Utilizing Voltage-Frequency Islands in C-to-RTL Synthesis for Streaming Applications

Xinyu He* Shuangchen Li * Yongpan Liu* X.Sharon Hu† Huazhong Yang*
Tsinghua National Laboratory for Information Science and Technology, Dept. of EE, Tsinghua University, Beijing, 100084, China*
Dept. of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, U.S.†
{ypliu,yanghz}@tsinghua.edu.cn*, shu@nd.edu†

Abstract—**Automatic C-to-RTL (C2RTL) synthesis can greatly benefit hardware design for streaming applications. However, stringent throughput/area constraints, especially the demand for power optimization at the system level is rather challenging for existing C2RTL synthesis tools. This paper considers a power-aware C2RTL framework using voltage-frequency islands (VFIs) to address these challenges. Given the throughput, area, and power constraints, an MILP-based approach is introduced to synthesize C-code into an RTL design by simultaneously considering three design knobs, i.e., partition, parallelization, and VFI assignment to get the global optimal solution. A heuristic solution is also discussed to deal with the scalability challenge facing the MILP formulation. Experimental results based on *four* well known multimedia applications demonstrate the effectiveness of both solutions.**

I. Introduction

Directly synthesizing the C-based application programs to hardware, i.e., C-to-RTL (C2RTL) synthesis [1] can greatly benefit system-on-a-chip (SoC) design by releasing the time-to-market pressure and filling the gap between hardware productivity and technology capacity. This hasled to a boom in academic and commercial C2RTL tools [2], [3], as well as C2RTL based designs [4].

However, existing C2RTL tools face challenges in terms of synthesis result quality for large C programs [5], especially under stringent constraints from the user. An efficient architecture and high-level optimization are need. Furthermore, low power consumption is essential for SoC design, and power optimization at the system level has a larger design space, which contributes more power reduction than any lower level. It can be extremely beneficial if power optimization can be considered at the C-program level using voltage-frequency islands (VFIs) [6], which is effective to reduce power consumption in the system.

To tackle these challenges, this paper proposes a power-aware C2RTL framework based on VFIs. Given throughput, area, and power constraints, a large C code is partitioned into smaller functional blocks to be synthesized using an existing C2RTL tool. The synthesized modules may also be parallelized to improve throughput. These paralleled modules are then assigned to proper VFIs to reduce power. We consider these three design knobs (partition, parallelization, and VFI assignment) simultaneously with respect to three metrics (throughput, area, and power) as one globe optimization problem. To our best knowledge, this is the first approach with such capabilities.

Authors in [7] consider optimal partition and parallelization under given constraints in C2RTL design flow, however, power optimization is absent. Considering VFI optimization, previous work takes processing cores [8], [9] or task graphs [10], [11] as inputs and assigns or maps them to given VFIs in an NoC architecture. However, we study the VFI in the C2RTL design flow taking C-program as input, which has a fine granularity. The existing work focuses on the given NoC structure, which shares more uniform behaviors than the streaming architecture to be designed in our work. Moreover, we combine VFI together with partition and parallelization in the C2RTL flow to ensure a globe optimal solution.

Our specific contributions include

- introducing a novel MILP based formulation to find an optimal solution to synthesis C code into VFI-based hardware with HLS tools, which **simultaneously** optimizes *partition*, *parallelization* and *VFI assignment* under throughput, area, and power constraints,
- proposing a heuristic solution to handle extremely large-scale instances when MILP takes a long time or fails, and
- validating the proposed methods through developing accelerators for multiple streaming applications.

This work was supported in part by the NSFC under grant 60976032, High-Tech Research and Development (863) Program under contract 2013AA013201, National Science and Technology Major Project under contract 2010ZX03006-003-01, and eXCite software donated from Y Explorations Inc..

DATE'13, March 18-22, 2013, Grenoble, France.
Copyright 978-3-9815370-0-0/DATE13/©2013 EDAA

II. Overview

In this section, we describe our motivation and the proposed power-aware C2RTL framework with VFIs. We then introduce the system models used in our optimization methods.

