Creating Explicit Communication in SoC Models Using Interactive Re-Coding

Pramod Chandraiah, Junyu Peng and Rainer Doemer
Center for Embedded Computer Systems
University of California, Irvine

Presented By,

Pramod Chandraiah

CECS, UCI

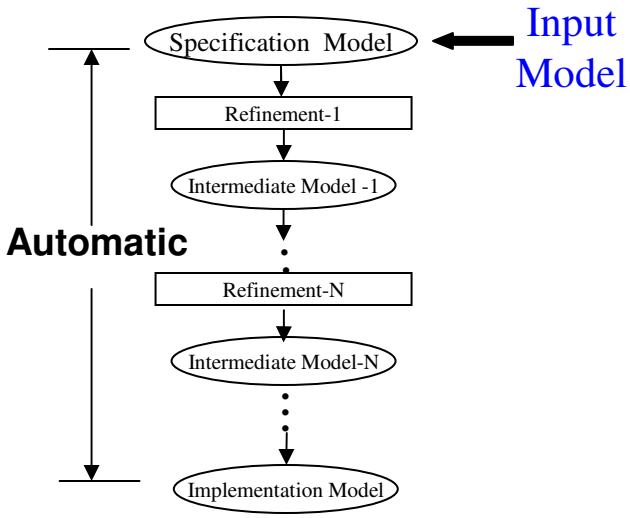
UCIrvine
University of California, Irvine

Outline

- Introduction
- Motivation
- Problem Definition
- Creating Explicit Communication
- Interactive Re-Coding
- Productivity Gains
- Summary and Conclusions

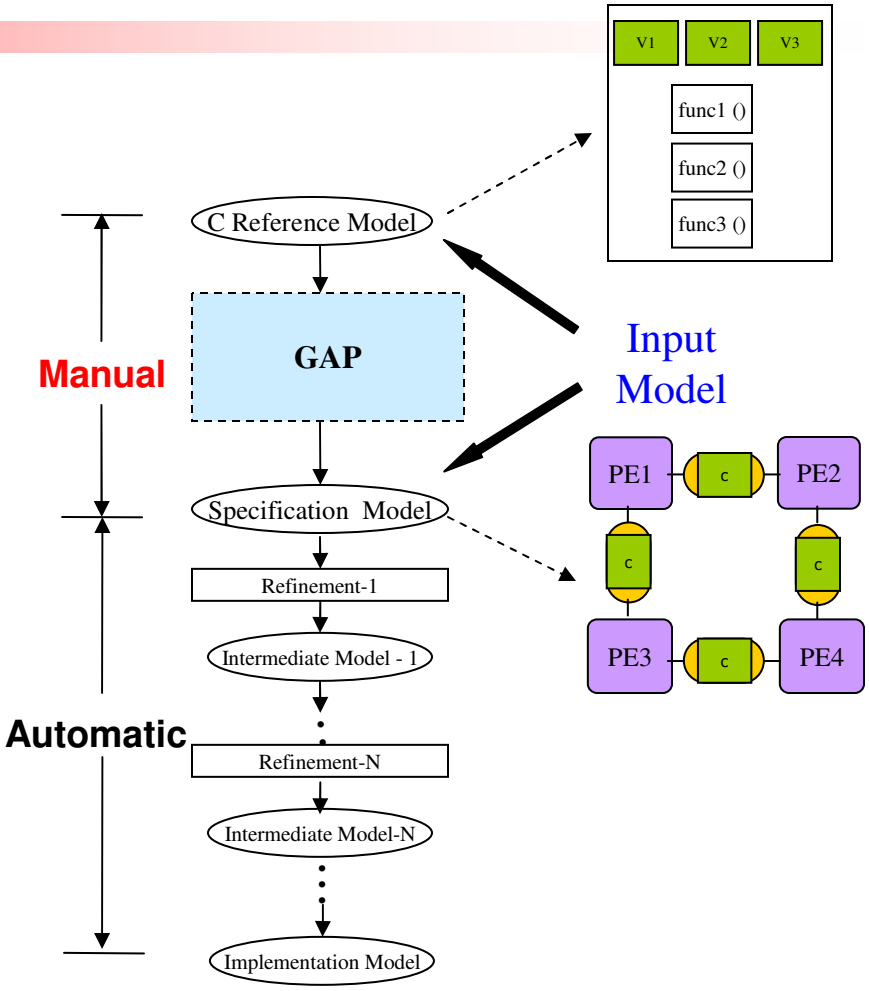
Introduction

- System Level Design
  - Increase productivity by using higher abstraction levels
- System Level Design flow
  - Top-Down
  - Refinement based
  - Starts from Specification Model
    - Using System Level Design Languages (SystemC, SpecC)
  - Successive automatic refinement stages result in end implementation



