

# On Deriving Equivalent Architecture Model from System Specification

Samar Abdi

Center for Embedded Computer Systems  
UC Irvine  
Irvine, CA 92697  
Tel: 949-824-8059  
Fax: 949-824-8919  
e-mail: sabdi@cecs.uci.edu

Daniel Gajski

Center for Embedded Computer Systems  
UC Irvine  
Irvine, CA 92697  
Tel: 949-824-4155  
Fax: 949-824-4155  
e-mail: gajski@uci.edu

**Abstract**— This paper presents a formal approach to correctly refine a system specification to an architecture model. The tasks in the system specification are distributed onto components of the system architecture to derive the architecture model. We present this refinement step and use formalisms to prove that the derived architecture model is equivalent to the specification. This approach aims at solving the verification problem in system level design through gradual refinements that produce an equivalent output model at each step.

## I. INTRODUCTION

The continuous increase in size and complexity of SoC designs has raised the abstraction level of system specification. We are thus faced with a new verification challenge, namely, how to make sure that abstract specification models written in system design languages like C++, SystemC [2] or SpecC [6] are equivalent to their cycle-accurate implementations.

There are two orthogonal phases in formal verification. The first phase is one where the designer must make sure that the created system specification model indeed implements the desired functionality of the system. Along with the system model, the designer also specifies some properties that he or she expects to hold in the system. Techniques like model checking [4] and theorem proving [1][5] are then employed to check for these properties in the model. The second phase in formal verification is to make sure that once a “golden” model of the system is available, any synthesis or optimization of the model produces a functionally equivalent model. Techniques like logic equivalence checking [3] are used in this phase. Figure 1 shows the role of formal verification in a system design flow.

The rise in abstraction level of system specification is amenable to property verification. The number of possible global states are reduced as a result of abstraction which eases capacity strains on model checkers. However, this comes at the cost of a verification gap as shown in figure 1. Logical equivalence checking cannot be extended to the system level because of a lack of supporting formalisms. Also, checking equivalence of two independently written

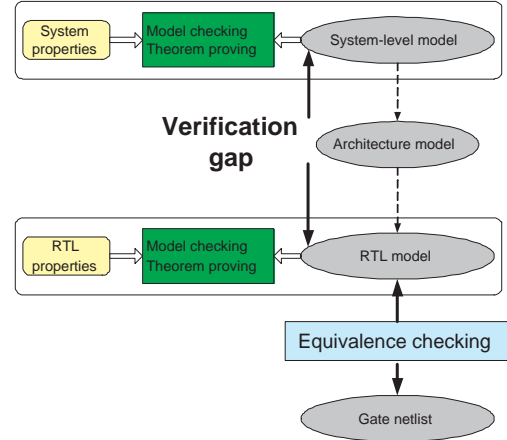


Fig. 1. Formal verification in a top down design flow

high level language programs is not feasible. **The only solution to the problem is deriving models through correct well defined refinement steps** from the specification model. This requires formalisms to capture the specification and proofs for well defined refinement steps. In this paper we present architecture refinement in an algebraic framework to show how we prove equivalence between a system specification model and its derived architecture model. The paper is organized as follows. In section II we present our algebraic framework along with relevant definitions and axioms. In section III, we present the architecture refinement process and the proof of its correctness. Finally, we wind up with conclusions.

## II. MODEL ALGEBRA

System models can be represented by various forms of descriptions including language-theoretic, algebraic and graphical [7]. Each form of description has its own advantage with respect to the problem we are trying to solve. Algebraic representations grouped under process algebras have been researched for several years. In classical books like [9] and [8], the authors present formalisms to define concurrent processes and their interactions. However, we see a need for formalisms closer to system level design languages that can be used for model refinement and proof

of its correctness. To perform correctness proofs for refinements, we introduce an algebraic system called the **model algebra**.

### A. Conceptual Overview

Generally speaking, a system is a set of tasks that are executed in a predefined partial order. These tasks also talk to each other by exchanging data. In order to develop an algebra for system models, we must introduce primitives to represent tasks and the data transactions amongst them.

The first primitive is the unit of computation in a model, referred to as a **behavior**. Behaviors can either be *leaf* or *composite*. A *leaf behavior* represents an **atomic** task in the system description. A *composite behavior* on the other hand is formed by combination of behaviors using operations of the model algebra. The hierarchical composition of behaviors can be seen in figure 2.

The other primitive is the unit of communication called a **channel**. A channel encapsulates the data item to be transferred and events to ensure the transfer semantics. In figure 2 behaviors  $b_2$  and  $b_3$  use channel  $c$  to exchange data variable  $v_2$ . The transfer semantics ensure that the receiving behavior will wait until the sender has written the data, and the sender behavior waits until the data item has been read by the receiver.