A. Motivation

Traditional C2RTL approaches [7], [12] transform a C program into hardware with the aim of maximizing throughput or minimizing area while satisfying area or throughput constraints. However, power consumption cannot be controlled or optimized. Yet, our power-aware C2RTL framework considers utilizing VFIs to reduce power at the C-program level and achieves considerable power savings. This is because VFIs can fill the time slack of any block that is not the bottleneck in a pipeline architecture. Table IV in the experiment section proves this point.

Moreover, it is essential to incorporate power optimization at the very beginning of the C2RTL framework, i.e., to consider power optimizing simultaneously with partition and parallelization. Optimizing in separate steps will lead to a sub-optimal solution due to some design space missing. This is due to the fact that the partition and parallelization will fix the hardware architecture and leave little design space to power optimization. Our experimental results show up to 64.3% power savings by considering power optimizing simultaneously with partition and parallelization in the JPEG encoder case.

B. Proposed framework

The high-level design flow of the proposed framework is shown in Figure 1. The original C/C++ program consists of or can be divided into N functions in a straight-line fashion. *STEP 2* is the main concern in this paper, in which we solve the three dimensional optimization problem subject to given constraints. Three operations are conducted, i.e. partitioning the functions into blocks, block-level parallelization to form processing elements (PEs), and giving each PE a VFI assignment. Voltage-frequency assignment selects the best combination of voltage and clock frequency, which optimizes the power consumption under given constraints.

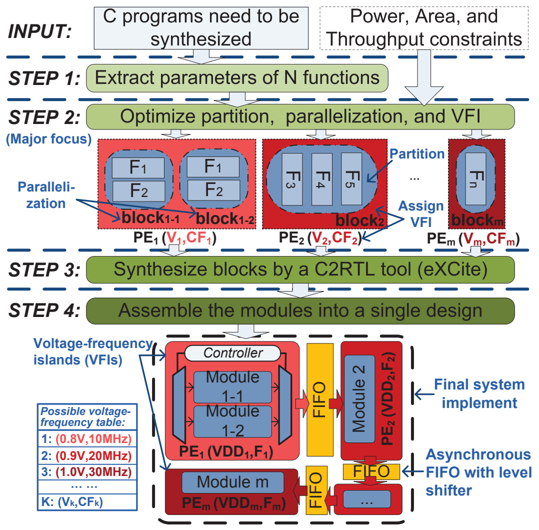

Fig. 1. Power-Aware C2RTL Framework using VFI.

C. System modeling

To support the exploration of the large hardware design space, we need to carefully model a given C program. Our system model focuses on characterizing the power, throughput and area parameters of functions, blocks and system, as shown in Table I. The parameters for the n-th function in a C program are listed in rows marked by F_n. We denote a block as $B_{i,j}$, which consists of adjacent functions of F_i, F_{i+1}, $\cdots$, F_j, and its parameters are listed in rows marked by $B_{i,j}$. The system parameters are summarized in rows marked by *system*.

TABLE I
PARAMETERS FOR F_n, $B_{i,j}$, AND *system*

	Parameters	Description
F_n	N	Total number of functions
	$T_n^{\mathrm{i}}/T_n^{\mathrm{o}}$	Input/output latency of F_n (cycle)
	$S_n^{\mathrm{i}}/S_n^{\mathrm{o}}$	Input/output data size for F_n (byte)
	$A_n^{\mathrm{l}}/A_n^{\mathrm{m}}$	Area of logic/memory used by F_n (μm^2)
	$\mathrm{CF}_n^{\max}$	The highest clock frequency of F_n (MHz)
	$P_{n,k}$	Power of F_n under (V_k,CF$_k$) (mW)
$B_{i,j}$	$T_{i,j}$	Latency of block $B_{i,j}$ (cycle)
	$T_{i,j}^{\mathrm{i}}/T_{i,j}^{\mathrm{o}}$	Input/output latency of $B_{i,j}$ (cycle)
	$S_{i,j}^{\mathrm{i}}/S_{i,j}^{\mathrm{o}}$	Input/output data size for $B_{i,j}$ (byte)
	$A_{i,j}^{\mathrm{l}}/A_{i,j}^{\mathrm{m}}$	Area of logic/memory by $B_{i,j}$ (μm^2)
System	R_{req}	Throughput constraint (byte/us)
	A_{req}	Area constraint (μm^2)
	A_{fifo}	Area cost by each byte in FIFO (μm^2/byte)
	P_{req}	Power constraint (mW)
	P_{fifo}	Power cost by each byte in FIFO (mW/byte)
	$\{(V_k,\mathrm{CF}_k)\}$	Possible (voltage,frequency) vectors for VFI
	K	Number of (V_k,CF$_k$) vectors for VFI

