第16回　回路とシステム
（軽井沢）ワークショップ
The 16th Workshop on Circuits and Systems
in Karuizawa, April 27, 28, 2003

Network-Flow Based Delay-Aware Circuit Partitioning Algorithm

Masato INAGI　Atsushi TAKAHASHI
Department of Communications and Integrated Systems,
Tokyo Institute of Technology

Abstract

We propose a delay-aware circuit partitioning algorithm under I/O pins and size constraints. These two constraints are essential in multi-device implementation. The partitioning in multi-device implementation affects the delay much, since the propagation delay of inter-device connection is considerably larger than that of intra-device connection. Many approaches without considering delay fail to obtain a reasonable solution. Our partitioning algorithm is an enhancement of a partitioning algorithm based on flow network without considering delay, called PART. The idea of our enhancement is the reflection of timing slack into the flow network in order to avoid cutting tighter slack net. Our algorithm is implemented and applied to benchmark circuits. In experiments, we observed that the maximum propagation delay between registers is shorter and the number of subcircuits is smaller compared with PART.

1 Introduction

As the size of a circuit to be implemented becomes larger, a circuit partitioning is required in the circuit layout or in implementing the circuit into MCM or FPGAs. In this paper, we discuss the partition problem to implement the circuit into multi-devices, e.g. multi-FPGAs. If the size of circuit is too large to implement in a single device, then the circuit must be partitioned into subcircuits to fit devices. In such cases, the size and I/Os of each subcircuit must not exceed the capacity of the device. The number of subcircuits is an objective of partitioning to reduce the cost of implementation. The circuit speed in multi-device implementation is usually much slower than that in single-device implementation since the propagation delay of the inter-device connection is larger than that of intra-device connection. In order to keep the circuit speed as fast as possible in multi-device implementation, the delay should be taken into account in partitioning.

The partitioning algorithms proposed in [6, 8] are based on a min-cut algorithm of a flow-network. In [6, 8], a circuit is transformed into a flow-network, and the subcircuits satisfying the size and I/O constraints are extracted repeatedly according to a min-cut of the flow-network. In the transformation to flow-network, each net is transformed into a local structure whose flow capacity is one. The flow capacity of each local structure corresponds to the I/O which is required when the corresponding net becomes inter-connection. Therefore, the size of min-cut which corresponds to the max-flow of the flow-network is equal to the number of I/Os of the extracted subcircuit. In each extraction, these algorithms enumerate a number of subcircuits satisfying the size and I/O constraints, and then select the largest one. However, these algorithm does not take the circuit speed into account.

In this paper, we propose a delay-aware partitioning algorithm which is an enhancement of the algorithm *PART* proposed in [8]. In our algorithm, a circuit is transformed into a flow-network as in *PART*. The principal diffence between our algorithm and *PART* is the definition of flow capacity of a local structure. In our transformation of a net into a local structure, the flow capacity is determined according to the delay-slack of the net. The local structure corresponding to a net with small delay-slack has the large flow capacity. Then the net with small delay-slack will not be included in a min-cut.

The rest of the paper is organized as follows. We define the graph model of a circuit in Section 2. We propose our algorithm in Section 3, and show experimental results in Section 4. Finally, the conclusions are described in Section 5.

2 Preliminaries

2.1 Circuit Model

We consider synchronous circuits whose flip-flops are triggered by a clock simultaneously. A circuit C is modeled as a hypergraph $G(V, E)$ which is called a circuit graph. $V = V_{gate} \cup V_{FFin} \cup V_{FFout} \cup V_{PI} \cup V_{PO}$ is the set of vertices of G where V_{gate}, V_{FFin},

V_{FFout}, V_{PI}, V_{PO} correspond to the gates, the input of flip-flops, the output of flip-flops, the primary input and primary output of C, respectively. $E = E_{sig} \cup E_{io}$ is the set of hyperedges of G where E_{sig} and E_{io} correspond to the internal signal nets and I/O nets, respectively. That is, $E_{io} = \{e \in E |\ V(e) \cap (V_{PI} \cup V_{PO}) \neq \emptyset\}$ where $V(e)$ denoted the set of vertices incident to e.

