Hardware/Software Partitioning in Verilog

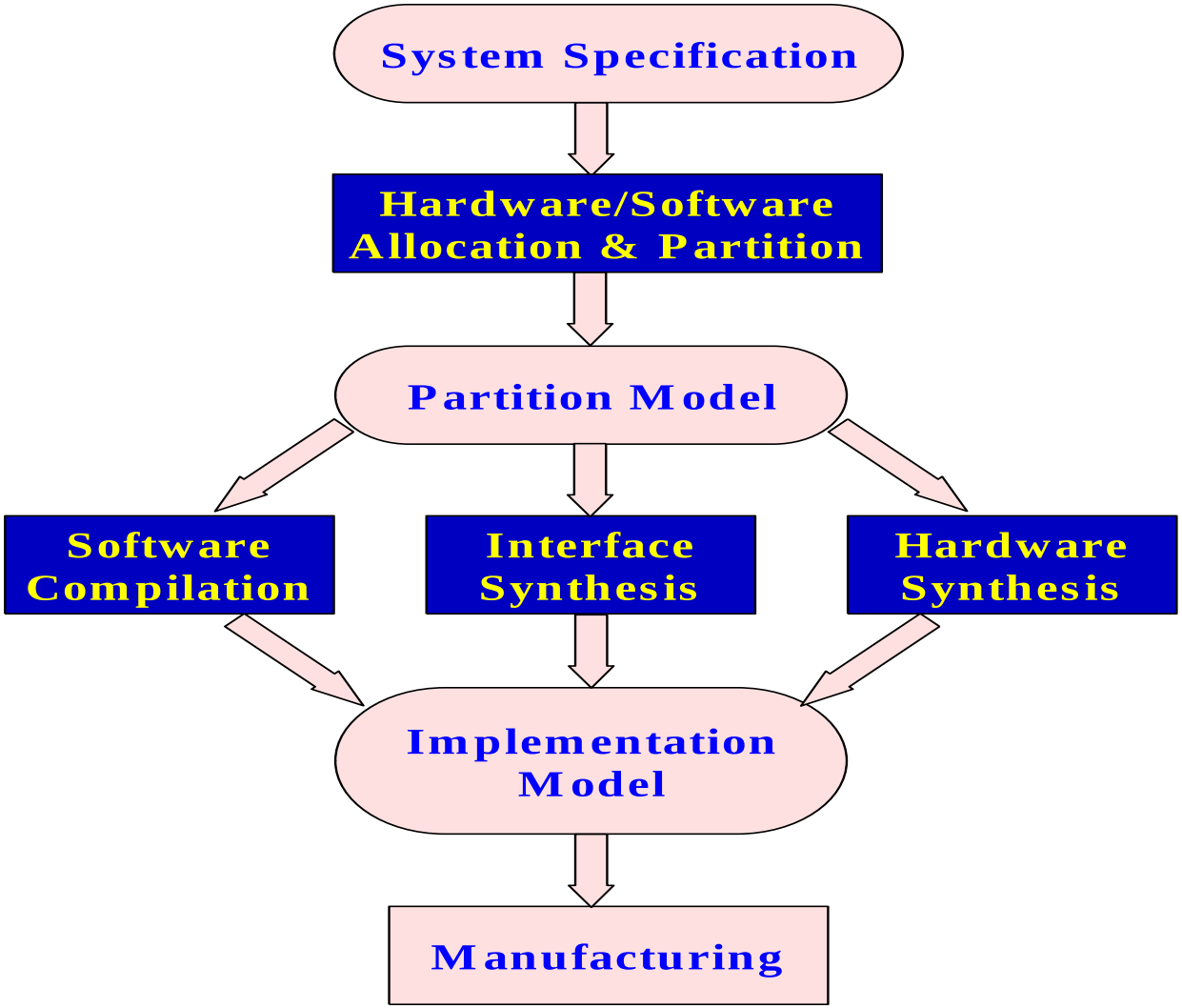
Qin Shengchao[1,3], He Jifeng[2], Qiu Zongyan[3], Zhang Naixiao[3]

