PAPER

A Plan-Generation-Evaluation Framework for Design Space Exploration of Digital Systems Design

Jun Kyoung KIM[†a)], ***Student Member and*** **Tag Gon KIM**[†b)], ***Nonmember***

SUMMARY Modern digital systems design requires us to explore a large and complex design space to find a best configuration which satisfies design requirements. Such exploration requires a sound representation of design space from which design candidates are efficiently generated, each of which then is evaluated. This paper proposes a plan-generation-evaluation framework which supports a complete process of such design space exploration. The plan phase constitutes a design space of all possible design alternatives by means of a formally defined representation scheme of attributed AND-OR graph. The generation phase generates a set of candidates by algorithmic pruning of the design space in an attributed AND-OR graph with respect to design requirements as well as architectural constraints. Finally, the evaluation phase measures performance of design candidates in a pruned graph to select a best one. A complete process of cache design is exemplified to show the effectiveness of the proposed framework.

key words: *design space exploration, plan-generation-evaluation framework, graph pruning, attributed AND-OR graph*

1. Introduction

The progress of silicon technology enables us to implement a highly complex system on a given chip area. The design of such a system requires the exploration of large design space not only for functional verification but for performance measurement. The most important factor in such a design is an efficient design framework which meets the time-to-market requirement. Thus, required is a design framework and an associated tool which can reduce cost of design space exploration. Such cost includes the design time for construction of a set of design candidates and the evaluation time for correctness verification and performance measurement.

Design space exploration mainly consists of three phases: plan, generation and evaluation. The plan phase constructs a design space of a family of design alternatives which includes architectural variants and different parameters for a given architecture. The generation phase generates feasible design candidates from the design space. Finally the evaluation phase evaluates the generated candidates. Thus, a reduction of the cost of the design space exploration requires each of the three phases to be performed in an efficient manner.

Much research has been done to attack on the design space exploration problem. However, most of them dealt with efficient generation of design candidates [1]–[3] and fast evaluation of each candidate [4], [5]. Most of them deals with the design space construction problem as just a parameterization one, with which a designer can vary some fixed parameters [6] and the degree of parameterization is totally decided by the tool builder. Schemes or algorithms can not be extended easily with the approach unless they are considered in the first design. In other words, they have limited extendibility due to the lack of expression power on architecture parameterization.

Binary Decision Diagram (BDD) is an approach to the solution space representation, which is a set of valid configurations [7]. Possible values for a parameter are encoded as a sequence of binary value. By configuring the resulting binary variables, explorer can get a set of valid configurations. However, it is hard to represent the and-rule, which can be called as a decomposition rule, with the approach. Moreover, we can not obtain the hierarchical framework without the and-rule, which is very important in complex digital systems design.

This paper proposes a plan-generation-evaluation framework for efficient design space exploration of digital systems design, with an emphasis on design space representation in the planning phase. The main goal of the representation is two folds: representation in formal semantics and expressive power of design alternatives. To meet such a goal, a formalism of attributed AND-OR graph (AAOG) is defined for design space representation. The AND and OR relations in the graph can represent not only a collection of components (AND relation) to form a system but variants of alternatives (OR relation) to be used for a given component. Moreover, attributes attached at such relations can represent ranges of parameters and constraints on selection of a set of alternatives located at different sub-modules.

AND-OR graph is a general tool for representation of information structure with alternative entities for which a search is performed to find an optimal solution. Basic definition of AND-OR graph can be found in [8] and some optimal search algorithms for such graph are found in [9], [10]. The basic AND-OR graph definition has been extended to adapt different applications which include floorplanning at digital system design [11], model checking./testing [12], network modeling [13], assembling /disassembly line modeling [14], and others. This paper extends the general AND-OR graph to Attributed AND-OR graph with an attempt to support "design space exploration" of digital systems. The main purpose of our graph is to express alternatives for ar-

Manuscript received June 16, 2005.
Final manuscript received November 22, 2005.

[†]The authors are with the Department of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea.
a) E-mail: jkkim@smslab.kaist.ac.kr
b) E-mail: tkim@ee.kaist.ac.kr

DOI: 10.1093/ietfec/e89–a.3.772

chitectural components as well as parameter values for design space representation. The main differences between our AAOG and existing ones are: (1) AAOG does not represent cost within the graph, which is evaluated by simulation; (2) Multiple AND-relation is represented to model a number of identical components as a subsystem of a system; (3) Attributes are attached to a vertex, the values of which represent parametric design space; (4) A set of constraints is attached to a vertex which defines valid combination of components to form a (sub)system as well as holds designers' knowledge for a system under design. In the above sense, the proposed AAOG is a new one which has not been previously used for the design space exploration problem in digital systems design.

The similar approach has been reported in [15] which proposed a relational algebraic framework for VHDL models management. The framework employed SES (system entity structure) formalism [16] to represent a family of VHDL models and supported an associated tool for models management which was implemented in a relational database system. However, the framework focuses on a specific simulation tool, the VHDL simulator, rather than design space exploration. Our work here explicitly defines the plan, generation and evaluation steps which are inevitable to design space exploration. Also, our work supports not only management of the design entities as in [15], but also design of a new algorithm or a system. To this end, we divide a design space in two sub-spaces: operational design space and structural design space. Configuration of algorithmic parameters at the operational design space, and then synthesis of the configuration at the structural design space help to reflect new algorithms or introduce new processing elements. Thus, the framework proposed in this paper can be more design-centric.

2. Proposed Framework

Figure 1 shows the design space exploration employing plan, generation and evaluation framework. Functional requirement enables us to make a set of designs/models, which we call functional design space. Technology requirement enables to check the variability of the models against it. Then the realizable set of models which comply with functional/technology requirement constitute the design space. This is plan phase. At generation phase, one can decrease the design space with the user constraints, which reflect the designers' knowledge. The resulting design space is called a pruned design space. With the model base which contains the operation of models, model synthesis is performed. To evaluate, all the models are compiled with the simulation library and the evaluation is performed by running the resultant simulator.

The three steps are tightly coupled but what schemes or algorithms are used in each step does not affect one another. For example, the form or semantics we use to represent a design space does not constrain the generation algorithm as in Fig. 2.

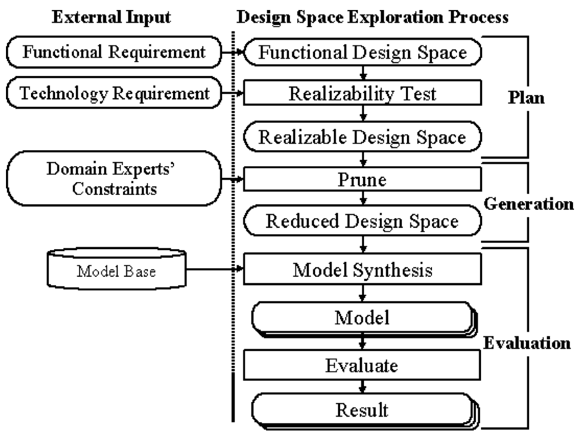


Fig. 1 Proposed framework.

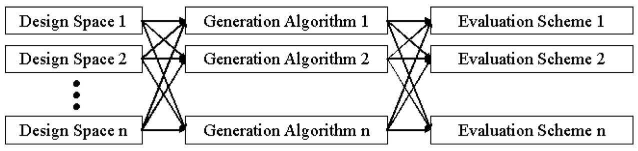


Fig. 2 Free combination of each step.