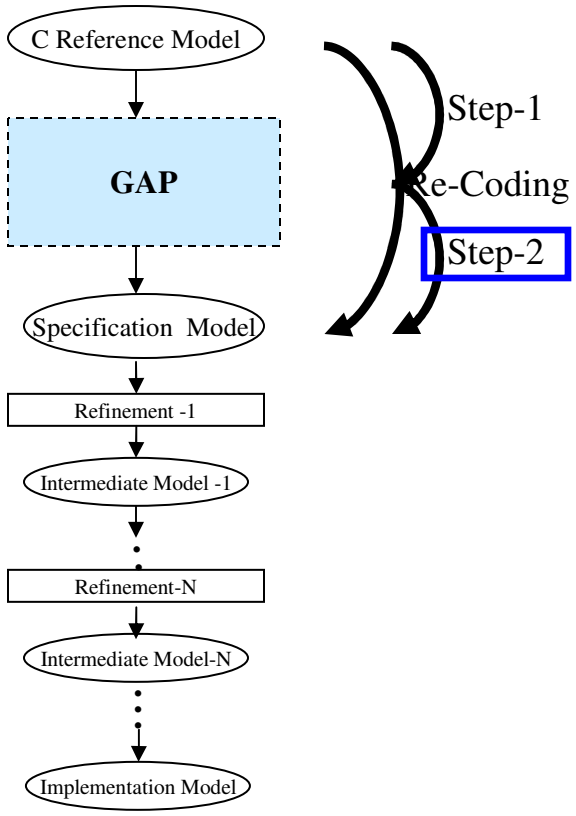
Motivation

- Need quality input model
  - To allow Exploration
- However, actual design starts with reference model (flat C)
  - Gap to reach Specification Model
- Existing design flows do not automatically create of this model
  - [Shin et.al, Lahiri et.al, Pasricha et.al]
- Use manual re-coding
  - Example: MP3 decoder
    - >90% of design time