III. MILP-BASED SOLUTION

A. Variable definitions

The variables used in the MILP formulation are listed in Table II. We use $\{x_n\}$ to indicate the parallelization degree and partition result. $\{x_n\}$ is non-zero only when F_n is the last module in a clustered block, and in this case, its non-zero value represents the parallelization degree of the block. $\{y_{i,j}\}$ indicates the partition result, which equals to one only if F_i, .., F_j are merged into a block $B_{i,j}$.

To indicate which voltage-frequency pair is adopted by each island, we use another 2-D binary variable $\{c_{n,k}\}$. If $c_{n,k}$ equals to one, it means the k-th pair of voltage-frequency (V_k,CF$_k$) is selected by F_n. An example $\{c_{n,k}\}$ for Figure 1 is listed in Equation (1). Other defined variables are listed in Table II.

$$\{c_{n,k}\} = \left\{ \begin{array}{cccc} 1, & 0, & 0, & \cdots \\ 1, & 0, & 0, & \cdots \\ 0, & 1, & 0, & \cdots \\ 0, & 1, & 0, & \cdots \\ 0, & 1, & 0, & \cdots \\ \cdots & & & \\ 0, & 0, & 1, & \cdots \end{array} \right\} \quad (1)$$

TABLE II
VARIABLES USED IN THE MILP FORMULATION

Name	Description
x_n	Representing partition and parallelization
$y_{i,j}$	Representing partition information
$c_{n,k}$	Representing VFI assignment information
$\mathrm{cf}_{i,j}$	Clock frequency of $B_{i,j}$ (MHz)
r_{all}	Throughput of the system (byte/μs)
$r_{i,j}$	Rate of $B_{i,j}$ after parallelization (1/μs)
$a_{\mathrm{fifo}}^{\mathrm{l}}/a_{\mathrm{fifo}}^{\mathrm{m}}$	Total FIFO Area (μm^2)
a_{all}	Total Area of the system (μm^2)
$p_{i,j}^{\mathrm{l}}/p_{i,j}^{\mathrm{m}}$	Power of logic/memory by $B_{i,j}$ (mW)
$p_{\mathrm{fifo}}/p_{\mathrm{pe}}$	Total FIFO/PE power in system (mW)
p_{all}	Total power of the system (mW)

B. MILP formulation

Using the parameters and variables in Table I and II, we have the MILP formulation[1] as follows:

$$\begin{array}{lllll} obj: & \min p_{\mathrm{all}} & \text{or} & \max r_{\mathrm{all}} & \text{or} \quad \min a_{\mathrm{all}} \\ s.t.: & r_{\mathrm{all}} \geq R_{\mathrm{req}} & \text{and} & a_{\mathrm{all}} \leq A_{\mathrm{req}} & \text{and} \quad p_{\mathrm{all}} \leq P_{\mathrm{req}} \end{array} \quad (2)$$

Below we describe each type of constraints in detail.