A partition of a circuit graph $G(V, E)$ is a decomposition of V into subsets $V_1, V_2, \ldots, V_m$, and $V_{PI} \cup V_{PO}$ such that $\bigcup_{i=1}^{m} V_i = V_{gate} \cup V_{FFin} \cup V_{FFout}$, $V_a \cap V_b = \emptyset$ $(1 \leq a, b \leq m$ and $a \neq b)$, and two vertices corresponding to the input and output of a flip-flop are in V_i $(1 \leq i \leq m)$.

The size size(V_i) of V_i is defined as $\sum_{v \in V_i}$ size(v) where size(v) is the size of circuit element corresponding to v in V. The number io(V_i) is the number of I/Os of subcircuit corresponding to V_i which is defined as

$$\mathrm{io}(V_i) = |\{e \in E |\ (V(e) \cap V_i \neq \emptyset) \wedge (V(e) \cap V \setminus V_i \neq \emptyset)\}|.$$

Let $d(v)$ be the delay of a circuit element v and $d(v', v)$ be the delay of a signal delay from v' to v. $d(v', v)$ increases if v' and v belong to different subcircuits.

Let $V_{fi}(v)(V_{fo}(v))$ be the set of fan-in (fan-out) vertices of v. The ASAP(v) is defined as

$$\mathrm{ASAP}(v) = \begin{cases} \max\limits_{v' \in V_{fi}(v)} (\mathrm{ASAP}(v') + d(v', v) + d(v)) & (v \notin V_{in}) \\ 0 & (v \in V_{in}) \end{cases}$$

where $V_{in} = V_{PI} \cup V_{FFout}$.

The minimum clock period of the completely-synchronous circuit (the delay of the combinational circuit) is $\max_{v \in V_{fo}}(\mathrm{ASAP}(v))$.

The problem is to find a better decomposition of V such that size$(V_a) \leq lim_{size}$ and io$(V_a) \leq lim_{io}$ where lim_{size} and lim_{io} are the maximum size and I/Os of subcircuit which can be implemented into a device. The first objective of the problem is the number of subcircuits and the second objective function is the delay of the resultant circuit.

2.2 Hyper Flow Transformation

Hyper Flow Transformation which transforms a circuit graph $G(V, E)$ to the flow graph $G_f(V_f, E_f)$ is used in [4, 8] and our algorithm.

Hereinafter, the flow edge which has direction from v_a to v_b is denoted as (v_a, v_b), and the flow edge (v_a, v_b) with capacity c is denoted as (v_a, v_b):c. Let cap(e) be the capacity of a flow edge e. For the set $V_s \subset V_f$ of vertices, the set of edges from V_s to $V_f \setminus V_s$ consists of a cut between V_s and $V_f \setminus V_s$. Let cap(V_s) be the sum of capacity of edges from V_s to $V_f \setminus V_s$. The set of edges from V_s to $V_f \setminus V_s$ consists of a s-t cut if the source vertex $v_s \in V_s$ and the sink vertex $v_t \notin V_s$.

Hyper Flow Transformation transforms every internal signal net $e \in E_{sig}$ into a local structure of the flow graph by Yang-Wong transformation(Y-W Trans.[4, 7]).

An example of transforming a net with 3 terminals is shown in Fig.1. As shown in Fig.1, in the local structure of flow graph for e, two vertices connected by an edge with finite capacity are added. The capacity of the other edges is ∞.