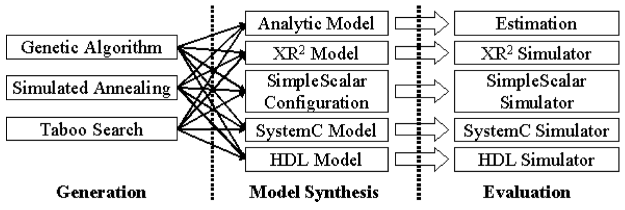


Fig. 3 Objective-driven generation-evaluation combination.

Figure 3 shows this property more obviously with the relation between generation and evaluation. If a designer needs short evaluation time, he or she can possibly adopt the analysis method for the evaluation scheme. If the designers' main concern is the performance of processor architecture, Simplescalar or XR2 simulator [17] is adequate. A designer has only to modify the model synthesis routine. It does not affect what kind of pruning algorithm such as genetic algorithm, simulated annealing, or taboo search, will be used.

3. Representing Design Space

3.1 Representation Formalism: Attributed AND-OR Graph

Attributed AND-OR graph expresses the design space of any structure by using the AND-rule and OR-rule. An AND-rule implies a decomposition relation between design entities. OR-rule defines the selection or specialization relationship between design entities. The attributed AND-OR graph is formalized as follows.

Figure 4(a) is an example of a design space of a digital system. A digital system is assumed to consist of three components: proc, peripherals and memory_subsystem.

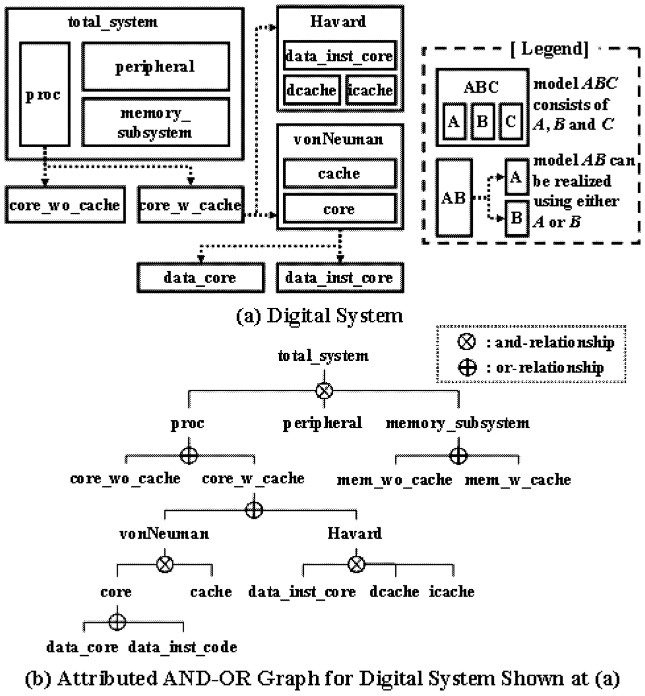


Fig. 4 Attributed AND-OR graph.

Two kinds of core, core_wo_cache and core_w_cache, are available for the proc entity. Core with on-chip cache, core_w_cache, has different forms depending on the internal structure. Harvard architecture has separate on-chip cache for instruction and data. On the contrary, only one on-chip cache can exist at vonNeuman. data_core is a core that sends only data address to the cache component, and data_inst_core sends both data address and instruction address to cache component.

Figure 4(b) is the attributed AND-OR graph representation of digital system shown in Fig. 4(a). The relation connected with indicates OR-relation and the relation connected with AND-relation. total_system consists of three entities, proc, peripheral and memory_subsystem. In the case of proc entity, there are two candidates, core_wo_cache and core_w_cache. core_w_cache can be implemented to vonNeuman or Harvard. The design space of a digital system is formally represented in this way.

A vertex can have attributes. For example, word_size can be attached to the proc entity as a parameter. Let's assume that the value, 16, 32, 64, is possible for the word_size. Then, the word_size is defined as following by definition: {(word_size, 16), (word_size, 32), (word_size, 64)} Together with OR-relation, parameters constitute the design space of a target system.

3.2 Embedding Pre-Constructed Design Space

While planning, a designer may need to embed a pre-constructed design space. To use the mechanism, a designer has only to specify the following:

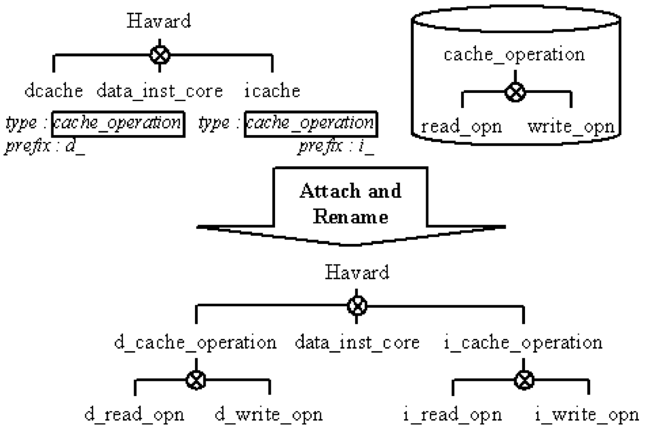


Fig. 5 Embedding pre-constructed design.

- nodes to use pre-constructed design space
- what kind of design space will be embedded
- the prefix used to differentiate the children nodes from the other nodes

Figure 5 shows a scenario of embedding the cache design space into Harvard architecture shown in Fig. 4. We assume that the cache design space is already constructed and verified while previous systems design. Then, the cache itself is independent of external environment. If a designer keeps the rules for interface for cache design space, he or she can take the cache design space freely into any designs that require cache. This scheme is used for Sects. 6 and 7.

3.3 Semantics for Constraint in AAOG

One can specify the constraint relationship for a vertex, or a combination of vertices. Constraint relationship is defined as follows

The first thing one can do with the constraint specification is to declare the physically possible combination of the vertices which constitute the design space. More importantly, one can reflect his/her own knowledge base to reduce the design space. For example, let's assume that a designer does not want to use the core_wo_cache design entities after analyzing the target application. Then, it is expressed in the following form:

const(core_wo_cache) = false

Another situation tells that core_wo_cache is not selected with mem_wo_cache. Then, we can express this constraint with the following manner:

const(core_wo_cache, mem_w_cache) = true

To investigate the effect of the constraints to design space, let's assume the design space problem as parameterization one. If the number of parameters is m and the number of candidates for each parameter is n, and all the parameters are independent of one another, then the number of design candidates becomes n^m, which is often too many to explore all the candidates. Let's assume that there is an expert about the system or application. He or she may fix a parameter to one value based on his knowledge or experience. Then the number of design candidates reduces to n^{m-1}. One of ultimate goals of this work is to provide means with which experts can store and reflect hisher own design knowledge or experience obtained during real design work in the form of constraints.