*1) **Throughput constraints**:* Obviously, the system throughput is determined by the bottleneck block, i.e., the slowest block. Hence,

$$r_{\mathrm{all}} = S_N^{\mathrm{o}} \cdot \min r_{i,j} \quad \forall r_{i,j} > 0 \quad (3)$$

$r_{i,j}$ is the rate of each PE, which is influenced by partition ($y_{i,j}$), parallelization (x_i), VFI assignment (cf$_{i,j}$) and block latency ($T_{i,j}$). Therefore, we have:

$$r_{i,j} = x_j \cdot y_{i,j} \cdot \mathrm{cf}_{i,j}/T_{i,j} \quad (4)$$

where cf$_{i,j}$ selects a pair of voltage and frequency values from $\{(V_k,\mathrm{CF}_k)\}$, i.e.,

$$\mathrm{cf}_{i,j} = \sum_{k=1}^{K} c_{j,k} \cdot \mathrm{CF}_k \quad (5)$$

[1] This is a multiple objective optimization using lexicographic method. For example, when minimizing p_{all}, we have three iterations: (i) minimize p_{all}, (ii) minimize a_{all} with P_{req}=p_{all}, (iii) maximize r_{all} with A_{req}=a_{all} and P_{req}=p_{all}.

*2) **Area constraints**:* The total area includes two parts: the blocks and the FIFOs (a_{fifo}) connecting adjacent blocks. Hence, we have:

$$a_{\text{all}} = \sum_{i=1}^{N}\sum_{j=i}^{N}(A^{\text{le}}_{i,j} + A^{\text{mem}}_{i,j}) \cdot x_j \cdot y_{i,j} + a_{\text{fifo}} \tag{6}$$

$A^{\text{le}}_{i,j}$ and $A^{\text{mem}}_{i,j}$ make up the area of $B_{i,j}$. They are calculated as follow,

$$A^{le}_{i,j} = \begin{cases} \sum_{\text{n=i}}^{j} A^{\text{le}}_{n} \cdot (1-\alpha^{\text{le}}) & i<j \\ A^{\text{le}}_{i} & i=j \end{cases} \tag{7}$$

$$A^{\text{mem}}_{i,j} = \max A^{\text{mem}}_{k} \qquad \forall k \in [i,j] \tag{8}$$

In Equation (7), α^{le} is the area saving factor when functions are clustered into a block, since some logic elements can be shared. In Equation (8), since functions in one block work serially, they can share one memory without conflicts. Therefore, the function with the largest memory area determines $A^{\text{mem}}_{i,j}$.

a_{fifo} should be proportional to the total size of input/output data. Therefore, we have:

$$a_{\text{fifo}} = (A^{\text{le}}_{\text{fifo}} + A^{\text{mem}}_{\text{fifo}}) \cdot \sum_{N-1}^{i=1}\sum_{j=i}^{N} z_{i,j} \cdot S^{\text{out}}_{j} \tag{9}$$

*3) **Power constraints**:* The total power consumption of the system can be calculated by Equation (10), which consists of the power of all PEs and asynchronous FIFOs. Equation (11) shows the PE's power, which equals to the sum of the power consumed by the blocks (after duplicating as indicated by x_j and $y_{i,j}$). In Equation (12), the FIFO overhead is estimated by the output data size of the duplicated blocks.

$$p_{\text{all}} = p_{\text{fifo}} + p_{\text{pe}} \tag{10}$$

$$p_{\text{pe}} = \sum_{i=1}^{N}\sum_{j=i}^{N} x_j \cdot y_{i,j} \cdot p_{i,j} \tag{11}$$

$$p_{\text{fifo}} = P_{\text{fifo}} \cdot \sum_{i=1}^{N-1}\sum_{j=i}^{N}(x_j \cdot y_{i,j}) \cdot S^{o}_{j} \tag{12}$$

In Equation (11), $p_{i,j}$ represents the total power consumed by block $B_{i,j}$. $p_{i,j}$ can be broken down to the logic part and the memory part in Equation (13). It computes the power consumed by the logic part of $B_{i,j}$ in Equation (14). We use $c_{j,k}$ to select the voltage and frequency of this block. The power of the block is calculated by the total energy ($\sum_{n=i}^{j} P^{l}_{n,k} \cdot T_n$) divided by the block's latency $T_{i,j}$. In Equation (15), the memory part of the power is considered. As stated in *Area Constraints*, the memory in a block can be multiplexed, and the power is estimated by the power of the functions clustered with the maximum memory, which is also selected by $c_{j,k}$.