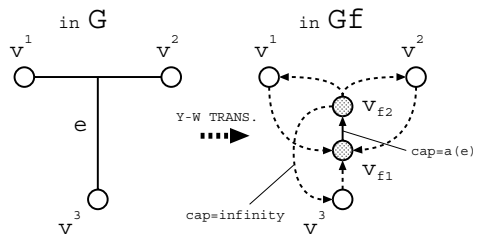

Figure 1: Yang-Wong Transformation

Let V_s be the set of vertices of the flow graph that contatins v_s but not v_t. V_s is said to be legal in terms of the flow graph if cap(V_s) is finite and if a dummy node of the local structure for e is contained in V_s $(V_f \setminus V_s)$ if and only if $V(e) \cap V_s \neq \emptyset$ $(V(e) \cap (V_f \setminus V_s) \neq \emptyset)$. A legal V_s is said to be feasible if size$(V_s) \leq lim_{size}$ and cap$(V_s) \leq lim_{io}$.

Let V' be any non-empty proper subset of $V(e)$, max-flow value from V' to $V(e) \setminus V'$ is equal to the capacity of the edge connecting two vertices. Thus we define the capacity of the edge with finite capacity in the local structure as the capacity of the local structure.

Note that cap(V_s) is equal to io(V_s) if V_s is legal and cap$(e) = 1$ for each edge e connecting two dummy vertices.

The details of Hyper Flow transformation is described as follows.

Hyper-Flow Transformation[7]

input: a circuit graph $G(V, E)$

output: a flow graph $G_f(V_f, E_f)$

1. $V_f = \{v_t\} \cup \{v_s\} \cup V_f^v \cup V_f^d$, where
   (a) v_s is the source vertex and v_t is the terminal vertex of G_f

(b) $V_f^v = V$

(c) $V_f^d = \bigcup_{e \in E_{sig}} \{v_{f1}(e), v_{f2}(e)\}$

2. $E_f = E_f^e \cup E_f^{io} \cup E_f^{FF} \cup E_f^t \cup E_f^s$, where

(a) E_f^e is a set of edges representing internal nets
$\{(v_{f1}(e), v_{f2}(e)):a(e) | e \in E_{sig}\}$
$\cup \{(v, v_{f1}(e)):\infty | e \in E_{sig} \wedge v \in V(e)\}$
$\cup \{(v_{f2}(e), v):\infty | e \in E_{sig} \wedge v \in V(e)\}$
where $a(e)$ is a constant.
(Y-W Trans.)

(b) E_f^{io} is a set of edges representing I/O nets ($E_f^{io} = \{(v_a, v_{io}(e)):\infty | e \in E_{io} \wedge v_a \in V(e) \setminus \{v_{io}(e)\}\}$)

(c) E_f^{FF} is a set of edges to prevent from dividing the circuit between FF's input and output ($E_f^{FF} = \{(v_i, v_o):\infty, (v_o, v_i):\infty | v_i (\in V_{FFin})$ and $v_o (\in V_{FFout})$ belong to the same flip-flop $\}$)

(d) E_f^t is a set of edges that connect I/O and sink v_t ($E_f^t = \{(v_f, t):1 | v_f \in V_{PI} \cup V_{PO}\}$)

(e) $E_f^s = \phi$

Note that, in this transformation, E_f^s is $\emptyset$. The flow-network for circuit extraction is completed by adding infnite capacity edges from v_s in the partitioning algorithm. Note that the size of mincut depends on how edges from v_s are added.

An edge contained in a minimum s-t cut of G_f corresponds to either the finite capacity edge of a local structure or an edge from I/O to sink v_t. If the circuit corresponding to the source part of a minimum s-t cut of G_f is extracted as a subcircuit, then the subcircuit should have I/O corresponding to each edge in the minimum s-t cut. When the capacity of each local structure is 1, the number of I/Os of subcircuit is equals to the number of edges in the minimum s-t cut and the maximum flow of G_f. In this case, by computing maximum flow after adding edges from v_s, we can determin whether a subcircuit that satisfies I/O constraints can be extracted or not.

An example of Hyper Flow Transformation is shown in Fig.2.

2.3 Min-Cut Partitioning Algorithm

In this section, we explain the outline of the network-flow based partitioning algorithm *PART*[8], on which our proposing algorithm is based. *PART* divides a circuit into subcircuits under the size constraint and the I/Os constraint. The objective of *PART* is to minimize the number of subcircuits. *PART* iteratively extracts the subcircuits as large as possible, not to violate the size and the I/O constraints.