4. Generation

Figure 6 is an algorithm to compute all the candidates from design space in attributed AND-OR graph. It works in post-order. Post-order search guarantees that the design candidates set of the child node is available when we compute the candidates of a parent node. The entity nodes are classified into three types, leaf node, parent node of AND-relationship and parent node of OR-relationship. Firstly, when this algorithm meets a leaf node, it does nothing and just returns. When this algorithm meets the parent of OR-relationship, the node makes its own candidate set by performing union of the children's candidate set. Lastly, when the algorithm meets a parent of AND-relationship, it computes the parent's candidate set by performing cross product of children's candidate set

This procedure is well-described in Fig. 7, which shows the read operation of cache. The read operation read_opn is composed of five operations, address_match, update_candidate, replacement, data_fetch and line_fill. There are three specialization children nodes for the update_candidate and four specialization children nodes with replacement.

```
Algorithm      computeCandidate(nd)
input : nd of type node;
given : each node has its container cand_set to hold candidates
begin
  for all the children $c_i$ of node nd begin
    compute_candidate($c_i$);        /* post-order search */
  end for
  if (nd is a leaf node) return;
  else if (nd is a parent of OR-relation) begin
    for all the children cj begin
      nd.cand_set ← nd.cand_set ∪({(nd.namd, $c_j$.name)}×$c_j$.cand_set)
    end for
  else if (nd is a parent of AND-relation) begin
    for all the children c1, c2, ..., cj begin
      nd.cand_set ← $\prod_{k=1}^{j}$ ($c_k$.cand_set ∪{(nd.name, $c_k$.name)})
    end for
  end if
end
```

Fig. 6 Algorithm for candidates generation.

Figure 7 shows that the candidate list of update_candidate is (update_candidate, hist_update), (update_candidate, fifo_update), (update_candidate, freq_update). The candidate set of the root node of this graph is shown somewhat mathematical representation. That implies that the candidate set of read_opn is the cross product of all the children's candidate set. We will not describe the evaluation here because it depends on the evaluation tool. Instead, we will show the case where the SystemC simulation environment is adopted for evaluation step in Sects. 6 and 7.

5. Implementation of Proposed Framework

Figure 8 shows the proposed framework. We used XML (eXtensible Markup Language) to specify all the contents of attributed AND-OR graph including constraints. XML is a markup language for documents containing structured information. Structured information contains both contents and some indication of what role that contents plays [18]. We used XML because it is easy to describe hierarchical data structure in XML.

Design entities in XML go through Expat XML parser to be data structure, which in turn is translated to the SQL queries. Through MySQL++, which is a library with which our framework can communicate with MySQL server, design entity in data structure is stored in MySQL server. Once it is stored to the database, any one in any place can connect with it to obtain or update the design space.

To generate feasible designs, the design space explorer receives user specification and constraints. Firstly, design space explorer fetches all the necessary entities from MySQL server via MySQL++ library and reconstructs the design space. By performing pruning/searching, our design space explorer can obtain a set of SystemC models.

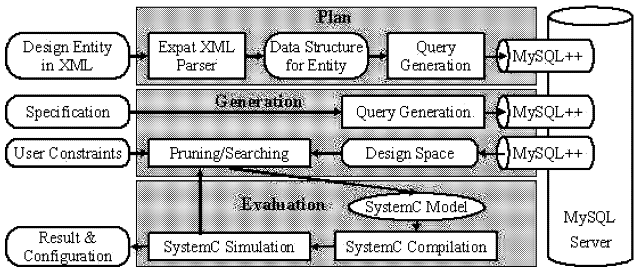


Fig. 8 Implementation of framework.

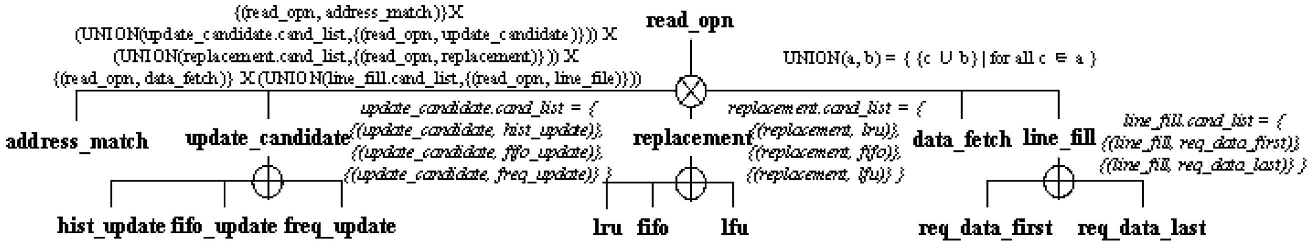

{(read_opn, address_match)}X
(UNION(update_candidate.cand_list,{(read_opn, update_candidate)})) X
(UNION(replacement.cand_list,{(read_opn, replacement)})) X
{(read_opn, data_fetch)} X (UNION(line_fill.cand_list,{(read_opn, line_file)}))
read_opn
UNION(a, b) = { {c ∪ b} | for all c ∈ a }
address_match
update_candidate
update_candidate.cand_list = { {(update_candidate, hist_update)}, {(update_candidate, fifo_update)}, {(update_candidate, freq_update)} }
replacement
replacement.cand_list = { {(replacement, lru)}, {(replacement, fifo)}, {(replacement, lfu)} }
data_fetch
line_fill
line_fill.cand_list = { {(line_fill, req_data_first)}, {(line_fill, req_data_last)} }
hist_update
fifo_update
freq_update
lru
fifo
lfu
req_data_first
req_data_last

Fig. 7 Computing all the design candidates from design space in AAOG.

SystemC compilation enables to obtain SystemC simulator. This is generation phase. Lastly, evaluation is done by running the resultant SystemC simulator. All the processes are designed so that it works in a fully automatic way without designer's intervention. The environment and the package used are listed below:

- WindowXP on Pentium4 3.0 GHz, 1 GB memory
- Compiler: Microsoft Visual Studio 6.0
- Evaluation tool: SystemC simulation environment
- XML parser: Expat 1-95-7
- Back-end DBMS: MySQL 4.1.14
- Library for connection with MySQL: MySQL++-1.7.1-1-win32-vc++

6. Example and Experiment: Cache Operation

In this section, we will give the cache operation example to investigate how one can construct a design space with the proposed methodology. Then, we will give the evaluation result of our framework. Cache is a good example for design space exploration because it has various parameters and schemes. We considered the following parameters and schemes.

- Mapping: direct-mapped, 2-way/4-way/8-way/16-way set associative mapping, fully associative mapping
- Cache size: 4K/8K/16K/32K/64K-bytes
- Line size: 4/8/16/32/64/128-bytes
- Line fill scheme: request data first, request data last
- Replacement scheme: LRU (least recently used), LFU (least frequently used), FIFO (first-in, first-out)

We wrote down all the components that constitute the cache operation and the resulting design space in attributed AND-OR graph is given in Fig. 9. We can divide the design space into two component, one from the OR relationship and the other from the parametric design space. In this example, the number of design candidates due to OR-relationship, denoted as NOR, is 6 because there are selection constraint between update_candidate and replacement. To use LRU, hist_update of update_candidate should always be chosen. The same relationship exists between fifo and fifo_update. For lfu, freq_update should be chosen. This is specified in constraints. Therefore, NOR is 6, not 18. Compared to that, the parameters can be freely combined. The number of design candidates due to parametric design space, denoted as NPARAM, is 180. Therefore, the total number of design space is 6 * 180 = 1080 because the design space due to OR-relationship and that due to parameters are independent of each other.

