New library development method by FSM based cell pattern extraction

Byung Su Kim\textsuperscript{1,2}, Tae Hee Han\textsuperscript{2}, Joon-Sung Yang\textsuperscript{2a)}, and Jaeyong Chung\textsuperscript{3b)}

\textsuperscript{1} System LSI, Samsung Electronics Co. Ltd., Giheung, Korea
\textsuperscript{2} Department of Semiconductor and Display Engineering, Sungkyunkwan University, Suwon, Korea
\textsuperscript{3} Department of Electronics Engineering, Incheon National University, Incheon, Korea
\textsuperscript{a)} js.yang@skku.edu
\textsuperscript{b)} jychung@inu.ac.kr

Abstract: The performance, power and area optimization with respect to an implementation cost is a fundamental problem in digital circuit design. With a custom design approach, finding patterns of frequent cell combinations can inspire the new cell development and optimization. A FSM (Frequent Subgraph Mining) method can help to develop new cell libraries, however, this requires huge engineering efforts for finding feasible operation conditions. This paper presents an optimized frequent subgraph mining platform by integrating various FSM methods. The experiment results with various designs demonstrate that the proposed method can reduce an overall runtime.

Keywords: standard cell library, FSM, graph mining

Classification: Integrated circuits

References

1 Introduction

A standard cell based circuit design method is mainly used for giga-scale system-on-a-chip (SoC) and application specific integrated circuit (ASIC) designs. Therefore, standard cells are mainly related with a chip design quality, such as power, performance and area [1]. Design automation tools such as logic synthesis and physical implementation tools try to optimize a design, however, the design performance and tool performance are fundamentally restricted by standard cell libraries. This means that the optimization of transistors, cells and cell combinations used in design is critical to enhance the performance.

If frequently used cell combinations are known in advance, a new cell development with the frequent cell combinations can help to solve the problems. To find the frequent cell combinations, it is required to figure out the number of subcircuits’ occurrence in an entire design. However, it is a well-known NP-hard problem. Recently, we present AFSEM [2] that is a frequent subgraph mining (FSM) based standard cell pattern extraction method. This method uses a gSpan [3, 4, 5] based FSM algorithm which represents a growing pattern approach starting with a single vertex. Therefore, this algorithm finds various cell combination patterns within a feasible runtime. However, it requires a high memory usage since it considers all candidate patterns and can cause memory problems. High engineering efforts need to be needed to apply this method adequately. Hence, several constraints such as subgraph size and the number of occurrence have to be limited to secure the appropriate memory usages and runtime. This paper presents a new practical approach to reduce engineering efforts by combining several FSM algorithms based on AFSEM.

Fig. 1. Overall flow of design flow with FSM based new library generation

2 Cell combinations pattern aware design method

Fig. 1 shows an overall design flow with the FSM based cell combination pattern extraction method. Firstly, an RTL source is synthesized to a cell netlist which is mapped by cell libraries. The cell libraries are standard technology libraries used by synthesis tools such as Synopsys design compiler. The technology library is generally provided from a manufacturing company.

Next, if a cell netlist fails to meet design specifications after logic synthesis, the proposed method tries to develop a new cell library with frequent cell combination patterns. This process iteratively continues after placement and global routing considering a physical implementation. This new flow gives another chance to optimize the design by custom design styles.

Frequent cell combination patterns found by the proposed method is usually composed of ~10 cells. Some of patterns could include a flip-flop and latch. This is a very interesting property to develop new cell libraries and to carry out a physical implementation. In addition, the frequent cell combination can be integrated with previous works [6, 7] that make new custom cells from critical paths to enhance circuit performances.

3 Frequent pattern extraction of standard cells combination by FSM algorithms

One of the FSM algorithms can find subgraphs from a single large connected graph. This problem is defined as follows. Given a graph $G$ and a minimum support threshold $\gamma$, the frequent subgraph mining problem is defined as finding all subgraphs $S$ in $G$ such that have occurrences over $\gamma$.

This method is very useful in finding several types of subcircuits. To solve the FSM problem, it needs to choose a graph translation and FSM algorithm. To translate a cell netlist to a graph, the following shows the graph definition, $G = (V, E, L)$ where $V$, $E$, $L$ denote vertices, edges and labels.