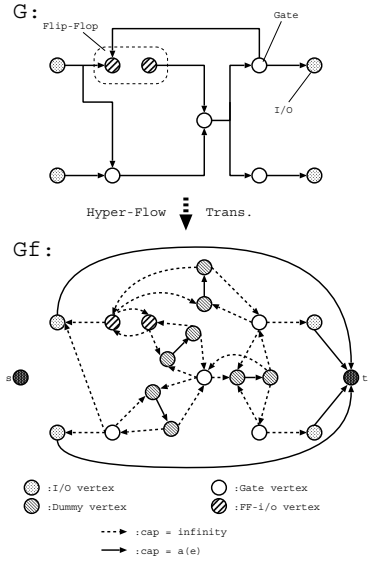

Figure 2: Hyper Flow Transformation

Algorithm *PART*[8]

Input a circuit graph $G(V, E)$

Output a partition of G ,$\{V_1, V_2, \cdots, V_m\}$

1. $m \leftarrow 1$ (m represents the number of subcircuits)
2. If V is feasible then $V_m \leftarrow V$, output $\{V_1, V_2, \ldots, V_m\}$ and stop
3. $j \leftarrow 0$. Transform $G(V, E)$ into a flow graph $G_f(V_f, E_f)$ by Hyper-Flow Transformation where $a(e \in E_{sig}) = 1$, and $G_f^0 = G_f$.
4. Let G_f^{j+1} be the flow graph obtained from G_f^j by adding edge $(v_s, v'_{j+1}):\infty$ where v'_{j+1} is a vertex adjacent to an I/O vertex ($V_{PI} \cup V_{PO}$) but not v_s in G_f^j.
5. If there is no feasible vertex set whose capacity is equal to the max-flow value of G_f^{j+1}, then goto step 8.
6. $j \leftarrow j + 1$. Let V'_j be a feasible vertex set whose size is maximum among legal vertex sets in terms of G_f^j whose capacity is equal to the max-flow value of G_f^j.
7. Get V''_j by expanding V'_j as large as possible by expansion procedure EXPAND (see [8] for detail).

8. If $j = 0$ then output "infeasible" and stop.

9. Let V_m be a feasible vertex set whose size is maximum among $V_1'', V_2'', \ldots, V_j''$.

10. $V \leftarrow V \setminus V_m, m \leftarrow m + 1$ and goto step 2.

In steps 6 and 9, ties are broken by the minumum io(V') followed by the maximum primIo(V') where primIo(V) is the number of original I/Os of the subcircuit that corresponds to V.

We explain the part of the process which extracts a subcircuit from the remaining circuit V in the following.

PART constructs a flow graph $G_f^0 = G_f(V_f, E_f)$ by transforming $G(V, E)$. And a number of flow graphs $G_f^1(V_f^1, E_f^1), G_f^2(V_f^2, E_f^2), \ldots, G_f^j(V_f^j, E_f^j)(E_f^{i+1} \leftarrow E_f^i + \{(v_s, v'_{i+1}):\infty\})$ are constructed where v'_{i+1} is a vertex adjacent to an I/O vertex ($V_{PI} \cup V_{PO}$) but not v_s in G_f^i, and j is the maximum number of i such that the max-flow of G_f^i is less than lim_{io}. v'_1 is selected from vertices with the maximum degree which are adjacent to an I/O vertex ($V_{PI} \cup V_{PO}$), and $v'_i (i = 2, \ldots, j)$ is a vertex with minimum distance measured by the lenth of undirected shortest pass between v_s and v'_i in G_f^{i-1} If the max-flow of G_f^1 is larger than lim_{io}, the algorithm fail a partitioning the circuit and stop. For $G_f^1, G_f^2, \ldots, G_f^j$, an candidate of subcircuit $V_1'', V_2'', \ldots V_j''$ is obtained respectively, and an best V_i'' is applied as an adopted subcircuit V_m. To obtain V_i'' which has small number of I/Os, *PART* search mincuts of G_f^i and the best subcircuit V_i' extracted by an mincut is selected. And V_i' is expanded as V_i'', as large as possible by expansion procedure *EXPAND*[**?**] based on n-th mincut. Note that $v'_1, v'_2, \ldots, v'_i$ are always included in V_i'' because the addition of an edge $(v_s, v'_i):\infty$ fix v'_i to source part of mincuts.

