슈퍼스칼라 프로세서 시뮬레이터의 생성을 위한 Attributed AND-OR 그래프

김준경* · 김탁곤*

Attributed AND-OR Graph for Synthesis of Superscalar Processor Simulator

Jun Kyoung Kim · Tag Gon Kim

Abstract

This paper proposes the simulator synthesis scheme which is based on the exploration of the total design space in attributed AND-OR graph. Attributed AND-OR graph is a systematic design space representation formalism which enables to represent all the design space by decomposition rule and specialization rule. In addition, attributes attached to the design entity provides flexible modeling. Based on this design space representation scheme, a pruning algorithm which can transform the total design space into sub-design space that satisfies the user requirements is given. We have shown the effectiveness of our framework by (i) constructing the design space of superscalar processor in attributed AND-OR graph, (ii) pruning it to obtain the ARM9 processor architecture, (iii) modeling the components of the architecture and (iv) simulating the ARM9 model.

I. Introduction

The progress of silicon technology enables us to implement more functionalities on a given chip area. This development of silicon technology has well been addressed by Moore's law[1]. Moore's law says that the amount of device integrated on a given silicon area doubles every year. This law has held until the late 1970s, but the doubling period has increased to eighteen month since late 1970s or early 1980s. As a result, current silicon technology can integrate tens of millions of gates on a single die, with manufacturing process around zero point one micrometer.

Designing such a complex system requires complex design space and rigorous verification of the candidate designs. This paper suggests the design space representation scheme by (i) representing the full design space of superscalar processor, (ii) providing a means to specify the design constraints or user constraints, (iii) pruning a total design space based on a set of constraints.

First, the design space of processors represented will be given in the attributed AND-OR graph.

* 한국 과학 기술원 전자전산학과 전기공학전공

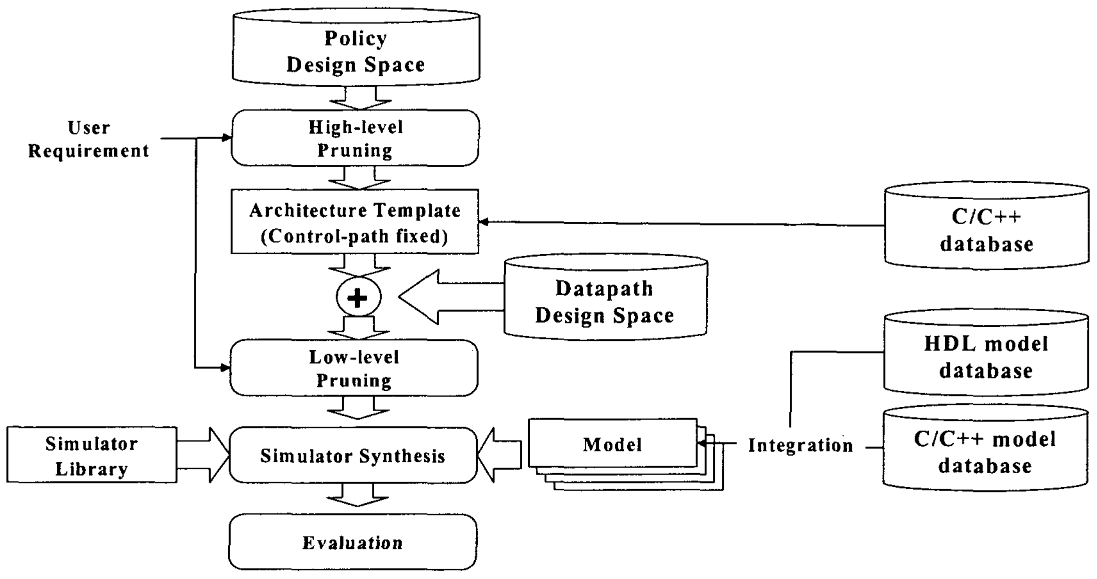

Fig. 1 Proposed Framework

Attributed AND-OR graph provides a useful tool to represent a design space, specify a set of constraints and show the pruned design entry. We will show how to specify the user constraints in terms of system parameters. By translating the user requirements and giving them as inputs to our framework, we can obtain a set of valid processor architectures. By considering and reflecting the knowledge base and user constraints can we choose satisfiable designs from the valid processor architectures. As a last step, simulator synthesis is performed.