[1]Singapore-MIT Alliance, National University of Singapore

[2]UNU/IIST

[3]School of Mathematical Sciences, Peking University

Hardware-Software Co-Design: General Methodology



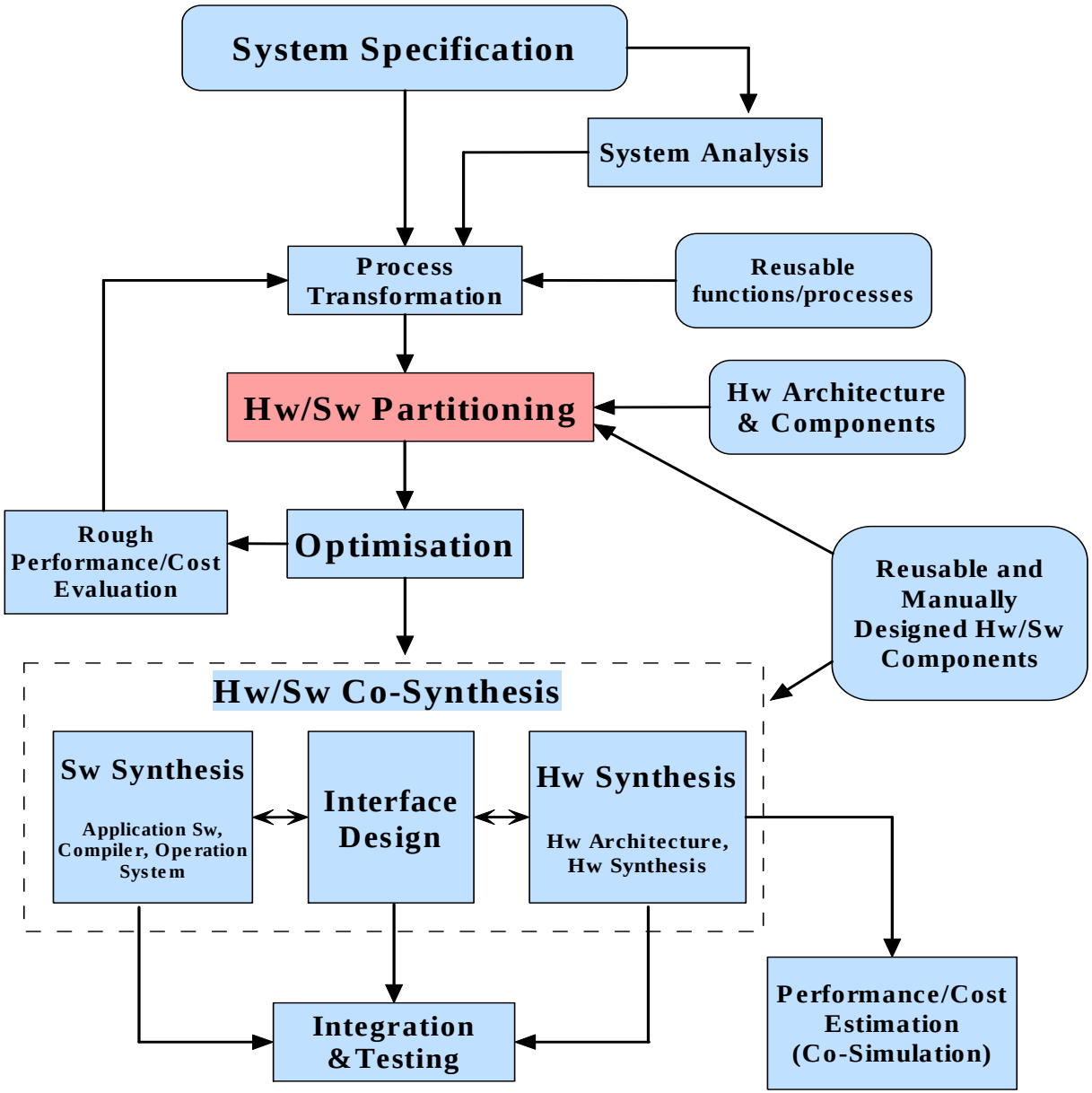
Our Hw-Sw Co-Design Approach & Motivations

An integrated approach to hardware-software co-design:

- ☞ introducing formal techniques (*algebraic approach*) into the co-design process to the utmost extent;
- ☞ integrating program analysis techniques with formal development process (transformation/refinement).