3 Delay-Aware Partitioning Algorithm

The objective of *PART* is to minimize the number of the subcircuits only. We propose the improved algorithm of *PART* to obtain the partitioned circuit which has shorter clock period without increase of the number of the subcircuits.

In our algorithm, we define *slack* which represents the delay margin and partition the circuit so that the net with less slack avoids crossing over different subcircuits.

Considering the fact that the larger the flow capacity of the net e is, the less likely the min-cut includes e, we reflect slacks to the flow graph.

3.1 Slack

We define the slack, that is the delay margin, of each net, as follows.

We define ALAP(v), which represents latest output time of v without increase of clock period, as

$$\mathrm{ALAP}(v) = \begin{cases} \min\limits_{v' \in V_{fo}(v)} (\mathrm{ALAP}(v') - d(v') - d(v, v')) & (v \notin V_{out}) \\ T & (v \in V_{out}) \end{cases}$$

where $V_{out} = V_{PO} \cup V_{FFin}$.

If the output of the gate $v_o (\in V_{fo}(e))$ is fixed before the time ALAP(v_o) even after partitioning, the clock period does not change. If the output of the gate $v_i (= v_{fi}(e))$ is fixed at the time ASAP(v_i), the permissible delay between v_i and v_o is at most (ALAP(v_o) $- d(v_o)$) $-$ ASAP(v_i). Considering the delay $d(v_i, v_o)$ of the internal net e, the slack between v_i and v_o is defined as

$$\begin{aligned} &\mathrm{slack}_{io}(v_i, v_o) \\ &\quad = \mathrm{ALAP}(v_o) - \mathrm{ASAP}(v_i) - d(v_o) - d(v_i, v_o). \end{aligned}$$

Taking into account the all $v_o (\in V_{fo}(e))$, permissible additional delay is at most

$$\min_{v_o \in V_{fo}(e)} (\mathrm{slack}_{io}(v_i, v_o)).$$

Therefore, the slack of net e $\mathrm{slack}_{sig}(e)$ is defined as follows.

$$\mathrm{slack}_{sig}(e) = \min_{v_o \in V_{fo}(e)} (\mathrm{slack}_{io}(v_i(e), v_o))$$

The above equation is developed as follows.

$$\begin{aligned} &\mathrm{slack}_{sig}(e) \\ &= \min_{v_o \in V_{fo}(e)} (\mathrm{ALAP}(v_o) - \mathrm{ASAP}(v_i(e)) \\ &\qquad\qquad -d(v_o) - d(v_i(e), v_o)) \\ &= \min_{v_o \in V_{fo}(e)} (\mathrm{ALAP}(v_o) - d(v_o) - d(v_i(e), v_o)) \\ &\qquad\qquad -\mathrm{ASAP}(v_i(e)) \\ &= \mathrm{ALAP}(v_i(e)) - \mathrm{ASAP}(v_i(e)) \end{aligned}$$

3.2 Delay-Aware Hyper-Flow Transformation

We explain the diffeence between our algorithm and *PART* in the following.

We note that the max-flow value of the flow graph depends on the flow capacity of the edge $(v_{fi}(e), v_{fo}(e)) \in E_f$. In *PART*, every capacity of

$(v_{fi}(e), v_{fo}(e))$ is 1. In our algorithm, hence nets with small slack may be cut. the capacity is varied by the function $weight(slack(e))$.