### B. Formal Definition

The Model Algebra is defined as:

$$A = \langle \mathcal{B}, \mathcal{C}, \mathcal{O}, \mathcal{R} \rangle$$

$\mathcal{B}$  is the set of behaviors,

$\mathcal{C}$  is the set of channels,

$\mathcal{O} = \{seq, par\}$  (Set of Operations)

$\mathcal{R} = \{\sim, \xrightarrow{v}\}$  (Set of Relations)

#### Operations

The operations mentioned in the above algebra are defined on elements in  $\mathcal{B}$ . The set  $\mathcal{B}$  is closed with respect to both *seq* and *par*. That is  $\forall b_1, b_2, b_3, \dots \in \mathcal{B}$ ,  $seq(b_1, b_2, b_3, \dots) \in \mathcal{B}$  and  $par(b_1, b_2, b_3, \dots) \in \mathcal{B}$

The *seq* operator creates a sequential composition, while a *par* operator creates a parallel composition of behaviors.

Composite behaviors are essentially functions formed using operators *seq* and *par* on behaviors. We will use the notation  $f(b_1, b_2, \dots, b_n)$  to represent a composite behavior formed using behaviors  $b_1$  through  $b_n$ .

#### Relations

We define the *synchronization* relation on  $\mathcal{B}$  in the following way.  $\forall b_1, b_2 \in \mathcal{B}$ , if  $b_1 \sim b_2$  then irrespective of the hierarchical composition, behavior  $b_2$  cannot start executing until behavior  $b_1$  completes.

Data transfers in a system can take place either through variables or channels. Sequentially composed behaviors communicate through variables, while those composed in

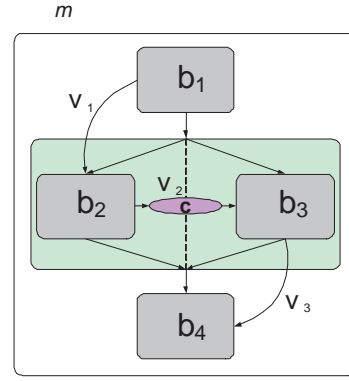


Fig. 2. A simple specification model

parallel use channels. In the latter case, data from the sender behavior is written to the channel. Subsequently, the receiver behavior reads the data from the channel. We define three kinds of *data transfer* relations as follows.

1. behavior to behavior:  $b_1 \xrightarrow{v} b_2, (b_1, b_2) \in \mathcal{B} \times \mathcal{B}$
2. behavior to channel:  $b_1 \xrightarrow{v} c_1, (b_1, c_1) \in \mathcal{B} \times \mathcal{C}$
3. channel to behavior:  $c_1 \xrightarrow{v} b_1, (c_1, b_1) \in \mathcal{C} \times \mathcal{B}$

#### Class of Identity Behaviors

We define the class of *Identity Behaviors*  $\mathcal{I}$  to be a **subset** of  $\mathcal{B}$  such that all behaviors belonging to  $\mathcal{I}$  simply output the input data. The following behaviors belong to the class  $\mathcal{I}$ :

**Behavior**(in var  $v_1$ , out var  $v_2$ )  $\{ v_2 = v_1 \}$

**Behavior**(in var  $v$ , out channel  $c$ )  $\{ c.write(v) \}$

**Behavior**(in channel  $c$ , out var  $v$ )  $\{ v = c.read() \}$

**Behavior**(in channel  $c_1$ , out channel  $c_2$ )  
 $\{ c_2.write(c_1.read()) \}$

### C. Model Definition

Based on the above algebra, a system model  $m$  can be defined as a tuple

$$m = B(m), R(m), \text{ where}$$

$B(m)$  : the hierarchical composition of behaviors representing the model  $m$ , and

$R(m)$  : set of relations on behaviors in  $B(m)$

Figure 2 shows a simple specification model comprising of four *leaf* behaviors  $b_1, b_2, b_3$  and  $b_4$ . Behaviors  $b_2$  and  $b_3$  are composed in parallel to create a *composite* behavior  $b_{23}$ , which in turn is composed sequentially with behaviors  $b_1$  and  $b_4$ . Channel  $c$  is used to send data  $v$  from  $b_2$  to  $b_3$ . Variable  $v_1$  is sent directly from behavior  $b_1$  to  $b_2$  since they are in sequential composition. It must be noted that each behavior is a distinct task instance in the system and should be named uniquely. The model  $m$  can be written as follows:

$$B(m) = seq(b_1, par(b_2, b_3), b_4)$$

$$R(m) = \{ b_1 \xrightarrow{v_1} b_2, b_2 \xrightarrow{v_2} c, c \xrightarrow{v_2} b_3, b_3 \xrightarrow{v_3} b_4 \}$$

## C.1 Terms and definitions

**Sub-behavior** The relation *sub-behavior* ( $\triangleleft$ ) is defined on  $\mathcal{B}$  as follows.  $\forall b_1, b_2 \in \mathcal{B}$ , if  $b_1 \triangleleft b_2$ , then  $b_1$  is a sub-expression in the hierarchical expression of  $b_2$ .