Ⅱ. Overall Framework

Fig. 1 shows the overall architecture of our framework. We propose a hierarchical framework for processor simulator synthesis. First, we can obtain the architecture template model by deciding the various policies. For decided architecture template, operational model in C/C++ can be integrated to synthesize the simulation model. After that, one can determine the data-path architecture which is related with the number and the types of execution units used. The operational model for the data-path execution unit can exist in the C/C++ model base, or even in the HDL model base. One can synthesize the complete simulation model by integrating these models. In the case that the models have different form, integration after translation is inevitable.[2]

The important ones to be considered in designing a design space representation scheme is as follows.

- Guaranteeing correctness of the design space
- The completeness of representing a design space
- The easiness of representing a design space
- The easiness of extending the design space
- The easiness of reflecting one's own, but very valuable knowledge base obtained at the practical processor design

Based on the above requirements, we have decided the attributed AND-OR graph is one of the best formalism to represent the processor architecture. The additional advantage of attributed AND-OR graph is the easiness of construction and intuitive access to the design space because this formalism depicts the design space with two simple rules, AND-rule and OR-rule.

Ⅲ. Formalism for Design Space

1. Attributed AND-OR Graph(AAOG)

Attributed AND-OR graph expresses the design space of any object in the world by alternating the AND-rule and OR-rule. An AND-rule implies a decomposition relation between design entities. An OR-rule defines the selection relationship between design entities.

The attributed AND-OR graph is formalized as follows.

Attributed AND-OR Graph *is a directed graph* $G = (V, E_{AND}, E_{OR}, ATTR, va)$ *where*

- V : *a vertex set*
- E_{AND} : *a set of AND-edge*
- E_{OR} : *a set of OR-edge*
- $ATTR$: *a set of attributes*
- va : *attribute mapping function*

with the constrains

- ■ V, $EAND$, EOR , $ATTR$: *finite sets;*
- ■ $E_{AND}, \subseteq V \times V$: *edge for AND-relationship*
- ■ $E_{OR}, \subseteq V \times V$: *edge for OR-relationship*
- ■ $V \rightarrow 2^{ATTR}$: *a function which maps a vertex to zero or more attributes.*

One can specify the constraint relationship for a vertex, or a combination of vertices. There are two types of constraint, local constraint and global constraint.

- ● ***local constraint*** *is defined for AAOG as*
  - ■ *local:* $V \times 2^{ATTR} \rightarrow \{true, false, do\_not\_care\}$

    with the constraints for $(v1, v2) \in EOR$,
    - ● $local(v2, va(v2)) = true$: *the v2 entity should always be selected as a choice of the v1 entity*
    - ● $local(v2, va(v2)) = false$: *the v2 entity should not be selected as a choice of the v1 entity*
    - ● $local(v2, va(v2)) = do\_not\_care$: *there is no constraint for this entity v2*

The local constraint is used to reflect the designer's intention to use an entity. The second type of constraint, global constraint, specifies the similar relationship between vertices as follows.

- ● ***global constraint*** *is defined for AAOG as*
  - ■ *gobal:* $V \times V \times 2^{ATTR} \times 2^{ATTR} \rightarrow \{true, false, do\_not\_care\}$
    - ● $global(v1, v2, va(v1), va(v2)) = true$: *if one decided to use the design entity v1, v2 should always be selected*
    - ● $global(v1, v2, va(v1), va(v2)) = false$: *if one decided to use the design entity v1, v2 should not be selected*
    - ● $global(v1, v2, va(v1), va(v2)) = do\_not\_care$: *there is no constraint for these entities v1 and v2*

With these two types of constraints, one can specify the physically possible combination of processor architecture.

2. Pruning Algorithm

Based on the formal definition of attributed AND-OR graph, we have constructed the prune algorithm. This algorithm transforms the original, total design space into the sub design space by considering the local and global constraints. The user requirements are reflected as a form of user constraints.

```
Algorithm Prune
   Input : Gin = (Vin, EinAND, EinOR, ATTRin, vain)

   Output : Gout (Vout, EoutAND, EoutOR, ATTRout, vaout)
functions:
   local(x, y) : local constraint function
   global(x, y) : global constraint function
   global'(x, y) : newly generated global constraint function for Gout

 begin prune
   reset(Gout):
   x = root of Gin
   insert(Vout, x)// step 1 : local constraint reflection
   while x <> null
       for y of (x, y) in EinOR begin
          if(local(y,vain(y) ) = true) begin
               insert(Vout, y):
               insert(EoutOR, (x, y)):
          end if
          else if(local(y, vain(y)) = false) begin
               remove(Gin, y):
           end if
           else begin
                insert(Vout, y):
                insert(EoutOR, (x, y)):
           end if
       end for
       for y of (x, y) in EinAND begin
           insert(Vout, y):
            insert(EoutAND, (x, y)):
       end for
   retrieve(x, Vin)
   end for
    for every x in Vout
       if(global(x, y) = true  y is in Vout)
           global'(x, y) := true:
       else if(global(x, y) = false  y is in Vout)
           global'(x, y) := false:
   end for
end Prune
```