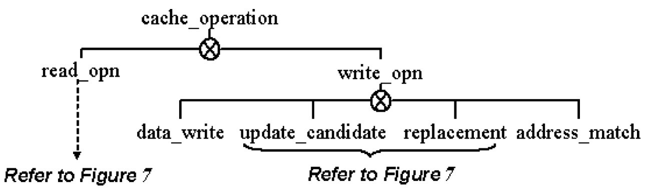


Fig. 9 Design space of cache operation in AAOG.

6.1 XML Description

Figure 10 describes the XML specification of the read_opn node that we can see in Fig. 7. It defines the interface, states, children and action. The content in the action is copied to the SystemC model as a function. One can easily understand the action of read_opn with the action component. It uses all the children nodes as functions. Therefore, we can think of this as a coupling strategy.

Figure 11 is XML specification of replacement scheme, which also appears in Fig. 6. Instead of the action section of AND-relationship, there is mapping section at the OR-relationship specification. With the mapping section, one can specify how the interface defined for the parent node is mapped to the interface of the children nodes.

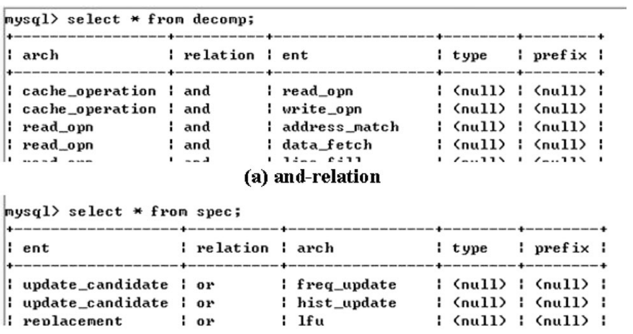
As will be explained at Sect. 6.3, one entity is transformed to one SystemC function in this abstraction level. Within the process, the sequence of the function argument should be decided based on these XML model. The order of

```
<operation name = "read_opn">
<input name = "address" type = "bitvector" width = "32"/>
<output name = "data" type = "bitvector" width = "32"/>
<input name = "unit_transfer" type = "bitvector" width = "2"/>
<state name = "hit" type = "bool"/>
<state name = "slot" type = "integer"/>
<state name = "setIndex" type = "integer"/>
<and_relation>  <child name = "address_match"></child>
                <child name = "update_candidate"></child>
                <child name = "replacement"></child>
                <child name = "data_fetch"></child>
                <child name = "line_fill"></child>
</and_relation>
<action>
<![CDATA[
  slot = address(addr_width-tag_width-1, byte_sel).to_uint();
  address_match(hit, address, slot, setIndex);
  if(!hit){ replacement(address, slot, setIndex);
            line_fill(address, slot, setIndex);  }
  data_fetch();   update_candidate(address, slot, setIndex);
]]>
</action>
</operation>
```

Fig. 10 XML specification of reand_opn node.

```
<operation name = "replacement">
<input name = "address" type = "bitvector" width = "32"/>
<output name = "slot" type = "integer"/>
<output name = "setIndex" type = "integer"/>
<or_relation>    <child name = "lru"/>
                 <child name = "rr"/>
                 <child name = "lfu"/>
</or_relation>
<mapping>
<![CDATA[
(this.address, lru.address)(this.slot, lru.slot)(this.setIndex, lru.setIndex)
(this.address, fifo.address)(this.slot, fifo.slot)(this.setIndex, fifo.setIndex)
(this.address, lfu.address)(this.slot, lfu.slot)(this.setIndex, lfu.setIndex)
]]>
</mapping>
</operation>
```

Fig. 11 XML specification of replacement node.



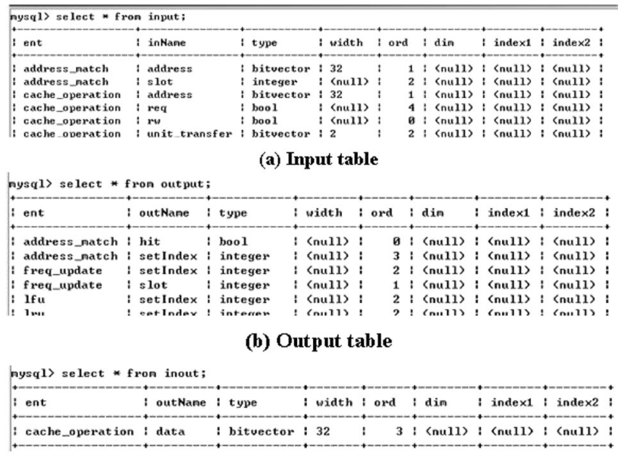
```
mysql> select * from decomp;
+------------------+----------+------------------+--------+--------+
| arch             | relation | ent              | type   | prefix |
+------------------+----------+------------------+--------+--------+
| cache_operation  | and      | read_opn         | <null> | <null> |
| cache_operation  | and      | write_opn        | <null> | <null> |
| read_opn         | and      | address_match    | <null> | <null> |
| read_opn         | and      | data_fetch       | <null> | <null> |
```

(a) and-relation

```
mysql> select * from spec;
+------------------+----------+------------------+--------+--------+
| ent              | relation | arch             | type   | prefix |
+------------------+----------+------------------+--------+--------+
| update_candidate | or       | freq_update      | <null> | <null> |
| update_candidate | or       | hist_update      | <null> | <null> |
| replacement      | or       | lfu              | <null> | <null> |
```

(b) or-relation

Fig. 12 And-relation and or-relation in database table.



```
mysql> select * from input;
+-----------------+---------------+-----------+--------+-----+--------+--------+--------+
| ent             | inName        | type      | width  | ord | dim    | index1 | index2 |
+-----------------+---------------+-----------+--------+-----+--------+--------+--------+
| address_match   | address       | bitvector | 32     |   1 | <null> | <null> | <null> |
| address_match   | slot          | integer   | <null> |   2 | <null> | <null> | <null> |
| cache_operation | address       | bitvector | 32     |   1 | <null> | <null> | <null> |
| cache_operation | req           | bool      | <null> |   4 | <null> | <null> | <null> |
| cache_operation | rw            | bool      | <null> |   0 | <null> | <null> | <null> |
| cache_operation | unit_transfer | bitvector | 2      |   2 | <null> | <null> | <null> |
```

(a) Input table

```
mysql> select * from output;
+---------------+----------+---------+--------+-----+--------+--------+--------+
| ent           | outName  | type    | width  | ord | dim    | index1 | index2 |
+---------------+----------+---------+--------+-----+--------+--------+--------+
| address_match | hit      | bool    | <null> |   0 | <null> | <null> | <null> |
| address_match | setIndex | integer | <null> |   3 | <null> | <null> | <null> |
| freq_update   | setIndex | integer | <null> |   2 | <null> | <null> | <null> |
| freq_update   | slot     | integer | <null> |   1 | <null> | <null> | <null> |
| lfu           | setIndex | integer | <null> |   2 | <null> | <null> | <null> |
| lru           | setIndex | integer | <null> |   2 | <null> | <null> | <null> |
```

(b) Output table

```
mysql> select * from inout;
+-----------------+---------+-----------+-------+-----+--------+--------+--------+
| ent             | outName | type      | width | ord | dim    | index1 | index2 |
+-----------------+---------+-----------+-------+-----+--------+--------+--------+
| cache_operation | data    | bitvector | 32    |   3 | <null> | <null> | <null> |
+-----------------+---------+-----------+-------+-----+--------+--------+--------+
```

(c) Inout table

Fig. 13 Interface tables.