$v_i$: $i^{th}$ cell, $v_j$: $j^{th}$ cell
$L(v_i) =$ vertex label using cell type name of $i^{th}$ cell
$V(v_i) =$ {index of $v_i$, $L(v_i)$}
$e(v_i, v_j) =$ pin to pin wire connection.
$L(e(v_i, v_j)) =$ label of pin to pin wire connection.

The cells are represented by vertices in $V(v_i)$ function. The vertex label uses a cell type name by function $L(v_i)$. Wires in the netlist may have multiple inputs and multiple outputs. However, the graph edge in the labeled graph has only single input and single output. We use a definition as $e(v_i, v_j, \text{input})$ to show a direct edge indicating an input to output direction. To describe a direction of the edge, the first vertex is a start point and the next vertex is an end point of the edge.

Secondly, FSM algorithms to find frequent subgraphs in a single large connected graph are Apriori-based approaches [8], gSpan based approaches [3, 4] and SUBDUE [9]. The Apriori-based approaches would not be efficient methods for a single large connected graph since it requires a huge runtime to check isomorphism.
among candidate sets. The gSpan based approaches are pattern growth methods that use lexical order and rightmost expansion policies [3] to avoid isomorphism check. Our preliminary work, AFSEM [2], is based on gSpan. It can extract various interesting subcircuits in a reasonable time from a design which has more than 100 million instances. However, it needs a significant memory usage. To reduce memory usages, AFSEM supports user constraints to shorten candidate sets.

Algorithm 1: SUBDUE($G, K, limit$) // $G$ is a graph, the $K$ is a number of subgraphs, and a $limit$ on the depth of the search.

1: $Q$ ← all single vertex subgraphs in $G$
2: $S$ ← NULL
3: while limit > N and $Q$ ≠ NULL
4: for each $p$ in $Q$
5: $S$ ← $S$ $∪$ InsertNewChild($p$)
6: end
7: $Q$ ← the best $K$ number of subgraphs in $S$
8: $limit$ ← $limit$ - 1
9: end
10: return $Q$

The SUBDUE algorithm only extracts $K$-subgraphs. It is different from the previous FSM algorithms. Algorithm 1 presents the SUBDUE algorithm. This search algorithm is a best-first version of breadth-first search, hence, children are expanded at each level of the search as shown in line 5. Then, the rest without the best $K$ number of subgraphs are removed at line 7. Due to this property, this algorithm becomes an efficient yet sometimes inexact method. $limit$ is used that the average logic depths between FF and FF of each circuits. The parameter, $K$, is empirically determined. As $K$ increases, the run time to extract sub-graphs increases. Hence, it is preferred to determine the parameter $K$ not too small or not too large (to avoid extraction of undesired sub-circuits and extremely long runtime).

For the proposed method, it is preferred to find sufficiently large sub-circuits that are often used. AFSEM and SUBDUE have complementary characteristics. If AFSEM is used, it would require a significant runtime with various numbers of supports. However, SUBDUE quickly finds $K$ sub-circuits which are big enough. We assume the size of $K$ sub-circuits found by SUBDUE as the largest sub-circuit size. Based on this, the frequency of sub-circuits found by SUBDUE is used as supports in AFSEM to reduce a runtime overhead. Then, AFSEM extracts all frequent sub-circuits. Therefore, we can reduce overall runtime by combining them.

4 Experiment results

In this section, we evaluate the proposed method with ISCAS85 benchmark circuits [10]. The proposed method is implemented in C++/Java JDK 8.0. Experiments were run on 2.5 GHz Intel i7-4710HQ CPU with 16 GB memory.

Table I shows comparison results of AFSEM and SUBDUE using various ISCAS circuits. SUBDUE generates only three frequent subcircuits because it
performs K-best search ($K = 3$). The runtime of SUBDUE is smaller than AFSEM and the memory usage of SUBDUE is only $\sim 1\%$ of AFSEM. Then, AFSEM carries all subcircuit extraction under the conditions. The proposed SUBDUE and AFSEM combination, AFSEM-SUBDUE, can effectively reduce engineering efforts.