*Example from Figure 2:*  $b_{23} \triangleleft B(m)$ .

**Leafs** This is the set of all leaf level sub-behaviors of a behavior.

$$\text{Leafs}(b) = \{x \mid x \triangleleft b, \nexists y \triangleleft b \text{ s.t. } y \triangleleft x\}$$

$$\text{Example from Figure 2: } \text{Leafs}(B(m)) = \{b_1, b_2, b_3, b_4\}.$$

**Predecessor** A behavior  $b_1$  is said to be a *predecessor* of behavior  $b_2$  in model  $m$  (denoted by  $b_1 \overset{m}{\prec} b_2$ ), if in the temporal order of execution  $b_1$  must complete before  $b_2$  begins.

$$\text{Examples from Figure 2: } b_1 \overset{m}{\prec} b_2, b_1 \overset{m}{\prec} b_4$$

**Immediate Predecessor** A behavior  $b_1$  is said to be an *immediate predecessor* of behavior  $b_2$ , in a model  $m$  (denoted by  $b_1 \overset{m}{\ll} b_2$ ) if  $\nexists b_3 \triangleleft B(m), b_3 \in \mathcal{B}$ , such that  $b_1 \overset{m}{\prec} b_3 \overset{m}{\prec} b_2$

$$\text{Examples from Figure 2: } b_1 \overset{m}{\ll} b_3, b_2 \overset{m}{\ll} b_4.$$

## C.2 Axioms

The execution semantics of the model follow directly from its algebraic description. Leaf behaviors in the model represent system tasks. Sequential and parallel composition of these behaviors enforces a partial order on the tasks. Synchronization relations and communication channels are used to preserve this order while changing task composition. Each of the axioms associated with model algebra shows a basic model transformation that would preserve the execution semantics.

We define the following set of axioms that are associated with the model algebra.

**Axiom 1 (Synchronization)** *A sequential composition of behaviors  $b_1, b_2$  in a model  $M$  may be replaced by a parallel composition of  $b_1, b_2$  by adding a synchronization relation  $b_1 \rightsquigarrow b_2$  in  $R(m)$ .*

$$f(\text{seq}(b_1, b_2), \dots), R(m) = f(\text{par}(b_1, b_2), \dots), R(m) \cup b_1 \rightsquigarrow b_2$$

**Axiom 2 (Flattening)** *If a composite behavior  $x$  in model  $m$  has parent  $b$  of the same composite type as  $x$ , and  $x$  does not have any synchronization relations, then  $x$  may be removed through flattening.*

$$\begin{aligned} \mathbf{2.1} \quad & x = \text{seq}(b_{i+1}, b_{i+2}, \dots, b_j) \\ & b = \text{seq}(b_1, b_2 \dots b_i, x, b_{j+1}, b_{j+2} \dots b_k) \triangleleft B(m) \wedge \\ & \nexists a \triangleleft B(m), \text{ s.t. } a \rightsquigarrow x \in R(m) \vee x \rightsquigarrow a \in R(m), \text{ then} \\ & b = \text{seq}(b_1, b_2, \dots, b_i, b_{i+1}, b_{i+2}, \dots, b_j, b_{j+1}, b_{j+2}, \dots, b_k) \end{aligned}$$

$$\begin{aligned} \mathbf{2.2} \quad & x = \text{par}(b_{i+1}, b_{i+2}, \dots, b_j) \\ & b = \text{par}(b_1, b_2 \dots b_i, x, b_{j+1}, b_{j+2} \dots b_k) \triangleleft B(m) \wedge \\ & \nexists a \triangleleft B(m), \text{ s.t. } a \rightsquigarrow x \in R(m) \vee x \rightsquigarrow a \in R(m), \text{ then} \\ & b = \text{par}(b_1, b_2, \dots, b_i, b_{i+1}, b_{i+2}, \dots, b_j, b_{j+1}, b_{j+2}, \dots, b_k) \end{aligned}$$