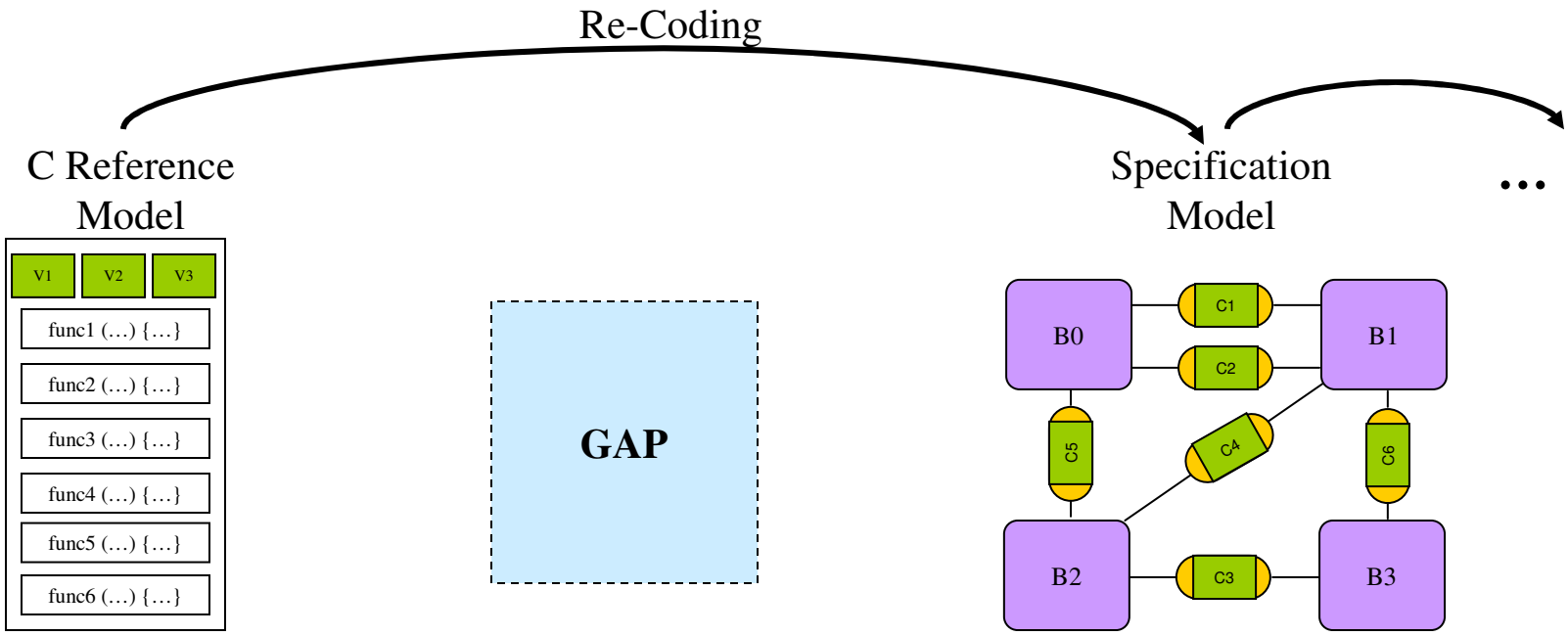


Problem Definition

- Overall goal:
  - Extend design flow to start at reference model
    - Introduce new re-coding stages
- This paper:
  - Introduce Re-coding
    - Code transformations
    - Automation
    - Interactive approach
  - Two steps of Re-coding
    - Focus on communication exploration