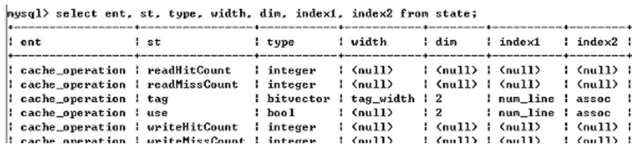
interface declaration will determines the formal argument of the function. Therefore, a designer should be careful in declaring interface.

6.2 Design Entities in Database

The design entities described in XML are stored to database. There are 15 tables in the database each of which stores and-relation, or-relation, input, output, inout, etc. In this section, we will show how some tables hold the information of design entities.

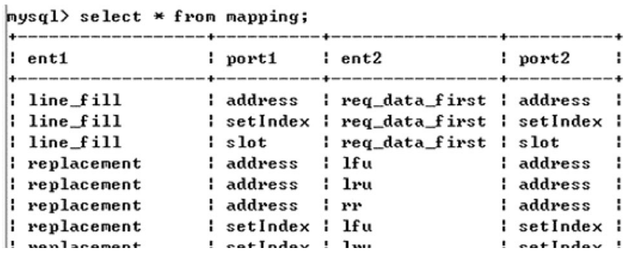
Figure 12 shows the MySQL snapshot of and-relation and or-relation. Table named decomp implies the and-relation. There are 5 columns with decomp table. The first one is the name of parent node, the second one the relation. The third column is the child name. We can know that there are two children, read_opn and write_opn, for cache_operation node. The last two columns are used when importing other design space. Type specifies what design space will be imported, and the prefix is specified to rename all the nodes attached below. or-relation is specified in the similar way.

Figure 13 shows the input, output and inout relation. The first row implies that address_match entity has input



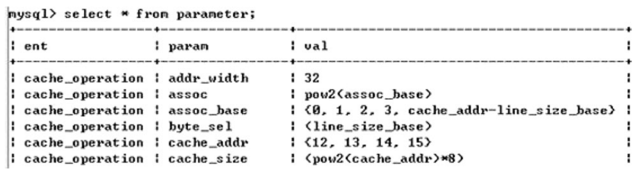
```
mysql> select ent, st, type, width, dim, index1, index2 from state;
+-----------------+----------------+-----------+-----------+--------+----------+--------+
| ent             | st             | type      | width     | dim    | index1   | index2 |
+-----------------+----------------+-----------+-----------+--------+----------+--------+
| cache_operation | readHitCount   | integer   | <null>    | <null> | <null>   | <null> |
| cache_operation | readMissCount  | integer   | <null>    | <null> | <null>   | <null> |
| cache_operation | tag            | bitvector | tag_width | 2      | num_line | assoc  |
| cache_operation | use            | bool      | <null>    | 2      | num_line | assoc  |
| cache_operation | writeHitCount  | integer   | <null>    | <null> | <null>   | <null> |
| cache_operation | writeMissCount | integer   | <null>    | <null> | <null>   | <null> |
```

Fig. 14 State table.



```
mysql> select * from mapping;
+-----------------+----------+----------------+----------+
| ent1            | port1    | ent2           | port2    |
+-----------------+----------+----------------+----------+
| line_fill       | address  | req_data_first | address  |
| line_fill       | setIndex | req_data_first | setIndex |
| line_fill       | slot     | req_data_first | slot     |
| replacement     | address  | lfu            | address  |
| replacement     | address  | lru            | address  |
| replacement     | address  | rr             | address  |
| replacement     | setIndex | lfu            | setIndex |
| replacement     | setIndex | lru            | setIndex |
```

Fig. 15 Mapping table for OR-relation.



```
mysql> select * from parameter;
+-----------------+---------------+-----------------------------------------+
| ent             | param         | val                                     |
+-----------------+---------------+-----------------------------------------+
| cache_operation | addr_width    | 32                                      |
| cache_operation | assoc         | pow2(assoc_base)                        |
| cache_operation | assoc_base    | (0, 1, 2, 3, cache_addr-line_size_base) |
| cache_operation | byte_sel      | (line_size_base)                        |
| cache_operation | cache_addr    | (12, 13, 14, 15)                        |
| cache_operation | cache_size    | (pow2(cache_addr)*8)                    |
```

Fig. 16 Parameter table.

named address, typed bitvector, and the width of the vector is 32. ord column implies the order of the port at the address_match entity. For example, address is the second argument for address match. If we look at the output table, we can see that the first argument of address_match is hit. This is necessary to synthesize the complete model. Interface can be n-dimensional at the real-life digital system design. Therefore we support this with the last three columns. We support only two-dimensional vector. index1 and index2 determines the size of the vector.

Every leaf node and the parent of and-relation can have states used within the action body. Figure 14 shows the state table. The first row implies that there is a state named readHitCount, whose type is integer. This will be declared as a variable during model synthesis.

and-relation has action body and or-relation has mapping relation, as shown in Fig. 10 and Fig. 11 respectively. There are two separate tables named action and mapping. It is difficult to show the action table here because the table contains the behavioral SystemC code. Instead, Fig. 15 shows the mapping relation between parent and children of the or-relation.

Figure 16 shows the parameter table. For example, addr_width parameter is attached to cache_operation entity, and the value is 32. Parameters also constitute the design space, and we can see this with assoc_base parameter. assoc_base parameter can have 0, 1, 2, 3, and (cache_addr-line_size_base) as its value. The associativity is computed with this assoc_base parameter.

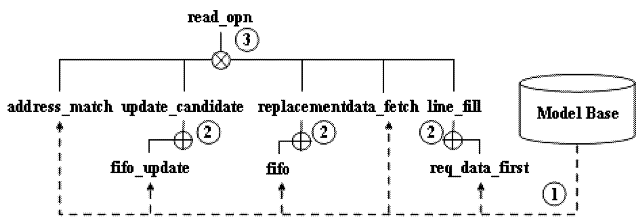
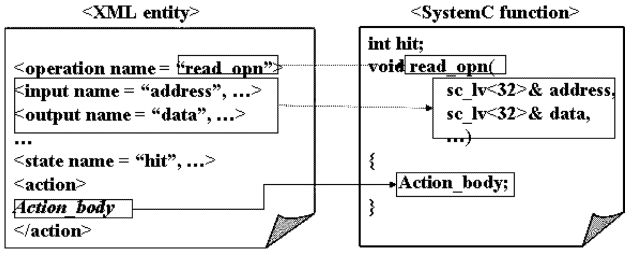


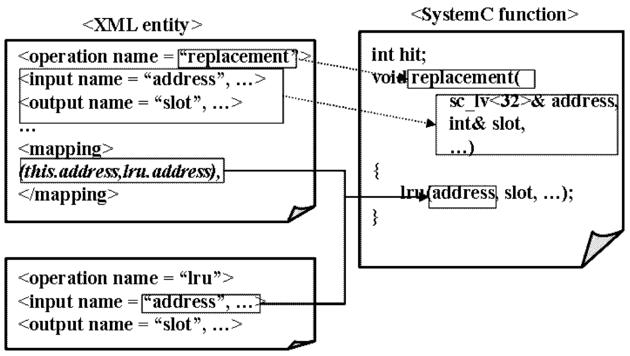
Fig. 17 Order of building model.



```
<operation name = "read_opn">
<input name = "address", ...>
<output name = "data", ...>
...
<state name = "hit", ...>
<action>
Action_body
</action>
```

```
int hit;
void read_opn(
        sc_lv<32>& address,
        sc_lv<32>& data,
        ...)
{
        Action_body;
}
```