**Axiom 3 (Forward Substitution) Synchronization** *relation for composite behaviors may be replaced by synchronization relation(s) on their child behaviors.*

$$\begin{aligned} \mathbf{3.1} \quad & b = \text{seq}(b_1, b_2, \dots, b_n) \triangleleft B(m), \text{ then } \forall a \triangleleft B(m) \\ & \text{if } a \rightsquigarrow b \in R(m), R(m) = (R(m) - a \rightsquigarrow x) \cup a \rightsquigarrow b_1 \\ & \text{if } b \rightsquigarrow a \in R(m), R(m) = (R(m) - b \rightsquigarrow a) \cup b_n \rightsquigarrow a \end{aligned}$$

$$\begin{aligned} \mathbf{3.2} \quad & b = \text{par}(b_1, b_2, \dots, b_n) \triangleleft B(m), \text{ then } \forall a \triangleleft B(m) \\ & \text{if } a \rightsquigarrow b \in R(m), R(m) = (R(m) - a \rightsquigarrow b) \cup \\ & \{a \rightsquigarrow b_1, a \rightsquigarrow b_2, \dots, a \rightsquigarrow b_n\} \\ & \text{if } b \rightsquigarrow a \in R(m), R(m) = (R(m) - b \rightsquigarrow a) \cup \\ & \{b_1 \rightsquigarrow a, b_2 \rightsquigarrow a, \dots, b_n \rightsquigarrow a.\} \end{aligned}$$

**Axiom 4 (Identity)** *Given a model  $m$  and behavior  $e \in \mathcal{I}$  such that  $e$  does not have any relations in  $R(m)$ , then  $\forall b \triangleleft B(m), \text{seq}(b, e) = \text{seq}(e, b) = \text{par}(b, e) = b$*

**Axiom 5 (Transitivity)** *Given a model  $m$  and  $b_1, b_2, b_3 \triangleleft B(m)$*

$$b_1 \overset{v}{\rightarrow} b_2 = \{b_1 \overset{v}{\rightarrow} b_3, b_3 \overset{v}{\rightarrow} b_2\} \text{ iff } b_1 \overset{m}{\prec} b_3 \wedge b_3 \overset{m}{\prec} b_2$$

**Axiom 6 (Channel creation)** *Given identity behaviors  $e_1, e_2 \in \mathcal{I}$  and channel  $c \in \mathcal{C}$*

$$\mathbf{6.1} \quad \{e_1 \overset{v}{\rightarrow} e_2, e_1 \rightsquigarrow e_2\} = \{e_1 \overset{v}{\rightarrow} c, c \overset{v}{\rightarrow} e_2\}$$

$$\mathbf{6.2} \quad \{e_1 \rightsquigarrow e_2\} = \{e_1 \overset{0}{\rightarrow} c, c \overset{0}{\rightarrow} e_2\}$$

## III. ARCHITECTURE REFINEMENT

Once we have determined the components in the proposed system architecture, we need to divide the system tasks into suitable groups. Each of these groups is mapped to a unique component in the architecture. From these decisions we need generate an executable architecture model for evaluation. This process is known as architecture refinement [10]. Essentially, we need to derive a model with top level behaviors composed in parallel and representing the components in system architecture. In this section, we present the architecture refinement process using our model algebra and prove that the produced architecture model is equivalent to the specification. Note that the refinement step in no way influences the decision on assignment of tasks to components. The refinement process **allows any possible distribution of tasks on components** in the system architecture.

### A. The refinement process

Given a model  $m = B(m), R(m)$ , a set of  $n$  components  $PE_1, PE_2, \dots, PE_n$  and  $n$  groups  $g_1, \dots, g_n$ . Each group is a set of leaf behaviors in  $m$ . The task distribution follows the following rules:

1.  $\bigcup_{i=1}^n g_i = \text{Leafs}(B(m))$ , and
2.  $g_i \cap g_j = \phi, 1 \leq i, j \leq n$
3.  $g_i$  is assigned to  $PE_i$

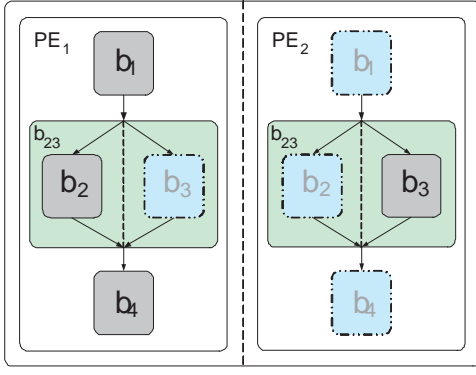


Fig. 3. Intermediate model after step 1 of model refinement.

Let  $Leafs(B(m)) = \{b_1, b_2, \dots, b_m\}$ , then we can write  $B(m) = f(b_1, b_2, \dots, b_m)$ . The architecture model  $m_a$  is generated as follows.