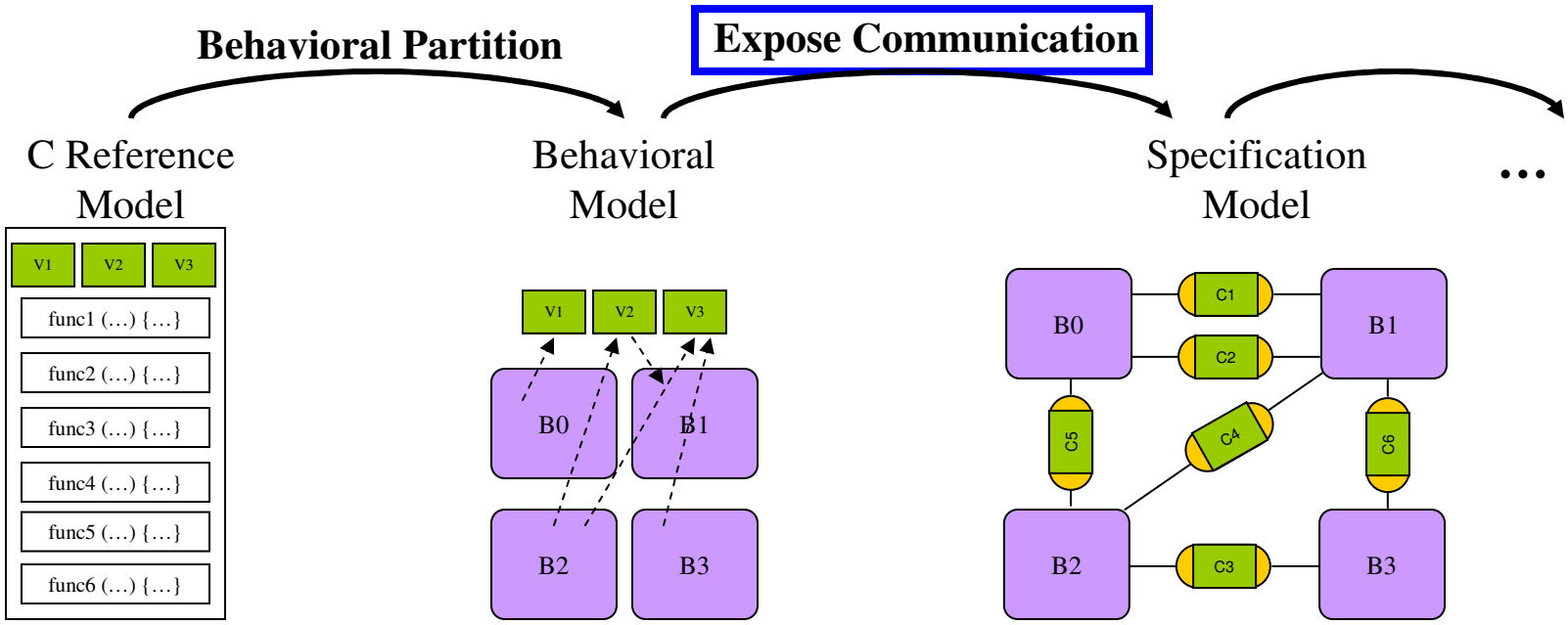


Overcoming the gap



Overcoming the gap…

- Introduce intermediate model
  - Behavioral Model



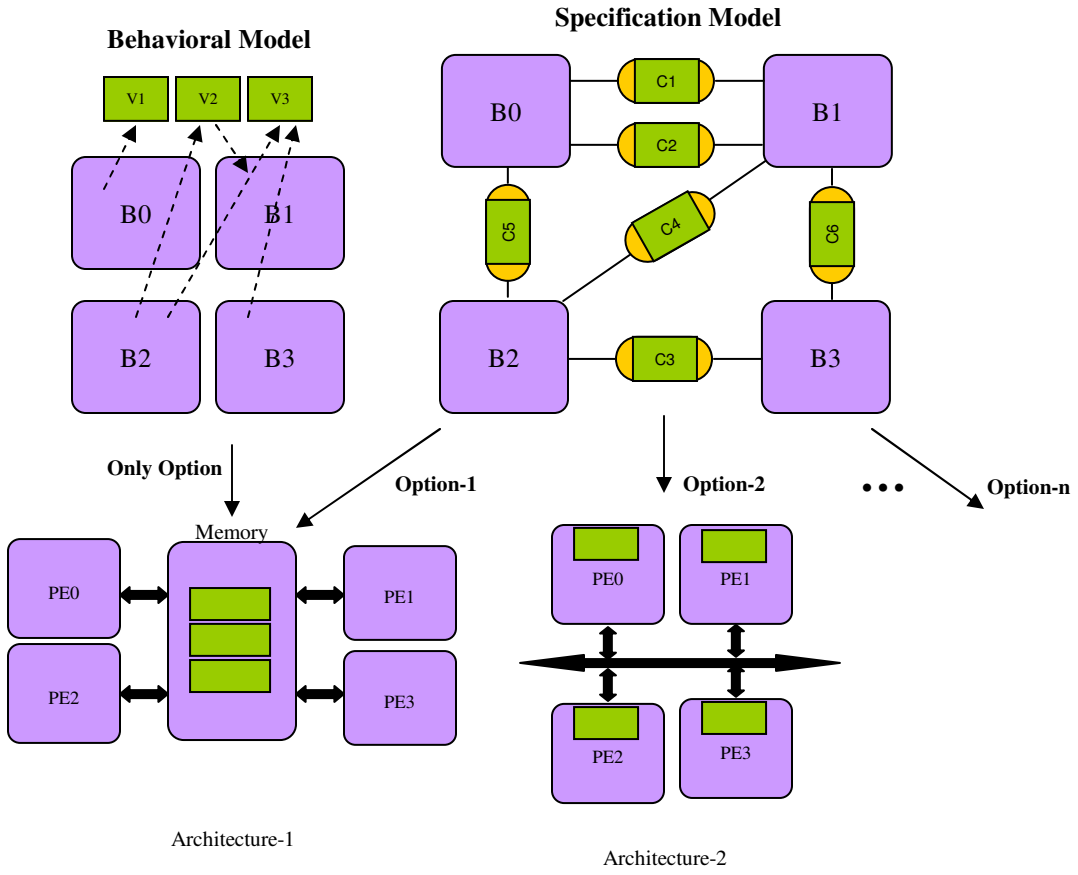
Overcoming the gap…

- Why expose communication?

- Quality of Communication Exploration
  - Number of explorations
  - Extent of automation
  - Time

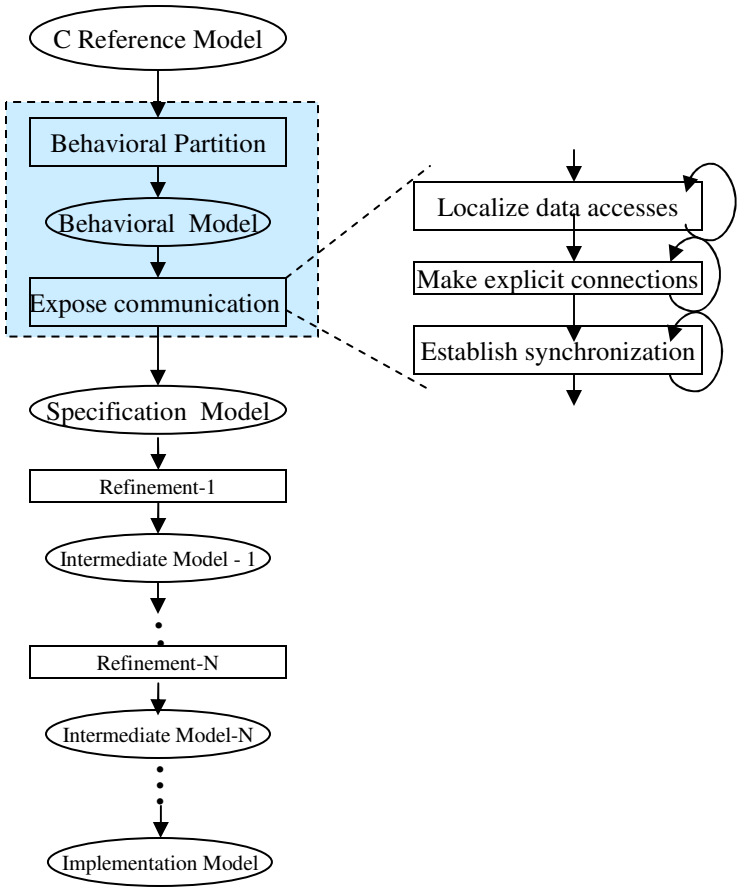
- Behavioral Model
  - Global variables limit the number of possible automatic explorations