To assign the larger capacity to the smaller slack net, the weight function of the slack x is defined as

$$\text{weight}(x) = \begin{cases} \alpha - x + 1 & (x < \alpha) \\ 1 & (x \geq \alpha) \end{cases}$$

where α is a constant.

Yang-Wong* Transformation (Y-W* Trans.) is defined as modified Yang-Wong Transformation, which assigns weight(slack(e)) to $(v_{fi}(e), v_{fo}(e))$ for each $e(\in E_{sig})$ (in *PART*, $(v_{fi}(e), v_{fo}(e))$:1). In our algorithm, Y-W Trans. is replaced by Yang-Wong* Trans. in Hyper Flow Transformation The example is shown in Fig.3

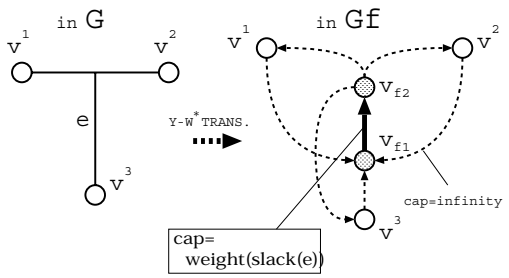

Figure 3: Yang-Wong* Transformation

According to this modification, the max-flow value is not equal to the number of I/Os of the subcircuit extracted by the min-cuts. Therefore, we count up the number of the I/Os of the subcircuit extract by the min-cut and evaluate just the subcircuit which satisfies the number of I/Os constraint.

Our algorithm generates some flow graphs by adding edges one by one as step 4 of *PART*. How to select vertices $v'_1, v'_2, \ldots$ connected with v_s by (v_s, v'_i):∞ is described as follows. $v'_1 \in V$ is a vertex with the maximum degree which are adjacent to an I/O vertex $(V_{PI} \cup V_{PO})$, and $v'_i \in V (i = 2, \ldots, j)$ is a vertex with minimum distance measured by the lenth of undirected shortest pass between v_s and v'_i in G_f^{i-1} If there are a number of candidates of v'_i, ties are broken by the minimum $\text{ALAP}(v'_i) - \text{ASAP}(v'_i)$.

4 Experimental Results

We have implemented the algorithms and applied to the ISCAS'85/'89 benchmark circuits shown in Table.1. *del* is the clock period of the circuit implemented in a chip using the PC with Athlon XP 1800+, DDR-768MB memory. In our experiments, we set constraints and constants as follows.

- size constraint $lim_{size} = 200$
- I/O constraint $lim_{io} = 40$
- delay of gate $d(v(\in V_{gate})) = 1$
- delay of I/O and FF $d(v(\in V \setminus V_{gate})) = 0$
- delay of inner net $d(v, v') = 0$
- inter subcircuits delay $d(v, v') = 5$
- size of gate and FF is 1

Table 1: Benchmark circuits

name	# gate	# net	# I/O	del
c499	202	243	73	21
c880	383	443	86	34
c1355	546	587	73	34
c1908	880	913	58	50
c3540	1669	1719	72	57
c5315	2307	2485	301	59
c6288	2416	2448	64	134
c7552	3512	3718	313	53
s510	217	236	26	14
s1196	547	561	28	34
s5378	2958	2993	84	31
s9234	5825	5844	41	59

We compared our algorithm(Our Algorithm) with the conventional method (*PART*). *ave.* shows the average of ratios of results of the algorithm for each circuit compared with *PART*. The results of the experiments demonstrate that our algorithm generated the partitioned circuit whose clock period is 80.7% compared with *PART*, and whose number of subcircuits is 98.2% compared with *PART*.

It is natural to think that the number of subcircuits partitioned by the our algorithm will be larger than the one of *PART*, since the our algorithm take into account the delay of the circuit in addition to the objective of *PART*. However, there is not so much difference between the results of the our algorithm and *PART* at the terms of the number of the subcircuits. The I/O nets tend to be on the min-cuts since the capacity of the flow edges which correspond to the inner nets increase and the capacity of the flow edges which correspond to the I/O nets does not increase. Therefore, the increase of the number of I/Os of the sink side remaining circuit can be avoided, and the increase of the number of the subcircuits can be avoided. In this reason, it is thought that it was possible to control the number of the subcircuits.