Motivations:

- ✌ existing successful applications in program algebra: ProCoS
- ✌ apt to ensure the correctness for the design process
- ✌ facilitate to exploit CAD tools with high reliability


System Specification
System Analysis
Process Transformation
Reusable functions/processes
Hw/Sw Partitioning
Hw Architecture & Components
Rough Performance/Cost Evaluation
Optimisation
Reusable and Manually Designed Hw/Sw Components
Hw/Sw Co-Synthesis
Sw Synthesis
Application Sw, Compiler, Operation System
Interface Design
Hw Synthesis
Hw Architecture, Hw Synthesis
Integration &Testing
Performance/Cost Estimation (Co-Simulation)

Hardware/Software Partitioning

- Implementation-oriented computation and resources *allocation*
- *Partitioning* requirement specification into HW and SW specifications
- Related work:
  - informal: focusing on the algorithmic aspect
  - formal:
    - ✔ Sampaio et. al : Splitting and Joining
    - ✔ Our former work: Hw/Sw Partitioning in Occam

Main Contribution

In this paper we

- propose an algebraic approach to hardware-software partitioning in Verilog HDL;
- design a set of compositional algebraic rules to hw/sw partitioning;
- explore algebraic proofs for all the partitioning rules
- investigate hw/sw partitioning for both kernel specification and whole environment-driven system

The Verilog Hardware Description Language

Main features:

- Event-driven computation
- Shared-variable communication
- Both behavioral and structural modelling

Syntax:

$$P ::= S \mid P \parallel P \mid var\, x \bullet P$$

$$S ::= PC(\text{primitive command}) \mid S; S \mid if\, b\, S\, else\, S \mid while\, b\, S$$
$$\mid (g\, S) [\![\ldots [\![(g\, S) \mid always\, S \mid case\, (e)\, (pt\ S) \ldots (pt\ S)$$

where

$$PC ::= v := e \mid skip \mid \perp \mid \rightarrow \eta_v \mid v := cg\ e$$

$$g ::= \#\Delta \text{ (time delay)} \mid eg \text{ (event control)} \mid \rightarrow \eta_v \text{ (output event)}$$

$$\eta_v ::= \sim v \text{ (value change)} \mid \uparrow v \text{ (value rising)} \mid \downarrow v \text{ (value falling)}$$