- Specification Model
  - Enables automatic exploration of more design alternatives



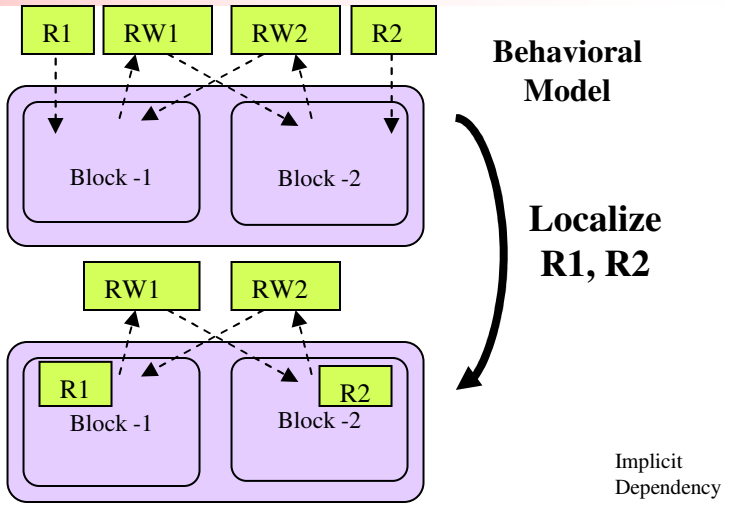
Creating Specification Model

- Two new re-coding steps
  - Behavioral Partitioning
  - Expose Communication
- Expose Communication
  - Input: Behavioral Model
  - Output: Specification Model
  - Transformations
    - Localize global variables
    - Explicit communication
    - Synchronization
  - Interactive approach
    - Designer iteratively invoke transformations instead of manual re-coding