Fig. 18 Parent of AND-relation to SystemC function.



```
<operation name = "replacement">
<input name = "address", ...>
<output name = "slot", ...>
...
<mapping>
(this.address,lru.address),
</mapping>
```

```
<operation name = "lru">
<input name = "address", ...>
<output name = "slot", ...>
```

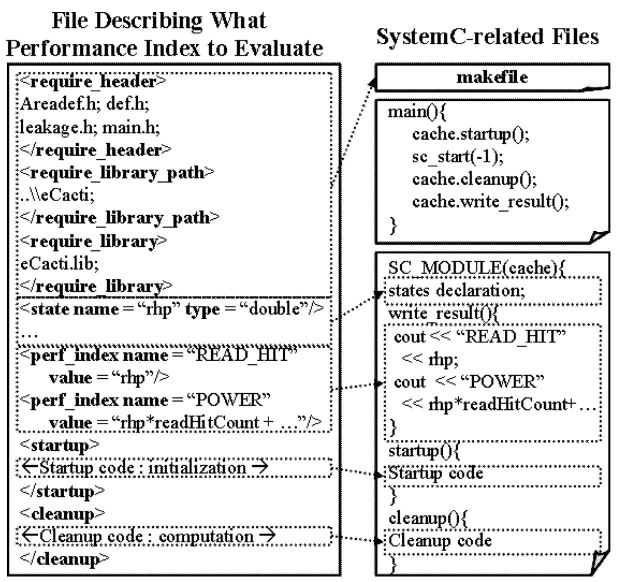
```
int hit;
void replacement(
        sc_lv<32>& address,
        int& slot,
        ...)
{
    lru(address, slot, ...);
}
```

Fig. 19 Parent of OR-relation to SystemC function.

6.3 SystemC Model Synthesis

Simulator is synthesized after pruning out unnecessary components and collecting all the candidates. In this section, we will briefly show how the simulator is synthesized from the design space stored at database. Briefly speaking, only one SystemC class that corresponds to the root node of design space is generated. All the children nodes become member functions within the class. Figure 17 shows the sequence of building model. We assume that the tree is pruned. In other words, every parent node of or-relation has only one child, and the parameters' values are fixed. Firstly, model synthesizer fetches all the required models from model base. And then, from leaf to root node, appropriate functions are synthesized considering the interface. The more detail procedure will be available with Fig. 18 and Fig. 19.

Actually, the SystemC model is synthesized from the data stored at database. However, the data stored at database is transparent because the data contained in XML document is all the same with the data stored at database. Therefore, we will describe the translation relation between XML entity and SystemC model. Figure 18 shows the translation of AND-relation to SystemC function. Operation name becomes function name and the interface declared becomes argument of the function. The order of declaring the interface is the same with the order of the argument. The action body is copied to the function body.



```
<require_header>
Areadef.h; def.h;
leakage.h; main.h;
</require_header>
<require_library_path>
..\\eCacti;
</require_library_path>
<require_library>
eCacti.lib;
</require_library>
<state name = "rhp" type = "double"/>
...
<perf_index name = "READ_HIT"
    value = "rhp"/>
<perf_index name = "POWER"
    value = "rhp*readHitCount + ..."/>
<startup>
←Startup code : initialization →
</startup>
<cleanup>
←Cleanup code : computation →
</cleanup>
```

```
main(){
    cache.startup();
    sc_start(-1);
    cache.cleanup();
    cache.write_result();
}
```

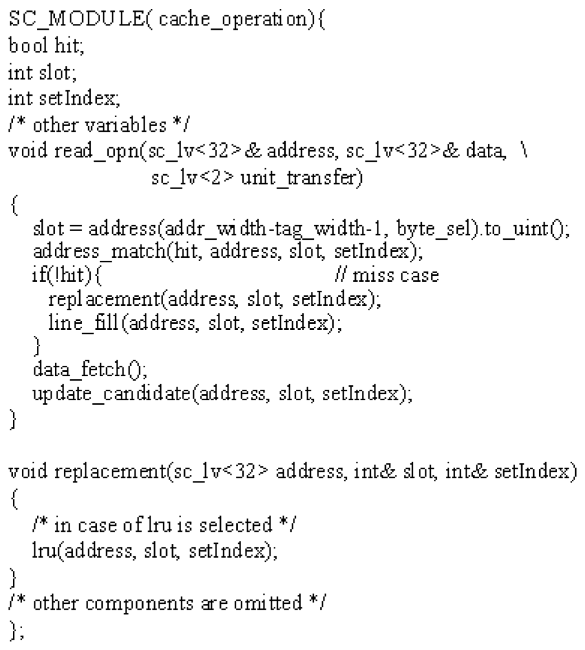
```
SC_MODULE(cache){
states declaration;
write_result(){
 cout << "READ_HIT"
  << rhp;
 cout  << "POWER"
  << rhp*readHitCount+ ...
}
startup(){
Startup code
}
cleanup(){
Cleanup code
}
```

Fig. 20 Using external library for evaluation.

Figure 19 shows how the parent of OR-relation is translated to SystemC function. In this case, the function body has only to call the proper function that is selected by the pruning algorithm. We will assume that lru algorithm is selected. Then, model synthesizer checks between the parent's interface, child's interface and the parent's mapping relation. Comparing them enables the synthesizer to call the child function appropriately.

A designer can also use the external library. For example, there is a toolkit named eCacti [19] which is designed to estimate the cache power. This toolkit is easily employed into our framework for power estimation. Figure 20 shows how we can use external library. A designer has only to write down an XML file that contains all the required information for the performance index that the designer is interested in. Header, library and all the paths are specified to be used at makefile for SystemC compilation. Three functions, startup, cleanup and write_result are synthesized and written to the model. startup and cleanup functions are copied from the corresponding section of the file. write_result function is synthesized from perf_index section of file. The correspondence is available in Fig. 20. In the main module, the function call is sequenced for the module to behave correctly. All these are shown in Fig. 20.

Figure 21 shows the actual code for Fig. 9, Fig. 10 and Fig. 11. We can see that there is only one SystemC class, cache_operation. There are variable declarations that correspond to the state component. Two functions are shown, each of which corresponds to the modules. With the gen-



```
SC_MODULE( cache_operation){
bool hit;
int slot;
int setIndex;
/* other variables */
void read_opn(sc_lv<32>& address, sc_lv<32>& data, \
                 sc_lv<2> unit_transfer)
{
   slot = address(addr_width-tag_width-1, byte_sel).to_uint();
   address_match(hit, address, slot, setIndex);
   if(!hit){                               // miss case
     replacement(address, slot, setIndex);
     line_fill(address, slot, setIndex);
   }
   data_fetch();
   update_candidate(address, slot, setIndex);
}

void replacement(sc_lv<32> address, int& slot, int& setIndex)
{
   /* in case of lru is selected */
   lru(address, slot, setIndex);
}
/* other components are omitted */
};
```

Fig. 21 Synthesized SystemC model.