$$p_{i,j} = p^{l}_{i,j} + p^{m}_{i,j} \tag{13}$$

$$p^{l}_{i,j} = \begin{cases} \sum_{k=1}^{K}(c_{j,k} \cdot \frac{\sum_{n=i}^{j} P^{l}_{n,k} \cdot T_n}{T_{i,j}}) & i<j \\ \sum_{k=1}^{K}(c_{j,k} \cdot P^{l}_{i,k}) & i=j \end{cases} \tag{14}$$

$$p^{m}_{i,j} = \sum_{k=1}^{K}(c_{j,k} \cdot \max_{n=i}^{j} P^{m}_{n,k}) \tag{15}$$

To sum up, the entire MILP formulation consists of Equations (2) to (15) and some connectivity constraints. We use the tool of *lp_solve* to solve the MILP problem in Section V.

IV. Heuristic solution

It is well-known that MILP based methods are unscalable. Our experiments in Section V show that it is time consuming even when N is small if the constraint regions are *un-smooth*. In this section we propose a heuristic algorithm to tackle these challenges.

Our heuristic algorithm creatively transforms the optimization problem into a constrained shortest path searching issue. First, we create a directed acyclic graph DAG G=(V,A) with parameters in Table I. Second, we use a modified A* Search Algorithm [13] to get the solution. We take the objective of $\min p_{\text{all}}$ as an example to describe the algorithm.

Creating DAG: Each possible block with each possible VFI assignment is regarded as a node. Hence, if there are N functions and K groups of VFI, we have $C^{2}_{\text{N+1}}$ possible clustered blocks, and accordingly $K \cdot C^{2}_{\text{N+1}}$ nodes. The parallelization degree of each node is the minimum number that satisfies the throughput constraint R_{req}. This is because enlarging parallelization degree increases throughput at the cost of more power and area. Therefore, setting it as the minimum to exceed R_{req} is enough. Then two virtual nodes are added as Source (*S*) and Destination (*D*). In the graph, arrows only connect nodes representing adjacent blocks, i.e., nodes representing $B_{i,j}$ and $B_{j+1,k}$ ($\forall i \le j < k$). Each arrow is associated with two weights, namely power and area of its source node. From the DAG we created, any path connecting *S* and *D* represents a possible solution with the cost of p_{all} and a_{all}. The problem is equivalent to finding the shortest power path while obeying the area constraint.

Modified A* Search: First, we perform Dijkstr's Algorithm to get the heuristic function $h(n)$[2]. The shortest path (with power weight) from each node to D is calculated as $h(n)$. Second, we perform a modified A* Search Algorithm. Different from original A*, we use the function *minFeasibleNodeInOPEN()* (shown in Algorithm 1) to pick and move nodes from OPEN set (nodes to be selected) to CLOSED set (nodes that have been evaluated). The function guarantees that the area constraint is satisfied whenever adding a node to the current path.

In function *minFeasibleNodeInOpen()*, we first find the node V in OPEN with the minimal estimate function $f(n)$ (Line 2). Whether V can be moved from OPEN to CLOSED depends on the following three cases. In *Case I* (Line 4-6), if at least one child node Q of V can make the extended path (with V and Q) obey the area constraint, V is moved to CLOSED, and V's *qualified* child nodes are added to OPEN. *Qualified* means that after adding the node to current path, the area cost won't exceed A_{req}. However, if we fail to get such a Q and adding any of V's child node breaks the area constraint, we go to *Case II* (Line 8-10) or *Case III* (Line 12). In these two cases, the current path containing V can't continue. We then try to find a suboptimal path through V. If this can be found, we replace the current path with the suboptimal one (*case II*) . If not, V has to be temporarily discarded (*case III*). To find the suboptimal path, we check whether V has parent nodes other than P in CLOSED (we denote V's previous node as P). If it does, we enter *Case II*. We select the node P' with the smallest cumulative cost $g(n)$ other than P, and set P' as V's previous node. This makes a new path going through V. We update the estimate $f(n)$, the cumulative power $g(n)$ and cumulative area $g_a(n)$, and then restart the loop. In *Case III*, no path through V could be found. Hence, V can't exist in any path currently (it may appear in a path in future). We remove V from OPEN and start the loop again (Line 11). Other parts are exactly the same as the original A* Algorithm.