Table 2: experimental results

-	*PART*					Our Algorithm				
name	# of device		del		sec	# of device		del		sec
c499	5	(100%)	36	(100%)	9.3	5	(100.0%)	36	(100.0%)	6.5
c880	6	(100%)	59	(100%)	26	4	(66.7%)	44	(74.6%)	40
c1355	5	(100%)	59	(100%)	155	5	(100.0%)	54	(91.5%)	121
c1908	7	(100%)	82	(100%)	233	6	(85.7%)	64	(78.0%)	161
c3540	15	(100%)	84	(100%)	1026	15	(100.0%)	72	(85.7%)	1124
c5315	22	(100%)	89	(100%)	1606	25	(113.6%)	98	(110.1%)	1995
c6288	13	(100%)	184	(100%)	3981	13	(100.0%)	186	(101.1%)	3811
c7552	28	(100%)	82	(100%)	2468	28	(100.0%)	70	(85.4%)	3500
s510	5	(100%)	43	(100%)	10	5	(100.0%)	31	(72.1%)	14
s1196	11	(100%)	79	(100%)	106	13	(118.1%)	79	(100.0%)	155
s5378	24	(100%)	66	(100%)	3227	22	(91.7%)	58	(87.9%)	2880
s9234	40	(100%)	109	(100%)	11655	41	(102.5%)	90	(82.6%)	13096
ave.		100%		100%	100%		98.2%		80.7%	111%

5 Conclusions

In this paper we proposed the delay-aware circuit partitioning algorithm based on the network-flow. Our algorithm controls the cut of the slack-tight net by reflecting the slack to the flow capacity of the partial flow graph corresponding to the net. Our algorithm generates the partitioned circuit whose number of the subcircuits is nearly equal to the one generated by *PART*, and whose clock period is shorter 11% than that generated by *PART*.

An improvement of computation time, an improvement of selection algorithm of vertices which are connected by infinite capacity edges, and to find better weight function of Yang-Wong* Transformation are included in our future work.

Acknowledgement

The authors would like to express thanks to Dr. Kengo R. Azegami for his valuable advice and support.

This work is a part of CAD21 project at Tokyo Institute of Technology.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, "Network Flows: Theory, Algorithms and Applications", Prentice Hall International Inc. 1993

[2] N. S. Woo and Jaeseok Kim, "An Efficient Method of Partitioning Circuits for Multiple-FPGA Implementations", Proceedings of DAC 1993, pp. 202-207

[3] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A survey", Integration, the VLSI Journal, No. 19, 1995, pp. 1-81

[4] H. H. Yang and D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", IEEE Transactions on Computer-Aided Design, Vol. 15, No. 12, December 1996, pp. 1533-1540

[5] H. H. Yang and D. F. Wong, "Circuit Clustering for Delay Minimization under Area and Pin Constraints", IEEE Transactions on Computer-Aided Design, Vol. 16, No.9, September 1997, pp. 976-96

[6] H. Liu and D.F. Wong, "Network-Flow-Based Multiway Partitioning with Area and Pin Constraints", IEEE Transactions on Computer-Aided Design, Vol. 17, No. 1, January 1998, pp. 50-59

[7] K. R. Azegami, A. Takahashi and Y. Kajitani, "Enumerating The Min-Cuts for Applications to Graph Extraction under Size Constraints", Proceedings of the IEEE ISCAS 1999, pp. VI.174-VI.177

[8] K. R. Azegami, M. Inagi, A. Takahashi, and Y. Kajitani, "An Improvement of Network-Flow Based Multi-Way Circuit Partitioning Algorithm" IEICE TRANS. FUNDAMENTALS, VOL.E85-A, NO.3, pp.655–663, MARCH 2002.