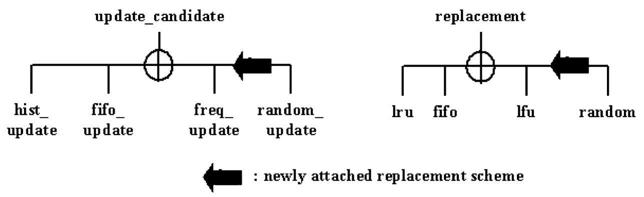
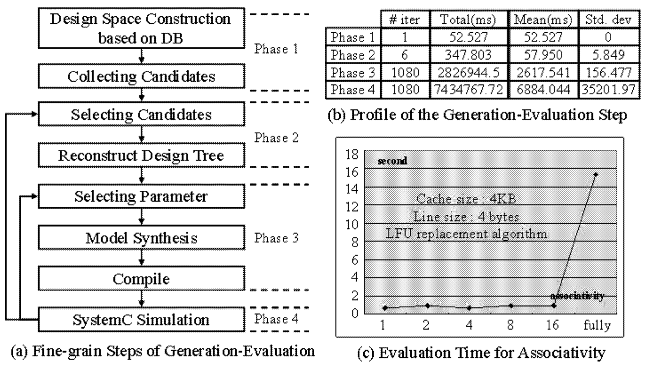


Fig. 22 Adding *random* replacement scheme in AAOG.



	# iter	Total(ms)	Mean(ms)	Std. dev
Phase 1	1	52.527	52.527	0
Phase 2	6	347.803	57.950	5.849
Phase 3	1080	2826944.5	2617.541	156.477
Phase 4	1080	7434767.72	6884.044	35201.97

Fig. 23 Profile for generation-evaluation steps.

erated SystemC model, one can perform functional simulation. The simulation result will be given in Sect. 7.2 with the evaluation of the framework itself.

7. Evaluation of Proposed Framework

7.1 Plan: Flexibility to Add New Algorithm Scheme

With the given design space, one can easily reflect his/her own algorithm by attaching it as a child node of OR-relationship. In this section, we will explain how to apply a new replacement scheme, random, to the pre-constructed design space of cache operation.

Figure 22 shows adding random replacement scheme in AAOG. There are two nodes related with replacement scheme, update_candidate and replacement. Therefore, to reflect random replacement scheme, two new nodes should be devised and attached to the right place. Figure 22 shows the process. random_update node should be attached to update_candidate node as a child of OR-relationship, and random node to the replacement node. This is done by simply modifying the children section of update_candidate and replacement specification. Writing the two nodes in XML is so simple that we could do it within five minutes. After that, we should write down an appropriate constraint in such a way that when random_update is chosen for update_candidate, random should always be chosen for replacement. Lastly, by running the program to reflect these changes into the database, planning phase completes. We need not modify any other nodes of the design space. Doing all these tasks took less than ten minutes. By just performing generation-evaluation steps, we could get the cache evaluation results.

7.2 Analysis on Generation-Evaluation Phase

We have experimented with the cache_operation design space to obtain the timing profile of our design space exploration environment. We modified ARMulator such that it emits the trace in DINERO III format whenever there is the memory reference to compare the simulation result with the DINERO simulator. We used adpcm benchmark as an application. Trace with 93645 memory references is extracted.

Figure 23(a) shows the detail steps of generation-evaluation phase. Firstly, our framework fetches all the relationship with which one can construct the design space from MySQL database. By applying constraints, we can reduce the design space to some degree. Next, all the candidates are computed using the algorithm shown above. The design candidates are based on the OR-relationship, without parametric design space. After choosing one candidate from the candidates set, we reconstruct the design tree where all the OR-relationship has only one child, i.e. pruned design graph. We apply constraints that consist of more than two entity nodes. Deciding the parameter enables us to construct a complete SystemC model. After compiling the generated SystemC model, we can evaluate by running the resultant executable.

Figure 23(b) shows the experimental result for each phase shown in Fig. 23(a). During the design space exploration, phase 1 is executed only once. Phase 2 is related with the design space of OR-relationship. The number of iteration is 6. Phase 3 and phase 4 are related with the both types of design space, resulting in the number of iteration 1080. During phase 3, compile time is the most overhead. But, the standard deviation informs us that the time for phase 3 is relatively close to constant. The standard deviation

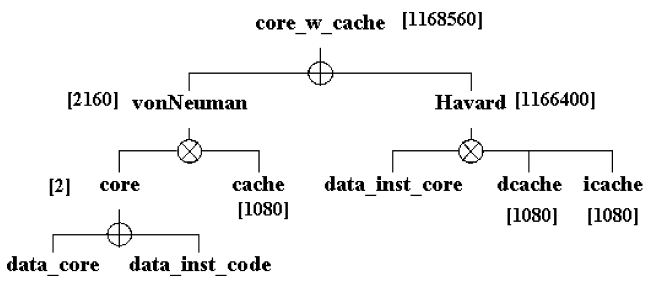


Fig. 24 AAOG with number of design.

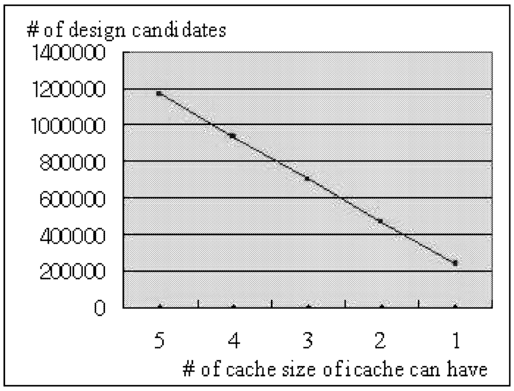


Fig. 25 Effect of *cache_size* on size of design space.

for phase 4 is very large. This is because when simulating the fully-associative cache, there happens many address matching. Figure 23(c) shows that the evaluation time is so long for the fully-associative mapping. We can formulate the time required for exploration as follows: $t_{explore} = t_{phase1} + t_{phase2} \times N_{OR} + N_{OR} \times N_{param} \times (t_{phase3} + t_{phase4})$.

7.3 Generation: Effects of Constraints

In this section, we will investigate the effect of constraints on the size of design space. Figure 24 shows the number of design space for each design entity. In Fig. 24, cache, dcache, and icache are of type cache_operation of Fig. 9. The number of design candidates of cache, dcache and icache is 1080 as explained in Sect. 6. The resulting number of design candidates of core_w_cache is 1168560, which is too many to explore completely. Let's assume that it takes about 10 seconds to evaluate one design candidate. Then more than 135 days are necessary to evaluate all the design candidates.

Appropriate usage of constraints can solve this problem. Figure 25 shows how the number of variation that cache_size of icache affects the number of design candidates. 1168560 number of design candidates has decreased to 235440 as the possible number of cache_size decreased from 5 to 1. By aggressively exploiting the designers' knowledge in the form of constraints, one can reduce the size of design space efficiently.

For example, let's assume that the following constraints are specified by considering the characteristics of applications and the hardware requirements.

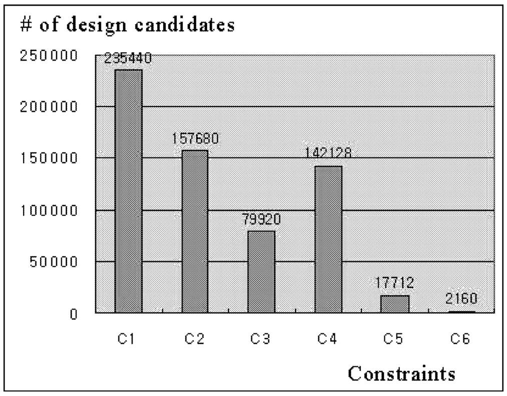


Fig. 26 Effect of constraints on size of design space.