The optimization of area follows the same procedure as power. As to throughput optimization, we briefly turn each node into

[2] For each node, we have the cumulative cost $g(n)$, the heuristic function $h(n)$, and their sum, the estimate function $f(n)$, as defined in [13].

Algorithm 1: Function *minFeasibleNodeInOPEN()*

```
input : {OPEN}, {CLOSED}
repeat
    V=nodeWithSmallestEstimationInOPEN(OPEN);
    for every child node (Q) of V do
        if $Q.area + V.accumulatedArea < A_{ref}$ then
            return $V$;    /* Case I */
        end
    end
    if $V$ has parent nodes other than $P$ in CLOSED then
        Set the node with the smallest $g$ as $V$'s previous node;
        Update $g(n)$, $g_a(n)$ and $h(n)$; /* Case II */
    else
        Remove $V$ from OPEN; /* Case III */
    end
until;
```

several nodes representing different parallelization degrees and add a throughput weight to each arrow. We also get $f(n)$ as the minimum of cumulative cost $g(n)$ and heuristic $h(n)$ instead of their sum. The main procedure is still the same, so we omit detailed explanation because of space limit.

V. Experimental Evaluation

In this section, we present our experimental results based on four real streaming programs from [14]. We use *eXCite* [3] for C2RTL synthesis, *Design Compiler* for RTL synthesis (*SMIC* $0.13\mu m$ technology is used), *ModelSim* for back-end simulation, and *Prime Time* for power measurement.

Both the MILP solution and the heuristic solution rely on the accuracy of the model we used. Table III compares the throughput, area and power values from the proposed models (see (3) to (12)) and these from the synthesized hardware. We can conclude from Table III that our models are accurate enough for high-level synthesis. The errors mostly come from the estimation of the FIFO size. However, such differences are tolerable for the system-level models.

TABLE III
System model validation

Benchmark	from proposed Models			from Synthesized Hardware		
	r_{all}	a_{all}	p_{all}	r_{all}	a_{all}	p_{all}
JPEG decoder	0.0227	1401600	15.32	0.0230	1324675	17.25
JPEG encoder	0.0563	1063400	17.29	0.0563	1054428	16.70

We evaluate the MILP based solution in Table IV. First, we compare the result from our MILP-based solution, the heuristic solution, and a *local optimal* solution under the same constraints while optimizing power. The *local optimal* solution is a multi-step optimization that minimizes area first with the method in [7], and after that each of the PE's voltage and frequency are set to the lowest one while not violating the throughput constraint to minimize power. Compared with the *local optimal* method, the power savings with our globe optimization method are shown in the eighth column. The running time of the MILP-based solution and the heuristic solution is shown in the last column.

From Table IV we can conclude that our MILP-based solution is effective. It meets the throughput and area constraints, and saves more power than the solution from the *local optimal* method. Considering the feature of MILP, the results from MILP-based solution is optimal. Meanwhile, our method saves at most 64.3% power than that from the *local optimal* solution. This comparison emphasizes our contribution and also demonstrates the necessity of simultaneously considering throughput, area, and power as a globe optimization problem.