Ⅳ. Example

Fig.2 shows the design space exemplified by superscalar processor architecture[3]. For example, a processor architecture can be decomposed to the fetch unit(fetch__unit), decode unit (decode__unit), registers and rename unit(register__rename), execution unit and shelving buffer in front of it(shelve__EU), and reorder buffer (ROB). There are two candidates for the fetch__unit, predecoded__f etch and only__fetching. In case of superscalar processor, it takes much time to decode an instruction and check the dependency between instructions. Therefore there can be a predecoder between instruction cache and instruction buffer to reduce the burden of the decoding block. As in the figure, the (fetch__unit, predecoded__fetch) and (fetch__unit, only__ fetching) is connected with OR edge. That is, one can select one of the two candidates. We can observe the global constraints in the figure between (predecoded__fetch, 1__leve__decode) and (only__fetching, 2__lev__decode). They are always true relationship. That is, when the predecoded__fetch is selected for the fetch__unit scheme, the 2__lev__decode should always be chosen for the decode__unit. To synthesize a processor simulator, the coupling relations are required, and they are specified at the AND-edge of the figure. The Fig.3 shows the pruned design space. By specifying the user constraints, and reflecting it to the total design space shown in Fig.2, we could obtain the pruned design space shown in Fig.3. We wrote down the user requirement in the way that the pruned architecture is ARM9 processor. Fig.4 shows the resulting architecture of the simulation model. The operational model for each entity is assumed to be stored at the model database.

