<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Multivoltage multifrequency low-energy synthesis for functionally pipelined datapath</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Xing, Xianwu; Jong, Ching Chuen</td>
</tr>
<tr>
<td><strong>Date</strong></td>
<td>2009</td>
</tr>
<tr>
<td><strong>URL</strong></td>
<td><a href="http://hdl.handle.net/10220/6239">http://hdl.handle.net/10220/6239</a></td>
</tr>
<tr>
<td><strong>Rights</strong></td>
<td>© 2009 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. <a href="http://www.ieee.org/portal/site">http://www.ieee.org/portal/site</a> This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.</td>
</tr>
</tbody>
</table>
Multivoltage Multifrequency Low-Energy Synthesis for Functionally Pipelined Datapath

Xianwu Xing and Ching Chuen Jong

Abstract—In this paper, an algorithm named MuV oF is proposed to perform multivoltage multifrequency low-energy high-level synthesis for functionally pipelined datapath under resource and throughput constraints. A datapath is partitioned into a number of pipelined stages such that the clock period can be extended maximally. A multivoltage assignment algorithm then utilizes the extended clock period to reduce energy by lowering the supply voltages of the resources. The results are further refined by four local transformations performed in an iterative process. The experiment results show that MuV oF is capable of exploring the design space effectively and achieves efficient energy reduction.

Index Terms—Functional pipelining, low energy, multivoltage multifrequency (MuV oF), high-level synthesis.

I. INTRODUCTION

As the density of VLSI systems is rapidly increasing and portable and wireless devices, in which energy consumption must be reduced to extend battery life, are more and more popular, energy consumption has become a more critical design factor than circuit size. The well-known formula $E_{\text{dynamic}} = 0.5 \cdot V_d^2 \cdot C_{\text{out}} \cdot E_{\text{transition}}$ shows that the dynamic energy consumption depends on three parameters: the supply voltage ($V_d$), the load capacitance ($C_{\text{out}}$), and the switching activity ($E_{\text{transition}}$). Power/energy optimization based on switching activity reduction has been widely explored [1], [2]. Due to the quadratic dependency on supply voltage, power/energy optimization based on voltage scaling is regarded as one of the most effective methods. Multivoltage techniques have gained popularity among voltage scaling techniques [3]–[6]. In [3], it was demonstrated that multivoltage scheduling problem was NP-hard and a dynamic programming approach was proposed. In [4] and [5], mobility-based heuristic algorithms were proposed for the scheduling problem of multivoltage datapath for simultaneous energy and transient power reduction. There were also works on frequency scaling. A multiple-clock scheme for register transfer level (RTL) datapath synthesis was proposed in [7]. The authors of [4]–[6] used dynamic frequency clocking (DFC) to achieve optimization on variable power and energy parameters. DFC does not directly reduce power, but compensates the extended delay caused by reducing voltage supply of resources.

In this paper, a technique named multivoltage multifrequency (MuV oF) is proposed for low-energy high-level synthesis for functionally pipelined datapath under resource and throughput constraint. Functionally pipelined structure has existed for a long time, but has not been explored for low-energy high-level synthesis. In MuV oF, the datapath is first partitioned into several pipelined stages, each of which operates at its own frequency subject to a global throughput constraint. The proposed stage partitioning algorithm optimally creates large slacks in some stages so that their clock periods can be extended under the throughput constraint. Then, a multivoltage assignment process utilizes the extended clock period for supply voltage reduction. To our best knowledge, this is the first technique addressing the high-level synthesis of multivoltage and multifrequency functionally pipelined datapath with the low energy as the optimization goal.

The next section introduces the multifrequency and multivoltage synthesis for functionally pipelined datapath. Section III describes the proposed algorithms. Section IV presents the test results and Section V concludes the paper.

II. MULTIVOLTAGE AND MULTIFREQUENCY SYNTHESIS FOR FUNCTIONALLY PIPELINED DATAPATH

MuV oF synthesizes energy-efficient multivoltage functionally pipelined datapath from a scheduled data flow graph (DFG). We use the DFG of an infinite impulse response (IIR) filter shown in Fig. 1(a) to illustrate the concept. The schedule is represented as $S_{\text{INIT1}}$, with the operations in the same control step (cstep) enclosed by a pair of brackets "( )", i.e.,

$$S_{\text{INIT1}} = \{(17, 16), \ (15, 19), \ (16, 21, 20, 14), \ (23), \ (13, 18), \ (14, 22), \ (24), \ (26), \ (25), \ (27), \ (28), \ (15, 29)\}.$$

A. Functional Pipelining for Power/Energy Optimization

A functionally pipelined datapath consists of several partitioned stages. Each stage has its own resources and operates at its own clock period. Let us consider $S_{\text{INIT1}}$ with a single supply of 5.0 V and a single clock period of 30 ns. The total delay would be $16 \times 30 \text{ ns} = 480 \text{ ns}$. If the input data arrives faster than this, additional resources are required to construct a pipelined datapath in order to satisfy the throughput. Suppose the data arrives every 300 ns and four multipliers and two arithmetic logic units (ALUs) are now available, the schedule $S1$ below is a feasible datapath with two pipelined stages,
each enclosed by a pair of “{ },” The datapath now has a total delay of 10 × 30 ns = 300 ns

\[
S_1 = \{(17, 16), (19, 15), (14, 20, 21), (23)\} \\
+ \{(13, 18), (22), (24), (26), (25), (27), (28), (29)\}
\]

The functionally pipelined datapath provides space for power/energy optimization. As different stages have different latencies, the clock periods in the stages with shorter latencies can be extended, resulting in peak power reduction. The extended clock period also facilitates supply voltage reduction, which reduces both power and energy. In addition, the operations in a stage have less dependency constraints than those in a nonpipelined datapath, thus creating more spaces for voltage reduction, as further discussed below.

