

Model 2

Model 3

Functional Hierarchy

Structural Hierarchy

Code and Data Partitioning

- Today’s parallelizing compilers (e.g. Intel)
  are ineffective for MPSoC platforms
  - Are fully automatic
  - Ineffective for heterogeneous architectures
  - Ineffective on embedded source codes
  - Cannot effectively utilize application-specific knowledge

- Our Approach
  - Designer-controlled programming
  - Designer-in-the-loop
    - Designer utilizes his application knowledge
    - Designer makes decisions
  - Programming by automatic transformations
    - Tools automatically transforms the source code
Code and Data Partitioning

- Desirable model features
  - Enable parallel execution
  - Allow mapping to different PEs
- Recoding tasks
  - Partition code
  - Partition data
  - Synchronize dependents
- Designer-controlled transformations
  1. Loop splitting
  2. Cumulative Access Type analysis
  3. Partitioning of vector dependents
  4. Synchronizing dependent variables

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
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

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
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()

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

(c) R. Doemer, P. Chandraiah, G. Schirmer
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;

Simple Example

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

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

<table>
<thead>
<tr>
<th>Points-to List</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1 → v1, v2</td>
</tr>
<tr>
<td>p2 → x</td>
</tr>
<tr>
<td>p3 → a[ ]</td>
</tr>
</tbody>
</table>

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

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1. int a[50];</td>
</tr>
<tr>
<td>2. int *p1,*p2, *p3, *p4;</td>
</tr>
<tr>
<td>3. ...</td>
</tr>
<tr>
<td>4. if(p1) p2 = &amp;v1;</td>
</tr>
<tr>
<td>5. else p2 = &amp;v2;</td>
</tr>
<tr>
<td>6. p3 = &amp;x;</td>
</tr>
<tr>
<td>7. *p3 = y+1;</td>
</tr>
<tr>
<td>8. p4 = &amp;a[2];</td>
</tr>
<tr>
<td>9. p4++;</td>
</tr>
<tr>
<td>10.*p4++ = 1;</td>
</tr>
</tbody>
</table>
**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 \((a[ip2])\)

**Recoding pointer to scalar**

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

**Recoding pointer to array**

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

**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 \((IP=0)\)
**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 paper]

---

**Interactive Source Recoder**

- Why interactive?
  - Recoding is not a monolithic transformation
    - Multiple iterative transformations
  - Designer-control necessary to create a properly structured model
    - Structure of the model is determined by
      - Type of code transformations
      - Order in which the transformations are applied
  - Recoding involves multiple hard problems
    - Handling pointers
    - Exposing concurrency
    - …
Interactive Source Recorder

- Controlled interactive approach
- Automatic programming
- It's a union of
  - Editor
  - AST
  - Parser
  - Transformations
  - Code generator

Interactive Source Recorder

- Text editor
  - Interface to the designer
  - Basic and advanced source-code editing
    - C/C++/SpecC
  - Document object
    - Based on Andrew text editor [8]
    - Basically text organized for faster access
Interactive Source Recorder

- Editor
- Abstract Syntax Tree
  - Captures the structure of the program
  - For transformation tools
  - Completeness
    - C and SLDLs
    - Correspondence with document object
  - Needed to re-generate code in its original form

Interactive Source Recorder

- Editor
- AST
- Preprocessor and Parser
  - Build AST
  - Keep AST in synch
  - Complement the editor
    - Color coding
    - Syntax high-lighting
Interactive Source Recorder

- Editor
- AST
- Parser
- Code Generator
  - Keeps text in sync
  - Generates the SLDL source-code

Interactive Source Recorder

- Editor
- AST
- Parser
- Code Generator
- Transformation tools
  - Code transformations
    - Code partitioning
    - Introducing behaviors, functions
  - Data transformations
    - Variable re-scoping
    - Data structure partitioning
  - Analysis
    - Dependency analysis
    - Pointer analysis
Source Recoder Implementation

- Interactive environment
  - Scintilla + QT + AST + Transformations
- Basic editing
  - Syntax highlighting
  - Auto-completion
  - ...
- Transformations
  - Splitting code & composite data structures
  - Variable localizing
  - Variable re-scoping
  - Inserting ports
  - Dependency analysis
  - ...

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)
  - ...

(c) R. Doemer, P. Chandraiah, G. Schirmer
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%)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Simple</th>
<th>JPEG</th>
<th>MP3</th>
<th>GSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>174</td>
<td>1642</td>
<td>7086</td>
<td>7492</td>
</tr>
<tr>
<td>Objects in AST</td>
<td>1073</td>
<td>5338</td>
<td>31763</td>
<td>26009</td>
</tr>
<tr>
<td>Synch AST</td>
<td>0.15 sec</td>
<td>0.19 sec</td>
<td>0.68 sec</td>
<td>0.55 sec</td>
</tr>
<tr>
<td>Synch Editor</td>
<td>0.16 sec</td>
<td>0.20 sec</td>
<td>0.73 sec</td>
<td>0.59 sec</td>
</tr>
</tbody>
</table>

Experiments and Results: Productivity

- Creating structural hierarchy
  - manually
  - automatically (Source Recoder)

- Manual time
  - estimated
  - by different designers
  ➢ in the order of weeks

- Recoding time
  - recoding time using Source Recoder
  ➢ in the order of minutes

➢ Significant estimated productivity gains!
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**
  - Productivity factors vary, but show significant gains!

Conclusions

- **Embedded System Design**
  - Start from higher level of abstraction
  - Need flexible system models
  - C reference models act as starting point
- **Motivation:** Gap between reference and system models
  - 90% of the overall design time spent on “coding” and “re-coding”
  - Need for design automation
- **Problem:** Complete automation is difficult
- **Approach:** Interactive Recoding using Source Recoder
  - Designer-in-the-loop
  - Programming becomes automatic recoding
- **Results:** Significant productivity gains shown
- **Future work**
  - Research and develop more transformations
  - Interactive graphical frontend
Interested in learning more?

- EECS Graduate Courses on Embedded System Design

  EECS222A: System-on-Chip Description and Modeling
  EECS222B: System-on-Chip Design and Exploration
  EECS222C: System-on-Chip Software Synthesis
  EECS222D: System-on-Chip Hardware Synthesis

  - Course A is prerequisite for B, C, and D; or: consent of instructor
  - Offered regularly since Fall'07
  - Instructors: Dömer (A, C), Gajski (B, D)