1. Initialize  $B(m_a)$  as a parallel composition of  $n$  behaviors  $PE_1$  through  $PE_n$  such that  $PE_i = f(b_{i1}, b_{i2}, \dots, b_{im})$ , where

$$b_{ij} = \begin{cases} b_j & : b_j \in g_i \\ \phi & : otherwise \end{cases}$$

2. Add following synchronization relations

$$\bigcup_{i=1}^n \{x \rightsquigarrow y | y \in g_i \wedge x \stackrel{m}{\ll} y \wedge x \in (Leafs(B(m)) - g_i)\}$$

3. Introduce identity behaviors for each synchronization relation as follows

$\forall b_1, b_2 \triangleleft B(m)$ , such that  $b_1 \rightsquigarrow b_2 \in R(m)$ ,  $e_1, e_2 \in \mathcal{I}$  replace behavior  $b_1$  with  $seq(b_1, e_1)$  and  $b_2$  with  $seq(e_2, b_2)$  and modify  $R(m)$  as follows

- (a) if  $\exists b_1 \xrightarrow{v} b_2 \in R(m)$ , then  $R(m) = (R(m) - \{b_1 \xrightarrow{v} b_2, b_1 \rightsquigarrow b_2\}) \cup \{b_1 \xrightarrow{v} e_1, e_1 \xrightarrow{v} c_1, c_1 \xrightarrow{v} e_2, e_2 \xrightarrow{v} b_2\}$
- (b) else  $R(m) = (R(m) - \{b_1 \rightsquigarrow b_2\}) \cup \{e_1 \xrightarrow{0} c_1, c_1 \xrightarrow{0} e_2\}$

An example of the refinement process is demonstrated on the simple specification model shown earlier comprising of four leaf behaviors viz.  $b_1, b_2, b_3$  and  $b_4$  as shown in figure 2. Recall that the model  $m$  may be expressed in our algebra as follows:

$$B(m) = seq(b_1, par(b_2, b_3), b_4),$$

$$R(m) = \{b_1 \xrightarrow{v_1} b_2, b_2 \xrightarrow{v_2} c, c \xrightarrow{v_3} b_3, b_3 \xrightarrow{v_3} b_4\}$$

We consider a possible assignment as follows.

$$g_1 = b_1, b_2, b_4 \text{ to } PE_1$$

$$g_2 = b_3 \text{ to } PE_2$$

The model is refined to a parallel composition of  $PE_1$  and  $PE_2$ . We follow steps 1 through 4 of the refinement process to derive an intermediate model  $m_i$ . Synchronization relations are added across groups to maintain the original partial order of execution.  $b_1 \stackrel{m}{\ll} b_3$  and

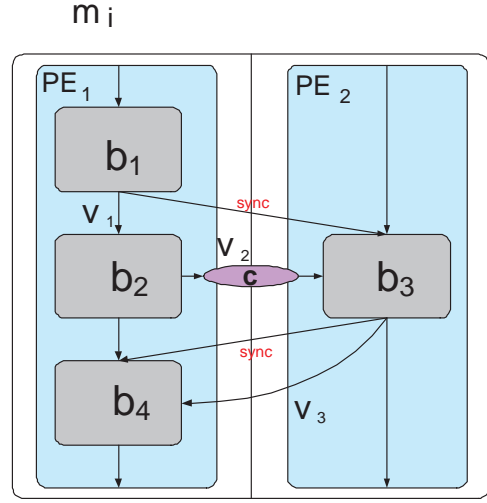


Fig. 4. Intermediate model after step 2 of model refinement.

$b_3 \stackrel{m}{\ll} b_4$ , are the only immediate predecessor relations across groups. Therefore, we add corresponding synchronization constraints i.e.  $b_1 \rightsquigarrow b_3$  and  $b_3 \rightsquigarrow b_4$  to derive intermediate model  $m_i$  as shown in figure 4.

$$B(m_i) = par(seq(b_1, b_2, b_4), seq(b_3)),$$

$$R(m_i) = \{b_1 \xrightarrow{v_1} b_2, b_2 \xrightarrow{v_2} c, c \xrightarrow{v_3} b_3, b_3 \xrightarrow{v_3} b_4, b_1 \rightsquigarrow b_3, b_3 \rightsquigarrow b_4\}$$

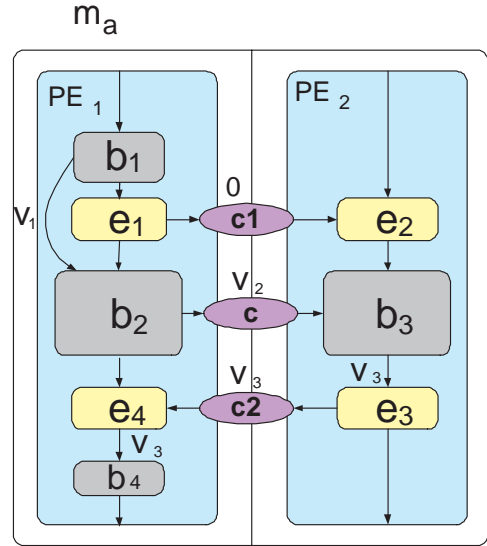


Fig. 5. Final model after partitioning.