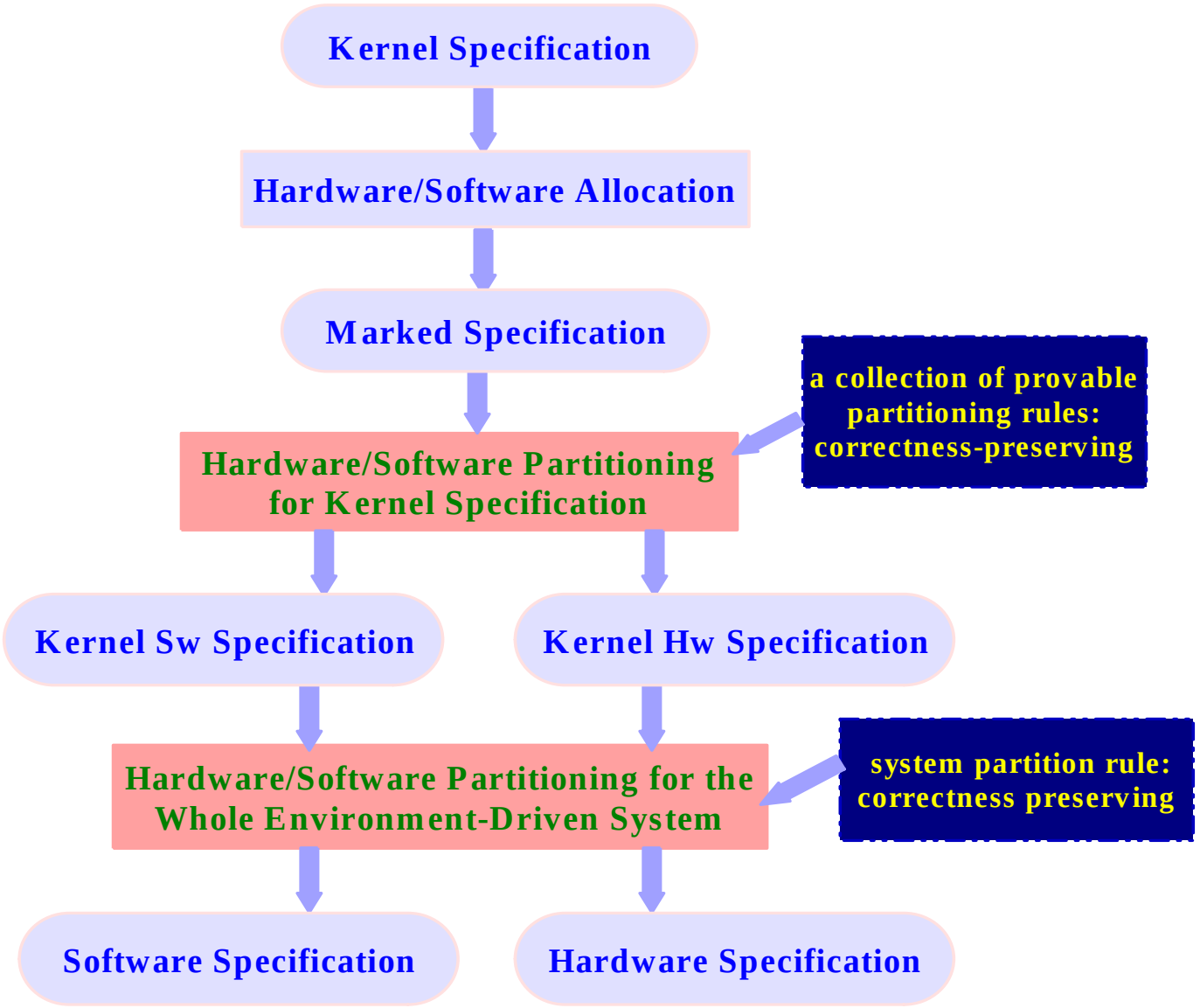
Related Work — Formal Semantics of Verilog

- Simulator-based interpretation: Gordon95
- Operational semantics: HeXu00, LiHe01, $\cdots$
- Denotational semantics: ZhuHe00
- Algebraic semantics: He02
- Linking different semantics: ZhuBowenHe01, He02

Our application:

$$\textit{algebraic laws} \Longrightarrow \textit{derived specific laws} \Longrightarrow \textit{hw/sw partitioning}$$