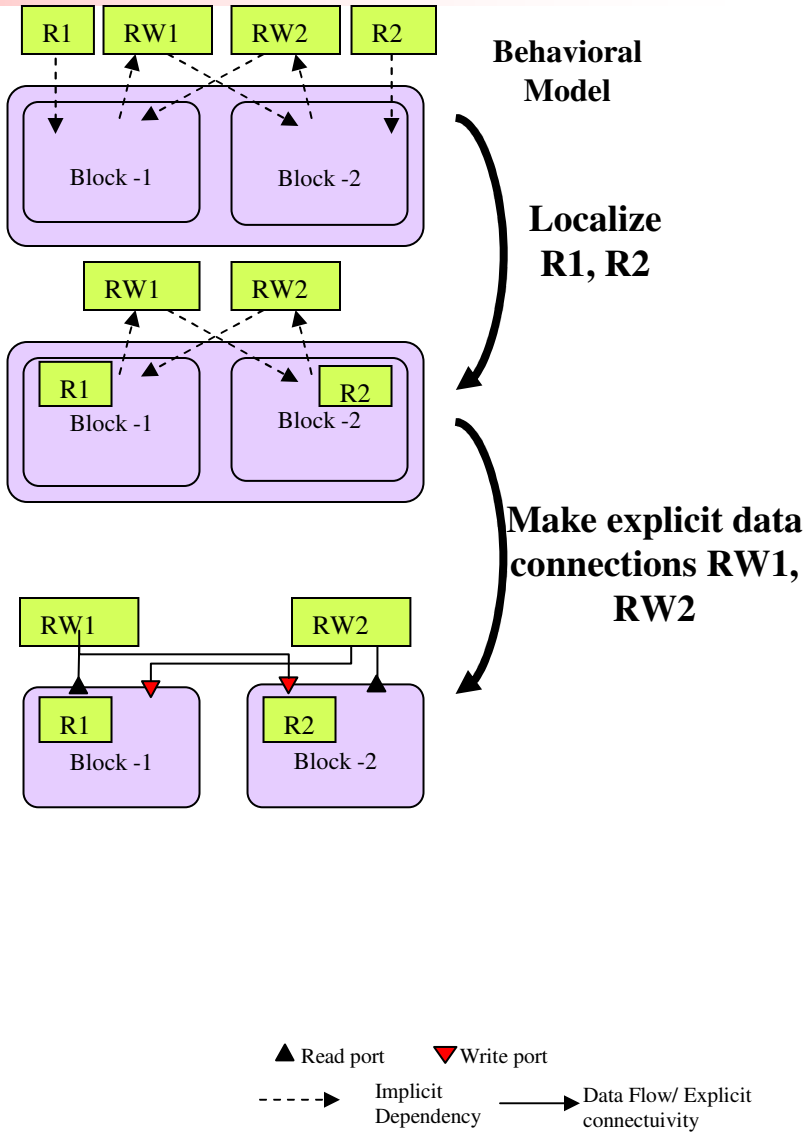
Communication: Localize

- Localize global variables to partitions
- To enable multiple explorations
- Procedure
  - Finding the Global variable
  - Determining the functions and behaviors accessing it
  - Migrate it to the only behavior accessing it



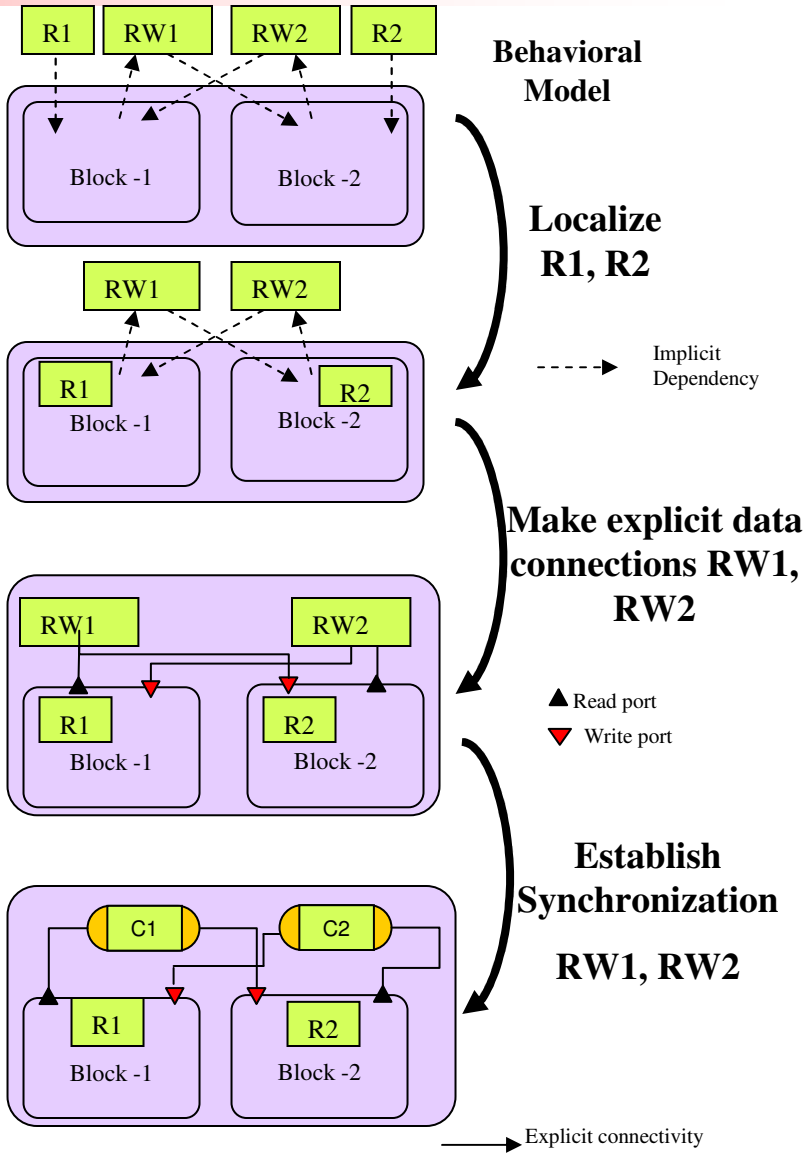
Communication: Expose

- Localize global variables to common parent and provide explicit access
- Makes subsequent analysis of models simpler
- Procedure
  - Finding the Global variable
  - Find the lowest common parent behavior
  - Provide access to the variable by recursively inserting ports in behaviors



Communication: Synchronize

- Use message passing channels instead of variables
- Defines synchronization scheme
  - Guides exploration tools
- Procedure
  - Create a typed synchronization channel
  - Replace all the ports corresponding to the original variable with the channel type
  - Modify each access to the original variable to use appropriate interface function of the channel