The final model  $m_a$  as shown in Figure 5. This model is derived by executing step 3 of the refinement process on the intermediate model  $m_i$ . Identity behaviors  $e_1, e_2, e_3$  and  $e_4$  and channels  $c_1$  and  $c_2$  are inserted corresponding to synchronization relations  $b_1 \rightsquigarrow b_3$  and  $b_3 \rightsquigarrow b_4$ . The pair of relations  $\{b_3 \rightsquigarrow b_4, b_3 \xrightarrow{v_3} b_4\}$  in  $m_i$  is replaced by  $\{b_3 \xrightarrow{v_3} e_3, e_3 \xrightarrow{v_3} c_2, c_2 \xrightarrow{v_3} e_4, e_4 \xrightarrow{v_3} b_4\}$  in  $m_a$ . Similarly,

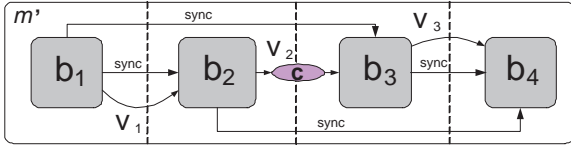


Fig. 6. Canonical model  $m'$

channel  $c_1$  is introduced to implement the synchronization relation between  $b_1$  and  $b_3$ . The final architecture model  $m_a$  can be expressed as:

$$\begin{aligned}
B(m_a) &= \text{par}(\text{seq}(b_1, e_1, b_2, e_4, b_4), \\
&\quad \text{seq}(e_2, b_3, e_3)), \\
R(m_a) &= \{b_1 \xrightarrow{v_1} b_2, b_2 \xrightarrow{v_2} c, c \xrightarrow{v_3} b_3, b_3 \xrightarrow{v_3} b_4, \\
&\quad e_1 \xrightarrow{0} c_1, c_1 \xrightarrow{0} e_2, b_3 \xrightarrow{v_3} e_3, e_3 \xrightarrow{v_3} c_2, \\
&\quad c_2 \xrightarrow{v_3} e_4, e_4 \xrightarrow{v_3} b_4\}
\end{aligned}$$

### B. Theorems

The axioms in Section II establish the basic properties of our algebra. We now present some useful theorems which can be employed to prove the correctness of architecture refinement. Proofs are omitted for space.

**Theorem 1 (Expression Exchange)** *A sequential composite behavior may be converted to parallel*  
 $f(\text{seq}(b_1, b_2, \dots, b_i), b_{i+1}, b_{i+2}, \dots), R(m) =$   
 $f(\text{par}(b_1, b_2, \dots, b_i), b_{i+1}, b_{i+2}, \dots),$   
 $R(m) \cup \{b_1 \rightsquigarrow b_2, b_2 \rightsquigarrow b_3, \dots, b_{i-1} \rightsquigarrow b_i\}$

### Theorem 2 (Permutation of parallel behaviors)

*A parallel composition of the type  $\text{par}(b_1, b_2, \dots, b_i)$  in a given model  $m$ , may be replaced by a parallel composition of behaviors  $b_1$  through  $b_i$  in any order.*

**Theorem 3 (Canonical form)** *Any system model  $m$  is equivalent to a canonical model  $m'$ , which is a parallel composition of all leaf behaviors in  $m$ , and each leaf behavior in  $m'$  has a synchronization relation from all its immediate predecessors in  $m$ .*

$$\begin{aligned}
B(m') &= \text{par}(\text{Leafs}(B(m))), \\
R(m') &= (R(m) - \{x \rightsquigarrow y | x \rightsquigarrow y \in R(m)\}) \cup \\
&\quad \{x \rightsquigarrow y | x \lll^m y \wedge x, y \in \text{Leafs}(B(m))\}
\end{aligned}$$

The canonical model  $m'$  for our simple example is shown in figure 6.

### C. Formal Verification of System Partitioning

We establish and prove the following theorem to prove the correctness of architecture refinement.