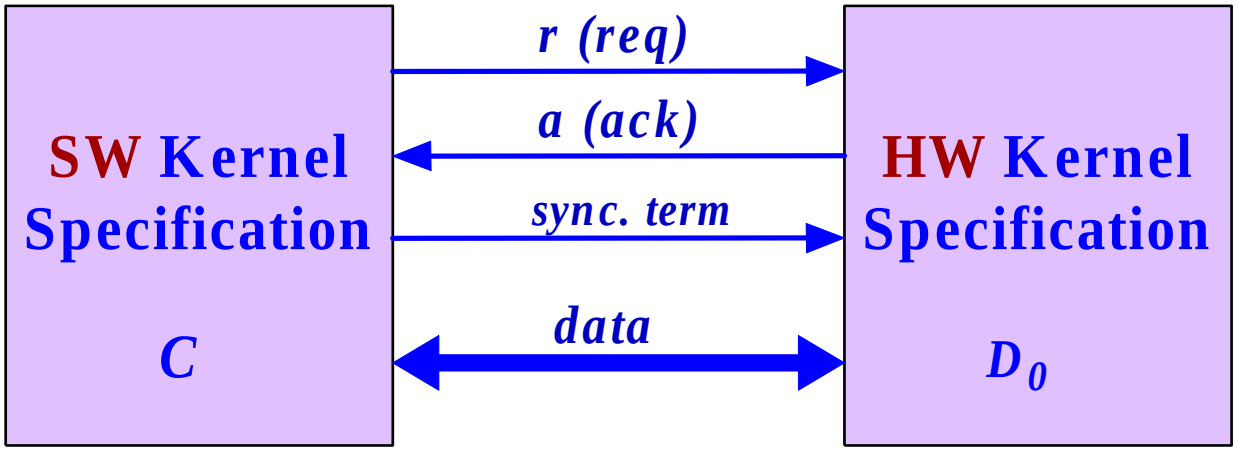
Hardware/Software Partitioning Strategy



Phase 1: Hw/Sw Splitting for Kernel Specification

- Specification language: a sequential subset of Verilog
- Hardware/Software target architectures
- Syntax-based splitting rules

Target Kernel Architecture



Kernel Specification for Software

A subset of Verilog $CP(r, a)$ composed of

1. An event control insensitive process not containing variables r, a;
2. $\rightarrow \eta_r; C; @\eta_a$, where C is a member of $CP(r, a)$ not mentioning r, a;
3. $C_1; C_2$, or $\mathit{if}\, b\, C_1\, \mathit{else}\, C_2$, or $C_1 \sqcap C_2$, or $(g_1\, C_1) \,[\!]\, (g_2\, C_2)$, where $C_1, C_2, g_1, g_2 \in CP(r, a)$;
4. $\mathit{while}\, b\, C$, where $C \in CP(r, a)$.

Kernel SW Specification: $(C; \rightarrow \eta_\varepsilon)$

Kernel Specification for Hardware

Kernel HW Specification:

$$D_0 \;=\; \mu X \bullet ((@\eta_r\, M; {\to}\eta_a; X) \,[\!]\, (@\eta_\varepsilon\; skip))$$

where $M \;=_{df}\; case\,(id)\,(p_1\, M_1)\ldots(p_n\, M_n)$ is a case construct not containing r, a, ε.

Theorem:

$$(C_1; C_2; {\to}\eta_\varepsilon) \parallel D_0 \;=\; ((C_1; {\to}\eta_\varepsilon) \parallel D_0);\; ((C_2; {\to}\eta_\varepsilon) \parallel D_0)$$

Syntax-based Splitting Rules: Bottom-Up Style

Bottom-Up Rule for Sequential Composition

$$\frac{Split_V(S_i, C_i, D^0), i = 1, 2 \qquad Var(S_1) = Var(S_2)}{Split_V(S_1; S_2,\ C_1; C_2,\ D^0)}$$

The *Split* Predicate

$$
\begin{aligned}
&Split_V(S, C, D^0) \;=_{df}\\
&\qquad S \;\sqsubseteq\; ((C; \rightarrow \eta_\varepsilon) \parallel D^0) \wedge\\
&\qquad (C \in CP_\varepsilon(r, a)) \;\wedge\; (D^0 \in DP_\varepsilon(r, a)) \wedge\\
&\qquad (V \subseteq Var(C; \rightarrow \eta_\varepsilon) \cap Var(D^0)) \wedge\\
&\qquad (V \cap Occ Var(S) = \emptyset)
\end{aligned}
$$