Explicit Communication

- Transformations require significant recoding

```
/* Global variables */
int R1, R2;
int RW1, RW2;

/*Top level behavior */
behavior Main( ) {
 int var1, var2, var3;

 b1 B1(var1, var2);
 b2 B2(var2, var3);

 int main(void)  {
   B1.main();
   B2.main();
 }
};
/* Sub modules */
behavior b1(in int i1, out int o1) {
 void main (void)  {
   o1 = R1*RW2*i1;
  if(RW2) RW1 = ((R1*RW2)*i1)&1;

 }
};
behavior b2(in int i1, out int o1) {
 void main(void)  {
    o1 = R2*RW1*i1;
    if(RW1) RW2 = ((R2*RW1)*i1)&1;
 }
};
```

(a) Model-1: Behavioral Model

```
/* Global variables */
int RW1, RW2;

/*Top level behavior */
behavior Main( ) {
 int var1, var2, var3;
 b1 B1(var1, var2);
 b2 B2(var2, var3);

 int main(void)  {
   B1.main();
   B2.main();
 }
};
/* Sub modules */
behavior b1(in int i1, out int o1) {
 int R1;
 void main (void)  {
   o1 = R1*RW2*i1;
   if(RW2) RW1 = ((R1*RW2)*i1)&1;
 }
};
behavior b2(in int i1, out int o1) {
 int R2;
 void main (void)  {
    o1 = R2*RW1*i1;
    if(RW1) RW2 = ((R2*RW1)*i1)&1;
}
};
```

(b) Model-2: After Localization

```
/*Top level behavior */
behavior Main() {
 int var1, var2, var3;
 int RW1, RW2; /* Now moved here, no longer global*/
 b1 B1(var1, var2, RW1, RW2);
 b2 B2(var2, var3, RW2, RW1);
 int main(void)  {
  B1.main();
  B2.main();
 }
};
/* No more Global variables */
behavior b1(in int i1, out int o1,
            out int RW1, in int RW2) {
 int R1;
 void main (void)  {
  o1 = R1*RW2*i1;
  if(RW2) RW1 = ((R1*RW2)*i1)&1;
 }
};
behavior b2(in int i1, out int o1,
            out int RW2, in int RW1){
 int R2;
 void main (void)  {
   o1 = R2*RW1*i1;
  if(RW1) RW2 = ((R2*RW1)*i1)&1;
}
};
```

(c) Model-3: Explicit connectivity

```
/*Top level behavior */
behavior Main()
{
 int var1, var2, var3;

 c_fifo ch1;  /*Channels instead of variables */
 c_fifo ch2;

 b1 B1(var1, var2, ch1, ch2);
 b2 B2(var2, var3, ch2, ch1);

 int main(void)
 {
  B1.main();
  B2.main();
 }
};
behavior b1(in int i1, out int o1, i_sender ch1,
        i_receiver ch2) {
 int R1;
 int RW1; /*local variables*/
 void main (void)  {
  o1 = R1*(ch2.receive(sizeof(RW2)))*i1;
  if(RW2) RW1 = ((R1*RW2)*i1)&1;
  ch1.send(RW1);

 }
};
behavior b2(in int i1, out int o1, i_sender  ch2,
        i_receiver ch1) {
 int R2;
 int RW2;
 void main (void)  {
  o1 = R2*(ch2.receive(sizeof(RW1)))*i1;
  if(RW1) RW2 = ((R2*RW1)*i1)&1;
  ch2.send(sizeof(RW2));
 }
};
```