- C1: fix the cache size of icache to 4 kbytes
- C2: fix the cache size of icache to 4/8 kbytes and line size to 4/8
- C3: fix the cache size of icache to 4/8 kbytes, line size to 4/8/16, and replacement scheme to LRU
- C4: fix the cache size of icache to 4 kbytes, cache size of dcache to 4/8/16 kbytes
- C5: fix the cache size of icache to 4 kbytes, cache size of dcache to 4/8/16 kbytes and the replacement algorithm of the both cache to LRU
- C6: use only vonNeuman

The effect of each constraint to the size of design space is shown in Fig. 26. From this, we can know that with a list of appropriate constraints, one can efficiently reduce the design space to manageable degree.

8. Conclusion and Further Work

This paper proposes a complete path of constructing a design space of a digital system by introducing plan-generation-evaluation framework.

First advantage we could obtain with this work is, of course, the easiness to construct a design space. Attributed AND-OR graph and related operations on it enable designers to construct design space easily and correctly. The next advantage is that one can use many evaluation techniques. There have been many approaches to solve the long-time evaluation problem as one of design space exploration research. The last and the most important one is that one can reflect his/her own knowledge base to prune the design space efficiently with constraints.

In addition to the above advantages about design space exploration, this framework can help in developing a new idea or modules based on pre-constructed design base as shown in Sect. 7.1.

Currently, the evaluation phase is performed by running all the executables and finding the model with optimum performance, without ranking the models. However, we can find the optimum, or at least optimal, solution without evaluating all the models if we apply evolutionary method to generation phase such as genetic algorithm[10] or simulated annealing. The design space representation of the proposed framework is based on sound graph-theoretical semantics, so a lot of algorithms on graph theory can be applied to implement such a generation method.

References

[1] S. Kobayashi, K. Mita, Y. Takeuchi, and M. Imai, "Design space exploration for DSP applications using the ASIP development system PEAS-III," IEEE International Conference on Acoustics, Speech and Signal Processing, pp.3168–3171, 2002.

[2] A. La Rosa, L. Lavagno, and C. Passerone, "Hardware/software design space exploration for a reconfigurable processor," Design, Automation and Test in Europe Conference and Exhibition, pp.570–575, 2003.

[3] M. Palesi and T. Givargis, "Multi-objective design space exploration using genetic algorithm," International Symposium on Hardware/Software Codesign, pp.67–72, 2002.

[4] L. Eeckhout, D. Stroobandt, and K. De Bosschere, "Efficient microprocessor design space exploration through statistical simulation," Simulation Symposium, pp.233–240, 2003.

[5] H. Blume, H. Hubert, H.T. Feldkamper, and T.G. Noll, "Model-based exploration of the design space for heterogeneous system on chip," IEEE International Conference on Application-specific Systems, Architectures and Processors, pp.29–40, 2002.

[6] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol.10, no.4, pp.416–422, Dec. 2002.

[7] T. Hadzic, S. Subbarayan, R.M. Jensen, H.R. Andersen, J. Moller, and H. Hulgaard, "Fast backtrack-free product configuration using a precompiled solution space representation," Proc. International Confernece on Economics, Technical and Organizational Aspects of Product Configuration Systems, pp.131–138, 2004.

[8] N.J. Nilsson, Problem Solving Methods in Artificial Intelligence, MacGraw-Hill, New York, 1971.

[9] C.L. Hang and J.R. Slagle, "An admissible and optimal algorithm for searching AND/OR graphs," Artificial Intelligence, pp.117–128, Feb. 1971.

[10] A. Mahanti and B. Bagchi, "AND/OR graph heuristic search method," J. ACM, vol.32, no.1, pp.28–51, Jan. 1985.

[11] P.S. Dasgupta, S. Sur-Kolay, and B.B. Bhattacharya, "VLSI floorplan generation and area optimization using AND-OR graph search," Proc. International Conference on VLSI Design, pp.370–375, Jan. 1995.

[12] D. Owen, B. Cukic, and T. Menzies, "An alternative to model checking: Verification by random search of AND-OR graphs representing finite-state models," International Symposium on High Assurance Systems Engineering, pp.119–126, Oct. 2002.

[13] O. Das and C. Murray Woodside, "The fault-tolerant layered queueing network model for performability of distributed systems," International Computer Performance and Dependability Symposium, pp.132–141, Sept. 1998.

[14] H. Mosemann and F.M. Wahl, "Automatic decomposition of planned assembly sequences into skill primitives," IEEE Trans. Robot. Autom., vol.17, no.5, pp.709–718, Oct. 2001.

[15] H.C. Park and T.G. Kim, "A relational algebraic framework for VHDL models management," Transactions of the Society for Computer Simulation, pp.43–55, June 1998.

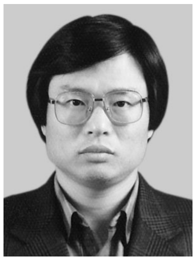
[16] B.P. Zeigler, H. Praehofer, and T.G. Kim, Theory of Modeling and Simulation, 2nd ed., Academic Press, 2000.

[17] J.K. Kim, H.Y. Kim, and T.G. Kim, "Top-down retargetable framework with token-level design for accelerating simulation speed of processor architecture," IEICE Trans. Fundamentals, vol.E86-A, no.12, pp.3089–3098, Dec. 2003.

[18] http://xml.com/pub/a/98/10/guide0.html?page=2

[19] http://www.ics.uci.edu/˜maheshmn/eCACTI/howto.htm

Jun Kyoung Kim received the B.S. degree in Electronic Engineering from Yonsei University in 1997 and M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology in 1999, respectively. From 1999 till now, he stays at Systems Modeling Simulation Laboratory (SMSLAB), in KAIST. His research interests include discrete event systems modeling/simulation, processor design, co-design and processor description language.



Tag Gon Kim received his Ph.D. in computer engineering with specialization in systems modeling/simulation from University of Arizona, Tucson, AZ, 1988. He was a Full-time Instructor at Communication Engineering Department of Bookyung National University, Pusan, Korea between 1980 and 1983, and an Assistant Professor at Electrical and Computer Engineering at University of Kansas, Lawrence, Kansas, U.S.A. from 1989 to 1991. He joined at Electrical Engineering Department of KAIST, Tajeon, Korea in Fall, 1991 as an Assistant Professor and has been a Full Professor at EECS Department since Fall, 1998. His research interests include methodological aspects of systems modeling simulation, analysis of computer/communication networks, and development of simulation environments. He has published more than 100 papers on systems modeling, simulation and analysis in international journals/conference proceedings. He is a co-author (with B.P. Zeigler and H. Praehofer) of Theory of Modeling and Simulation (2nd ed.), Academic Press, 2000. He was the Editor-in-Chief of SIMULATION: Trans of SCS published by Society for Computer Simulation International (SCS). He is a senior member of IEEE and SCS and a member of ACM and Eta Kappa Nu. Dr. Kim is a Certified Modeling and Simulation Professional by US National Training Systems Association.