Syntax-based Splitting Rules: Top-Down Style

Top-Down Rule for Conditional

$$\frac{\begin{array}{c} Split_V(S_i,\ C_i,\ D_i) \\ Var(S_1) = Var(S_2) \\ mergable(D_1, D_2) \end{array}}{Split_V(\mathit{if}\ b\, S_1\, \mathit{else}\, S_2,\ \ \mathit{if}\ b\, C_1\, \mathit{else}\, C_2,\ \ \mathit{int}(D_1, D_2))}$$

Atomic Commands Splitting — Timed Assignment

For timed assignment $(v := f(x,c))_n$, where $f \in HW,\ x \in SW$:

$Split_B(S = ((v := f(x,c))_n),\ C,\ D)$, where

$C =_{df} ((id := 1)_0; (lx := x)_0; \rightarrow \eta_r; @\eta_a; (v := ly)_0)$, and

$$D =_{df} \mu X \bullet \left(\begin{array}{l} (@\eta_r\ case\ (id)\ (1\ (ly := f(lx,c))_n); \rightarrow \eta_a; X) \\ [\!] \ (@\eta_\varepsilon\ skip) \end{array} \right)$$

A Toy Example — the Source Program

```
w := u + v;
#1; →(w_ready);
@(z_ready); if (z = u) (v := u × v) else skip;
while ((b && w ≤ u × v)_{n1}){
        u := u + w;
        w := ((u − v) × (u + v))_{n2};
        #1; →(w_ready)
}
```

Kernel Specification for Hardware

$$\mu X \bullet ((@\eta_r \ \mathsf{case}\ (\mathsf{id}) \left\{ \begin{array}{ll} 1 & \mathsf{lv} := \mathsf{v} \\ 2 & \mathsf{v} := \mathsf{lv} \\ 3 & (\mathsf{lb} := \mathsf{b}\ \&\&\ \mathsf{w} \leq \mathsf{lu} \times \mathsf{v})_{n_1} \\ 4 & (\mathsf{lw} := (\mathsf{lu} - \mathsf{v}) \times (\mathsf{lu} + \mathsf{v}))_{n_2} \end{array} \right\} ; \rightarrow \eta_a ; X) [\![(@\eta_\varepsilon\ skip))$$

Kernel Specification for Software

$$\mathsf{id} := 1;\ \rightarrow\eta_r;\ @\eta_a;\ \mathsf{w} := \mathsf{u} + \mathsf{lv};$$

$$\#1;\ \rightarrow(\mathsf{w\_ready});\ @(\mathsf{z\_ready});$$

$$\mathsf{if}\ (\mathsf{z} = \mathsf{u}) \left(\begin{array}{l} \mathsf{id} := 1;\ \rightarrow\eta_r;\ @\eta_a;\ \mathsf{lv} := \mathsf{u} \times \mathsf{lv}; \\ \mathsf{id} := 2;\ \rightarrow\eta_r;\ @\eta_a \end{array} \right) \mathsf{else\ skip};$$