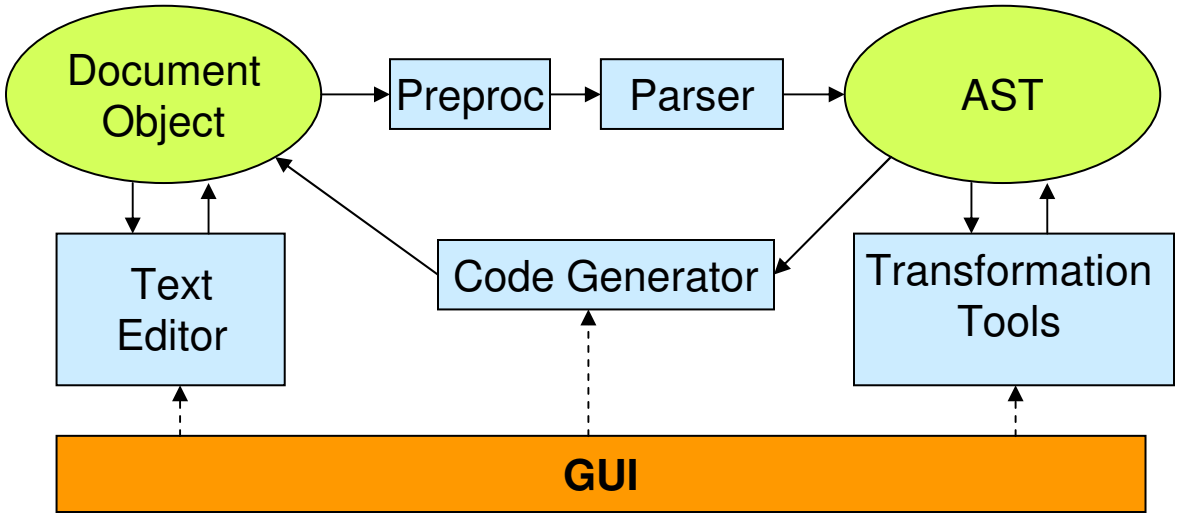
(d) Model-4: Synchronized Model

Interactive Re-coder

- Why not fully Automatic Re-coding?
  - Difficult challenges, for example
    - Determining the type of message passing channel
    - Handling pointer ambiguity
  - Designer control needed
- Existing Interactive Re-coding environments
  - ParaScope, D-Editor, SUIF Explorer

Interactive Re-coder

- Source Re-coder
  - Controlled Interactive approach to replace manual programming
  - Enables generation of models suitable for design flow
  - It's a union of
    - Textual Editor
    - Abstract Syntax Tree
    - Preprocessor/Parser
    - Transformation and analysis functions
    - Code generator



Experimental Results

- Design examples
  - JPEG, MP3, GSM
- Created Specification models
  - Manual
  - Source Re-Coder
- Estimated Manual time
  - Time to implement the transformations manually for 10 variables and extrapolated for all the variables
  - In the order of minutes
- Re-coding time
  - Time to implement the transformations using Source Re-coder
  - In the order of seconds
- Productivity gain
  - **100x**

Statistics	JPEG	MP3	GSM
Lines of Code	1642	7086	7492
Global Variables localized	8	70	83
New Ports added	2	146	163
New Channels added	1	6	2

Design Time	JPEG	MP3	GSM
Estimated Manual time	53 mins	497 mins	585 mins
Re-coding time	27 secs	246 secs	260 secs
Productivity factor	**117**	**121**	**135**

Summary

- We introduced Re-coding to overcome the gap between reference C model and Specification Model
- We presented transformations to expose communication
- We presented an interactive source re-coder
  - Performs transformations to expose communication
  - It creates flexible models in less time
- Our results show significant productivity gains
  - 100x
  - Enables better communication exploration

Conclusions

- SoC Specification Models
  - Good models are necessary for effective exploration and synthesis
  - C Reference Models act as starting point
- Gap between reference model and specification model
  - Specification Modeling needs automation
  - Complete automation is difficult
- Interactive Re-coding
    - Programming specification is no longer tedious typing
    - Becomes guided automatic re-coding
- Future work
  - Transformations to create Behavioral model from C reference model
  - Coupling Source Re-Coder with system profiling and estimation tools

Thank You!

References

1. D. Shin, A. Gerstlauer, R. D¨omer, and D. D. Gajski. Automatic network generation for system-on-chip communication design. In *'CODES+ISSS*,New York, NY, USA, 2005.
2. S. Pasricha, N. Dutt, and M. Ben-Romdhane. Fast exploration of busbased on-chip communication architectures. In *CODES+ISSS*, NewYork, NY, USA, 2004.
3. K. Lahiri, A. Raghunathan, and S. Dey. Efficient exploration of the soc communication architecture design space. In *ICCAD*, Piscataway, NJ,USA, 2000
4. T. Gr¨otker, S. Liao, G. Martin, and S. Swan. System Design with SystemC.Kluwer Academic Publishers, 2002.
5. S.-W. Liao, A. Diwan, R. P. B. Jr., A.M. Ghuloum, andM. S. Lam. SUIF explorer: An interactive and interprocedural parallelizer. In *Principles Practice of Parallel Programming*, pages 37–48, 1999
6. K. Kennedy, K. S. McKinley, and C.-W. Tseng. Analysis and transformation in the ParaScope Editor. In *ACM International Conference onSupercomputing*, Cologne, Germany, 1991.
7. S. Hiranandani, K. Kennedy, C.-W. Tseng, and S. K. Warren. The D editor: anew interactive parallel programming tool. In Supercomputing, 1994