TABLE IV
Power optimization results for four benchmarks
(obj: min p_{all})[1]

Benchmark		Constraint		Results[2]			Power saving	time (sec.)
		R_{req}	A_{req}	r_{all}	a_{all}	p_{all}		
AES encryption N=7	MILP	0.0167	2.2*10^6	0.0185	2068124	60.10	38.4%	7.9
	Heuristic			0.0185	2068124	60.10		0.001
	Local opt.			0.0169	1981901	97.50		–
JPEG decoder N=8	MILP	0.0200	1.8*10^6	0.0210	1721100	9.55	23.6%	62.5
	Heuristic			0.0210	1721100	9.55		0.003
	Local opt.			0.0227	1401572	12.50		–
JPEG encoder N=10	MILP	0.1333	1.8*10^6	0.1383	1691154	12.06	64.3%	108.6
	Heuristic			0.1383	1691154	12.06		0.005
	Local opt.			0.1408	1425830	33.75		–
GSM N=10	MILP	0.0500	1.0*10^6	0.0521	975293	40.58	14.4%	4291
	Heuristic			0.0521	975293	40.58		0.002
	Local opt.			0.0507	902510	47.41		–

[1] We implement a multiple objective optimization using lexicographic method, with the preferred sequence of p_{all}, a_{all}, r_{all}.
[2] There are 5 voltage-frequency pairs available to be selected, i.e., (0.8V, 10MHz), (0.9V, 20MHz), (1.0V, 30MHz), (1.1V, 40MHz), (1.2V, 50MHz).

The results in Table IV also show that the heuristic solution is effective, which always generates the same solution as the optimal MILP with a much shorter running time. The results in Table IV show about an average 400'000x speedup in running time. The speedup will be more significant when N becomes larger.

VI. Conclusions and Future work

In this paper, we focus on the development of accelerators for streaming applications with C2RTL design flow. We propose an MILP solution to find the optimal C-code partition, block-level parallelization and VFI assignment simultaneously as a globe optimization instead of a multi-step optimization. To tackle the scalability challenge that MILP faces, an efficient heuristic algorithm is also proposed. Experimental results obtained from four real applications show that both approaches are effective in exploring the throughput, area, and power design space.

References

[1] J. Cong, B. Liu, and et al., "High-level synthesis for fpgas: From prototyping to deployment," *TCAD*, vol. 30, no. 4, pp. 473–491, 2011.
[2] S. Gupta and et al., "Spark: A high-level synthesis framework for applying parallelizing compiler transformations," in *ISVLSI*, 2003, pp. 461–466.
[3] "*eXCite*," http://www.yxi.com.
[4] B. Schafer and et al., "Design of complex image processing systems in esl," in *ASP-DAC*, 2010, pp. 809–814.
[5] S. Li and et al., "A hierarchical c2rtl framework for fifo connected stream applications," in *ASP-DAC*, 2012, pp. 133–138.
[6] D. Lackey and et al., "Managing power and performance for system-on-chip designs using voltage islands," in *ICCAD*, 2002, pp. 195–202.
[7] S. Li and et al., "Optimal partition with block-level parallelization in c-to-rtl synthesis for streaming applications," in *ASPDAC*, 2013.
[8] K. Niyogi and et al., "Speed and voltage selection for gals systems based on voltage/frequency islands," in *ASP-DAC*, 2005, pp. 292–297.
[9] N. Kapadia and et al., "Vision: a framework for voltage island aware synthesis of interconnection networks-on-chip," in *GLSVLSI*, 2011, pp. 31–36.
[10] Y. Liu and et al., "Clustering-based simultaneous task and voltage scheduling for noc systems," in *ICCAD*, 2010, pp. 277–283.
[11] U. Ogras and et al., "Design and management of voltage-frequency island partitioned networks-on-chip," *TVLSI*, vol. 17, no. 3, pp. 330–341, 2009.
[12] J. Cong and et al., "Combining module selection and replication throughput-driven streaming programs," in *DATE*, 2012, pp. 1018–1023.
[13] P. Hart and et al., "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, 1968.
[14] Y. Hara and et al., "Chstone: A benchmark program suite for practical c-based high-level synthesis," in *ISCAS*, 2008, pp. 1192–1195.