$$\mathsf{id} := 3;\ \mathsf{lu} := \mathsf{u};\ \rightarrow\eta_r;\ @\eta_a;$$

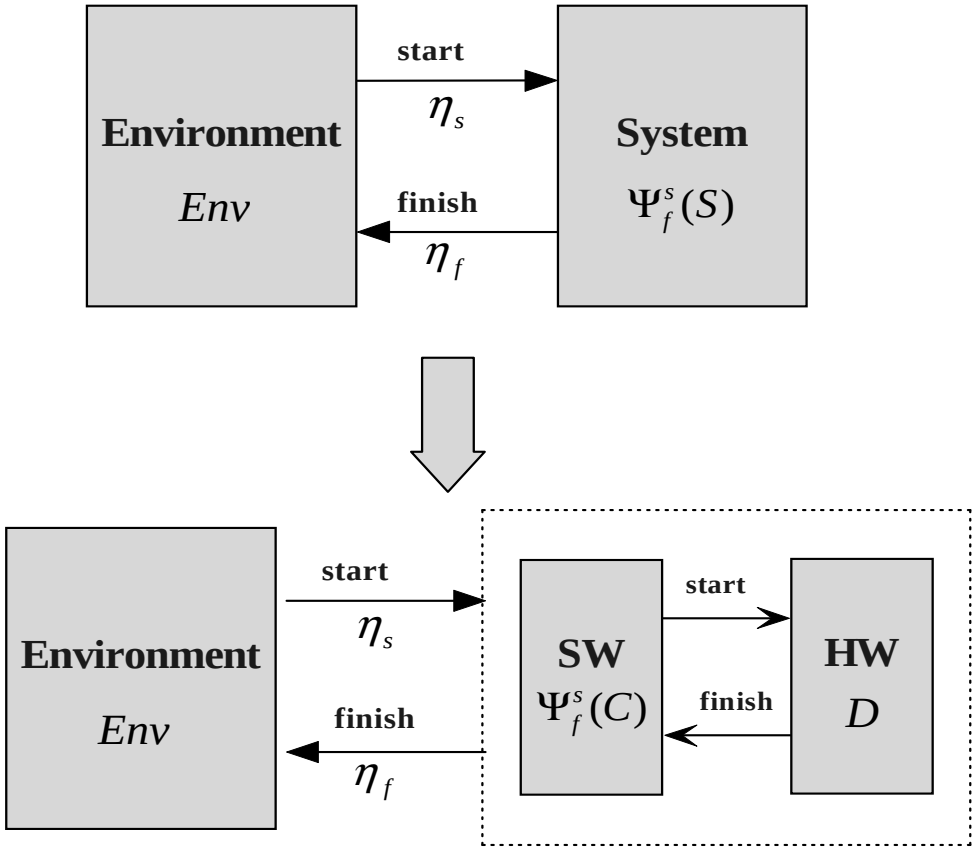
$$\mathsf{while}\ (\mathsf{lb})\{$$

$$\mathsf{u} := \mathsf{u} + \mathsf{w};\ \mathsf{id} := 4;\ \mathsf{lu} := \mathsf{u};\ \rightarrow\eta_r;\ @\eta_a;\ \mathsf{w} := \mathsf{lw};$$

$$\#1;\ \rightarrow(\mathsf{w\_ready});\ \mathsf{id} := 3;\ \mathsf{lu} := \mathsf{u};\ \rightarrow\eta_r;\ @\eta_a$$

$$\}$$

Phase 2: Derive Hw/Sw Partitioning for the System



System Specification & Partitioning Rule

System:

$$\Psi^s_f(S) \;=_{df}\; always\,(@\eta_s\, S; \rightarrow \eta_f)$$

Environment:

$$Env \;=_{df}\; always\,(\rightarrow \eta_s; @\eta_f)$$

Partitioning rule:

$$\frac{Split_V(S,\ C,\ D)}{Part(\Psi^s_f(S),\ \Psi^s_f(C),\ \Psi^r_a(M))}$$

where $Part(S,\ C,\ D) \;=_{df}\; ((S \parallel Env) \sqsubseteq (C \parallel D \parallel Env))$

$$\Psi^v_u(P) \;=_{df}\; always\,(@\eta_v\, P; \rightarrow \eta_u)$$

Conclusion

In this paper,

- we apply Verilog algebra to hardware/sofware decomposition; and
- design and prove a complete set of compositional rules for both the kernel part and the whole specification.
- Based on our partitioning process, we can straightforwardly develop an automatic partition tool for co-design; moreover,
- our results (hw/sw specifications) are still in an abstract level, which facilitates us to pursue further transformation/refinement.

Future Work

As part of future work, we shall

- investigate further *optimization* and *reconfiguration* on the hardware specification generated by our partitioning algorithm, and connect it with the work on formal synthesis in Verilog (IyodaHe01); and
- involve more *program analysis techniques* into the partition and co-synthesis process for performance estimation; and
- apply our approach to *industrial case studies*.