To analyze an AFSEM-SUBDUE performance, we choose s38417 and s38584 among ISCAS85 benchmark circuits because they have a number of cells 23,816 and 20,706 cells, respectively. Fig. 2 presents the number of patterns, memory usage, runtime and runtime per each pattern. The $X$ axis in Fig. 2 shows a normalized number of supports that is calculated as “a number of occurrences/a number of all instances in circuit”. The $Y$ axis illustrated a normalized number of (A) pattern, (B) memory usage, (C) runtime and (D) runtime per each pattern. Initially, we run AFSEM at various supports (a number of pattern’s occurrences) to find cost effective condition for frequent cell combination extraction. Unlike AFSEM, AFSEM-SUBDUE is performed only once since it helps to find the feasible condition by SUBDUE. The red line of each graph is AFSEM results of s38417 using various supports. The blue line is AFSEM results of s38584. The red circle is AFSEM results at the optimized operation condition extracted from AFSEM-SUBDUE.

As shown in Fig. 2(A), AFSEM generate a large number of patterns using small supports. Some cell combination is treated as a frequent cell combination even if it is less frequent. As in Fig. 2(B) and (C), the memory usage and runtime are increased very quickly as supports decrease. This is obvious because the FSM algorithm by AFSEM tries to find a complete set of frequent subgraphs within a number of supports. Finally, we define a cost function that is a runtime per each pattern (total runtime/a number of patterns). It is very useful to decide an operation condition of AFSEM. Fig. 2(D) shows the cost value of AFSEM at the optimized operation condition extracted from AFSEM-SUBDUE. AFSEM-SUBDUE operates at efficient operating conditions in two design cases. This low cost feature ensures a reasonable performance at the initial stage of chip design. From this point of view, AFSEM-SUBDUE are a cost effective solution than multiple iterations of AFSEM to find feasible operation conditions. In actual circuit optimization, we could make more optimized cells by reducing the pattern extraction process, which took more than a week, to one day.

### Table I. Comparison results of AFSEM and SUBDUE

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th># of Cell</th>
<th># of Sub circuit</th>
<th>AFSEM Runtime (s)</th>
<th>Memory (Byte)</th>
<th>SUBDUE Runtime (s)</th>
<th>Memory (Byte)</th>
<th>Difference</th>
<th>Runtime</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>S13207</td>
<td>8621</td>
<td>70</td>
<td>21.65</td>
<td>2365404</td>
<td>3</td>
<td>3.33</td>
<td>7316</td>
<td>15.38%</td>
<td>0.31%</td>
</tr>
<tr>
<td>S15850</td>
<td>10370</td>
<td>95</td>
<td>31.33</td>
<td>2395960</td>
<td>3</td>
<td>4.49</td>
<td>8824</td>
<td>14.33%</td>
<td>0.37%</td>
</tr>
<tr>
<td>S35932</td>
<td>17794</td>
<td>643</td>
<td>655.6</td>
<td>9251600</td>
<td>3</td>
<td>12.42</td>
<td>14780</td>
<td>1.89%</td>
<td>0.16%</td>
</tr>
<tr>
<td>S38417</td>
<td>23816</td>
<td>880</td>
<td>1022.68</td>
<td>10403576</td>
<td>3</td>
<td>9.82</td>
<td>20452</td>
<td>0.96%</td>
<td>0.20%</td>
</tr>
<tr>
<td>S38584</td>
<td>20706</td>
<td>1173</td>
<td>2143.2</td>
<td>23632001</td>
<td>3</td>
<td>10.07</td>
<td>18668</td>
<td>0.47%</td>
<td>0.08%</td>
</tr>
</tbody>
</table>

© IEICE 2018
DOI: 10.1587/elex.15.20171226
Received December 12, 2017
Accepted January 9, 2018
Publicized January 24, 2018
Copyedited May 10, 2018
5 Conclusion

For optimizing performance, power, and area of a circuit, an effective extracting method of frequent cell combinations is needed. Therefore, we present an optimized frequent subgraph mining platform by combination of various FSM methods. This new method can reduce an overall runtime without iterations to find best operation conditions.

Acknowledgments

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea by the Ministry of Education under Grant NRF-2015R1D1A1A01058856, in part by the Korea Institute for Advancement of Technology (KIAT) by the Korean Government (Motie: Ministry of Trade, Industry & Energy, HRD Program for Software-SoC Convergence) under Grant N0001883, in part by the MOTIE (Ministry of Trade, Industry & Energy) and KSRC (Korea Semiconductor Research Consortium) support program for the development of the future semiconductor device.