**Theorem 4 (Architecture refinement)** *Model  $m_a$  generated by architecture refinement of specification model  $m$ , is equivalent to  $m$ .*

*Proof:*

The proof for this theorem is divided into two parts. First we prove that the intermediate model  $m_i$ , produced after step 2 of the refinement process, is equivalent to  $m$ . In the second part of the proof we show that the final architecture model,  $m_a$ , is equivalent to the intermediate model  $m_i$ .

### Part I

$$\begin{aligned}
\text{Let } m' &= \text{par}(\text{Leafs}(B(m))), \\
&\quad (R(m) - \{x \rightsquigarrow y | x \rightsquigarrow y \in R(m)\}) \cup \\
&\quad \{x \rightsquigarrow y | x \lll^m y \wedge x, y \in \text{Leafs}(B(m))\}
\end{aligned}$$

We have  $m' = m$ , using theorem 3

Similarly, for model  $m_i$ , we have

$$\begin{aligned}
m'_i &= \text{par}(\text{Leafs}(B(m_i))), \\
&\quad (R(m_i) - \{x \rightsquigarrow y | x \rightsquigarrow y \in R(m_i)\}) \cup \\
&\quad \{x \rightsquigarrow y | x \lll^{m_i} y \wedge x, y \in \text{Leafs}(B(m_i))\}
\end{aligned}$$

$m'_i = m_i$ , using theorem 3

$$B(m_i) = \text{par}(PE_1, PE_2, \dots, PE_n)$$

$$\begin{aligned}
\text{Leafs}(B(m_i)) &= \bigcup_{i=1}^n \text{Leafs}(PE_i) \\
&= \bigcup_{i=1}^n \text{partition}_i \\
&= \text{Leafs}(B(m)) \text{ by grouping rules.} \\
B(m'_i) &= \text{par}(\text{Leafs}(B(m_i))), \\
&= \text{par}(\text{Leafs}(B(m))) \\
&= B(m') \text{ using theorem 3}
\end{aligned}$$

Now, we try to prove that the immediate predecessor relations are unchanged as we transform  $m$  to  $m_i$ .

$$\forall b_1 \in \text{partition}_i, b_2 \in \text{partition}_j,$$

$$\text{if } i = j, \text{ then } b_1 \lll^{m_i} b_2 \Leftrightarrow x \lll^m y,$$

$$\text{since each PE is a copy of } B(m)$$

$$\text{if } i \neq j, \text{ then}$$

$$b_1 \lll^{m_i} b_2 \Leftrightarrow b_1 \rightsquigarrow b_2 \in R(m_i),$$

$$\text{since } m \text{ has no synchronization relations}$$

$$\text{Also } b_1 \rightsquigarrow b_2 \in R(m_i) \Leftrightarrow b_1 \lll^{m_i} b_2$$

$$\text{since } b_1 \triangleleft PE_i \wedge b_2 \triangleleft PE_j \text{ and}$$

$$PE_i \text{ and } PE_j \text{ are composed in parallel}$$

$$\text{Therefore } b_1 \lll^m b_2 \Leftrightarrow b_1 \lll^{m_i} b_2$$

For the relations in  $m'_i$ , we have

$$\begin{aligned}
R(m'_i) &= (R(m_i) - \{x \rightsquigarrow y | x \rightsquigarrow y \in R(m_i)\}) \cup \\
&\quad \cup \{x \rightsquigarrow y | x \lll^{m_i} y \wedge x, y \in \text{Leafs}(B(m_i))\} \\
&= (R(m) - \{x \rightsquigarrow y | x \rightsquigarrow y \in R(m)\}) \cup \\
&\quad \cup \{x \rightsquigarrow y | x \lll^{m_i} y \wedge x, y \in \text{Leafs}(B(m_i))\}, \\
&\quad \text{since data-transfer relations are preserved in } m_i \\
&= (R(m) - \{x \rightsquigarrow y | x \rightsquigarrow y \in R(m)\}) \cup \\
&\quad \cup \{x \rightsquigarrow y | x \lll^m y \wedge x, y \in \text{Leafs}(B(m))\}, \\
&\quad \text{since immediate predecessors are unchanged} \\
&= R(m')
\end{aligned}$$

Therefore  $m'_i = m' \implies \mathbf{m}_i = \mathbf{m}$

## Part2

We now prove that executing refinement step 3 on the intermediate model  $m_i$  produces an equivalent model  $m_a$ .

Let  $b_1, b_2 \triangleleft B(m_i)$  and  $b_1 \xrightarrow{v} b_2, b_1 \rightsquigarrow b_2 \in R(m)$ . By definition of predecessor  $b_1 \overset{m_i}{<} b_2$ . Using axiom 4, we have

$$\begin{aligned} b_1 &= seq(b_1, e_1), e_1 \in \mathcal{I} \text{ and} \\ b_2 &= seq(e_2, b_2), e_2 \in \mathcal{I} \end{aligned}$$

hence  $B(m_i) = B(m_a)$ .