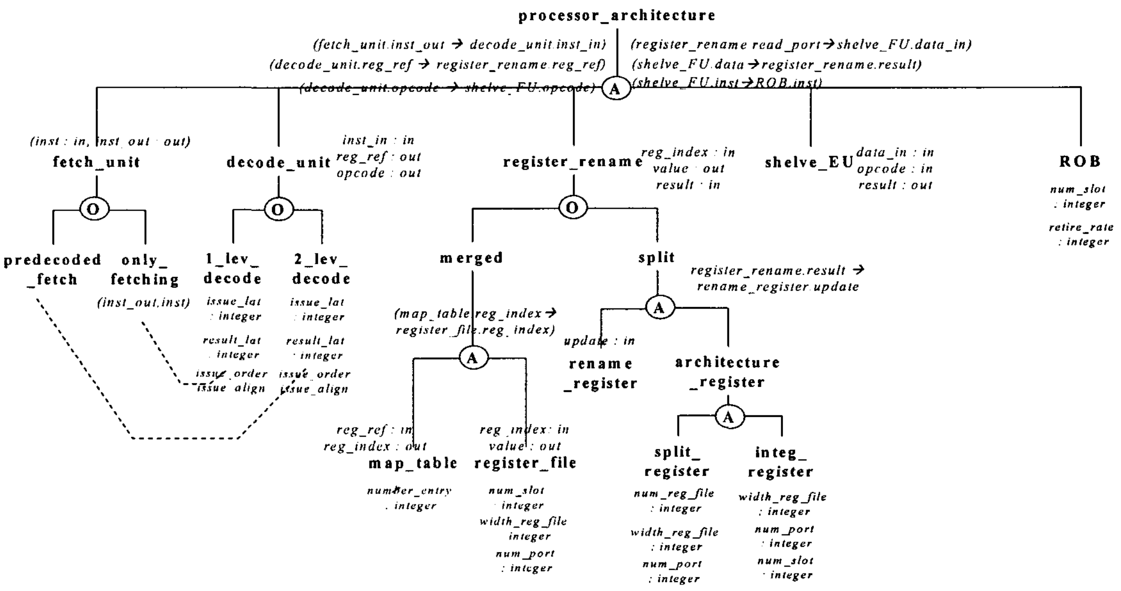

Fig. 2 Design Space of Superscalar Processor Architecture

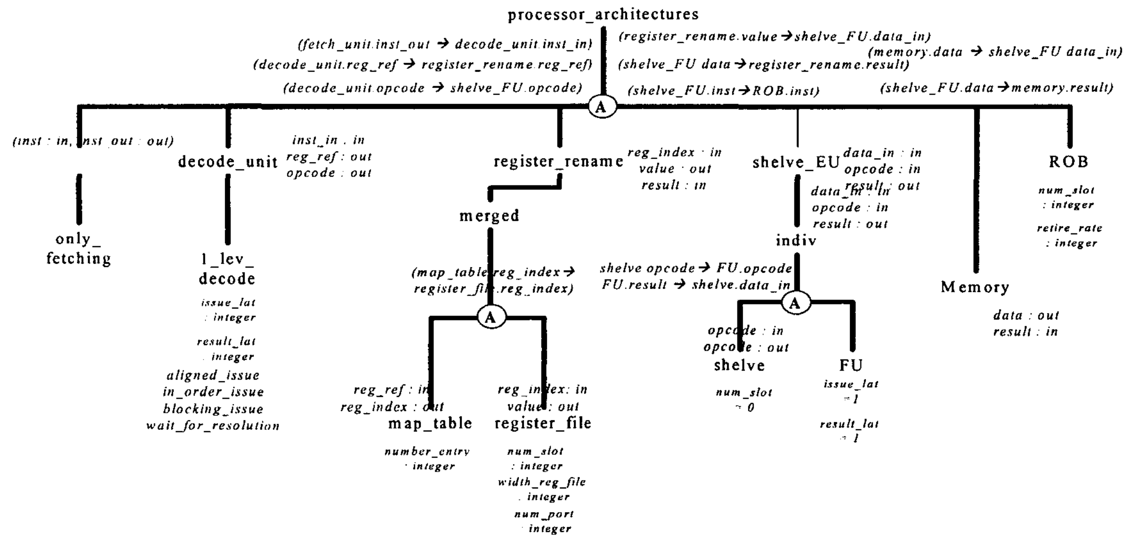

Fig. 3 Pruned Design Space

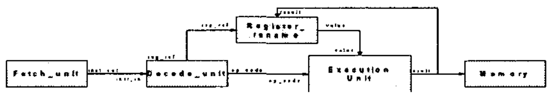

Fig. 4 Resulting Simulation Configuration

By simulating this model with the pipeline simulator [4], we could obtain the simulation result as in Fig.5.

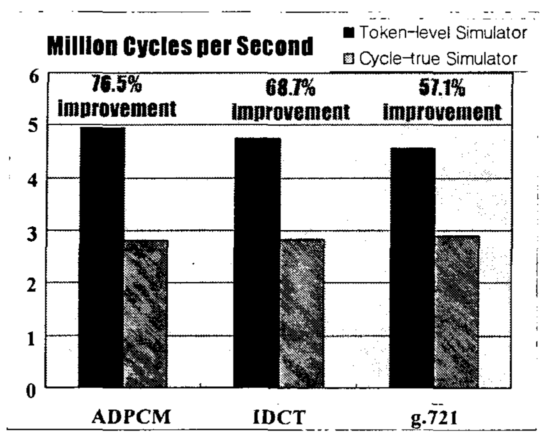

Fig. 5 Simulation Results

V. Conclusion

This paper proposes the simulator synthesis scheme which is explored from the total design space in attributed AND-OR graph. In addition, we have defined two types of constraints with which a modeler can reflect his or her own design objectives or design knowledge. We have shown how the design space of superscalar processors can be constructed using the formalism. The ARM9 processor has been achieved by exploring the design space, and simulated.

References

[1] Moore, G. "Progress in Digital Integrated Electronics", *IEEE International Electronic Devices Meeting*, 1975

[2] Jun Kyoung Kim, Tag Gon, Kim, "DHMIF: DEVS-based Hardware Model Interchange Format", Proceedings of 13th European Simulation Symposium, 2001

[3] Deszo Sima, Terence Fountain and Peter Kacsuk, *Advanced Computer Architectures A Design Space Approach*, 1997, Addison-Wesley

[4] Jun Kyoung Kim, Tag Gon Kim, "Trace-driven Rapid Pipeline Architecture Evaluation Scheme for ASIP Design", *Proceeding of Asia South Pacific Design Automation Conference*, 2003, pp129-134