We now prove equivalence of relations in the two models,

$$\begin{aligned} b_1 \rightsquigarrow b_2 &= seq(b_1, e_1) \rightsquigarrow b_2, \text{ using axiom 4} \\ &= e_1 \rightsquigarrow b_2, \text{ using axiom 3} \\ &= e_1 \rightsquigarrow seq(e_2, b_2), \text{ using axiom 4} \\ &= e_1 \rightsquigarrow e_2, \text{ using axiom 3} \\ b_1 \xrightarrow{v} b_2 &= b_1 \xrightarrow{v} e_1, e_1 \xrightarrow{v} b_2 \text{ using axiom 5} \\ &= b_1 \xrightarrow{v} e_1, e_1 \xrightarrow{v} e_2, e_2 \xrightarrow{v} b_2 \text{ using axiom 5} \end{aligned}$$

Using the above results, we have

$$\begin{aligned} R(m_i) &= (R(m_i) - \{b_1 \xrightarrow{v} b_2, b_1 \rightsquigarrow b_2\}) \cup \\ &\quad \{b_1 \xrightarrow{v} e_1, e_1 \xrightarrow{v} e_2, e_2 \xrightarrow{v} b_2, e_1 \rightsquigarrow e_2\} \\ &= (R(m_i) - \{b_1 \xrightarrow{v} b_2, b_1 \rightsquigarrow b_2\}) \cup \\ &\quad \{b_1 \xrightarrow{v} e_1, e_1 \xrightarrow{v} c, c \xrightarrow{v} e_2, e_2 \xrightarrow{v} b_2\}, \\ &\quad c \in \mathcal{C} \text{ using axiom 6} \\ &= R(m_a) \end{aligned}$$

If there is no data transfer relation between  $b_1$  and  $b_2$ , but  $b_1 \rightsquigarrow b_2 \in R(m_i)$ , we have

$$\begin{aligned} R(m_i) &= (R(m_i) - \{b_1 \rightsquigarrow b_2\}) \cup \{b_1 \rightsquigarrow b_2\} \\ &= (R(m_i) - \{b_1 \rightsquigarrow b_2\}) \cup \{e_1 \rightsquigarrow e_2\} \\ &= (R(m_i) - \{b_1 \rightsquigarrow b_2\}) \cup \{e_1 \xrightarrow{0} c, c \xrightarrow{0} e_2\}, \\ &\quad c \in \mathcal{C} \text{ using axiom 6} \\ &= R(m_a) \end{aligned}$$

Hence under all cases

$$B(m_i) = B(m_a) \wedge R(m_i) = R(m_a) \implies m_i = m_a$$

Using results from Part 1 and Part 2 of the proof, we get  $\mathbf{m} = \mathbf{m}_a$

## IV. CONCLUSION AND FUTURE WORK

In this paper, we presented a formal approach to correctly derive system level models and applied it to the problem of architecture refinement. This is a fresh perspective to verification in system level design by deriving one model from another, rather than the traditional way of comparing two independently written models. We

showed how models at different abstraction levels may be expressed in the proposed model algebra and how their transformations can be proved to be correct. The strategy of accurate refinement can be extended to prove other system level refinements as well.

The architecture refinement process produced a new model that was derived from the specification model through a series of well defined refinement steps. A theorem was established and proved to show that this refinement produced a model that is equivalent to the specification model. This approach to system level design validation shows a lot of promise. In the future, we will try to expand the algebra to incorporate more modeling capabilities like FSM and pipeline composition of behaviors. This will enable us to develop theorems that will be used to verify more general and complex refinement algorithms.

## REFERENCES

- [1] PVS[online]. Available: <http://pvs.csl.sri.com/>.
- [2] SystemC, OSCI[online]. Available: <http://www.systemc.org/>.
- [3] R. E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computer*, pages 677–691, August 1986.
- [4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.
- [5] D. Cyrluk, S. Rajan, N. Shankar, and M. Srivas. Effective theorem proving for hardware verification. In *Theorem Provers in Circuit Design*, pages 203–222, September 1994.
- [6] D. Gajski, J. Zhu, R. Domer, A. Gerstlauer, and S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, January 2000.
- [7] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [8] C. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
- [9] R. Milner. *A Calculus of Communicating Systems*. Springer, 1980.
- [10] J. Peng, S. Abdi, and D. Gajski. Automatic model refinement for fast architecture exploration. In *Proceedings of the Asia-Pacific Design Automation Conference*, pages 332–337, January 2002.