B. Multivoltage and Multifrequency Synthesis

Given ALU and multiplier as the available logic components, their delay (ns) and energy (pJ) characterization in Table I is adopted from [4].

In a functionally pipelined datapath, the total energy \( E \) can be modeled as (1), where \( N \) is the number of stages, \( M_i \) is the number of resources in stage \( i \), and \( \alpha_{i,j} \) is the switching activity

\[
E = \sum_{i=1}^{N} E_i = \sum_{i=1}^{N} \sum_{j=1}^{M_i} C_{i,j} V_{i,j}^{\alpha_{i,j}}.
\]

For simplicity, the examples here consider energy dissipated by functional units only. For \( S_1 \), if the clock period of stage 1 remains as 30 ns, the energy is \( E_1 = 2202 \times 9 + 57 \times 8 = 20274 \) pJ. However, since the clock period can be 50 ns now, there is longer clock cycle slack in this stage, which will enable the voltages of the resources to be lowered. Fig. 1(b) shows a feasible solution, represented by \( S_2 \) below, where the superscript on a node indicates the supply voltage assigned to the node while the superscript on a cstep or a stage applies to all the nodes of the cstep or the stage, respectively. The default supply voltage is 5.0 V

\[
S_2 = \{(17, 16), (19, 15), (14, 20, 21), (23)\}^{3.3 \text{ V}} \\
+ \{(13, 3.3 \text{ V}), (18), (22), (24), (26), (25), (27), (28), (29)\}
\]

In Fig. 1(b), all operations in stage 1 can be assigned with 3.3 V without violating the timing constraint. The voltage of operation 13 in stage 2 is also reduced to 3.3 V, which is now possible due to looser dependency constraints in stage 2 than in the original DFG. If operation 13 of the original DFG in Fig. 1(a) was assigned with 3.3 V, there would be one 3.3-V multiplier and one 5.0-V multiplier. In this case, either operation 20 or operation 14 has to go to cstep 7, breaking the resource constraints in cstep 7. This situation does not occur after the partition, as shown in Fig. 1(b). The energy of \( S_2 \) is \( E_2 = (960 \times 6 + 25 \times 2) + (960 \times 1 + 2202 \times 2 + 57 \times 6) = 11516 \) pJ, giving a 43.2% reduction over \( E_1 \).

C. Influence of Stage Partitioning and Scheduling

Different stage partitions result in different frequency scaling in the stages, which affects the multivoltage assignment. Consider a different partition scheme \( S_3 \)

\[
S_3 = \{(17, 16^{3.3 \text{ V}}), (15, 19), (21, 20, 14^{3.3 \text{ V}}), (23), (13, 18), (22), (24), (26), (25), (27), (28), (29)\}
\]

With this partition, the clock period can be extended to 300 ns/8 = 37.5 ns for both stages. After the voltage scaling is performed, stage 1 requires one 3.3-V multiplier, two 5.0-V multipliers, and one 5.0-V ALU, while stage 2 needs one 5.0-V multiplier and one 5.0-V ALU. The total resources are still four multipliers and two ALUs. The energy of \( S_3 \) is \( E_3 = (960 \times 2 + 2202 \times 6 + 57 \times 2) + (2202 \times 1 + 57 \times 6) = 17970 \) pJ. This achieves only 12.3% reduction when compared to \( E_1 \), which is not as significant as in \( E_2 \).

Another factor that influences the energy reduction is the initial schedule. Consider an alternative initial schedule

\[
S_J \text{INIT} \text{2} = \{(15, 17, 14, (19, 20, 16), (13, 18), (21), (23), (25), (27), (28), (29))\}
\]

One possible partition scheme with three pipelined stages is

\[
S_4 = \{(17, 14, 15), (20, 16, 19), (18, 13)\}^{3.3 \text{ V}} \\
+ \{(21), (23), (25)\}^{2.4 \text{ V}} \\
+ \{(22^{3.3 \text{ V}}), (24^{2.4 \text{ V}}), (26^{3.3 \text{ V}}), (28^{2.4 \text{ V}}), (29^{3.3 \text{ V}})\}
\]

Stage 1 has three 3.3-V multipliers and a clock period of 300 ns/3 = 100 ns. Stage 2 has one 2.4-V ALU and a clock period of 100 ns. Stage 3 has one 2.4-V multiplier and one 3.3-V ALU and a clock period of 60 ns. The energy of \( S_4 \) is \( E_4 = (960 \times 8) + (13 \times 3) + (507 \times 1 + 25 \times 5) = 8351 \) pJ, which is 58.8% reduction over \( E_1 \), even better than the 43.2% obtained in \( E_2 \).

Therefore, the three major factors influence the energy consumption in a functionally pipelined datapath are the initial schedule, the stage partition, and the multivoltage assignment.

III. ALGORITHM FORMULATION

MuVoF is proposed to optimize the above three factors in an integrated manner. It consists of two main processes: a core process and an iterative process as described below.

A. The Core Process

Given a scheduled DFG, a resource constraint, and a throughput constraint, the core process partitions the DFG into several pipelined stages to optimize the clock periods and performs multivoltage assignment to reduce energy.

1) Stage Partition: The objective of the stage partitioning is to divide the operations into a number of pipelined stages such that each stage has the maximum clock period, while still satisfying the resource and throughput constraints. An optimal partition is the one that allows the clock period to be extended maximally such that the energy has the best opportunity to be reduced via voltage reduction. The stage partition is given in Algorithm 1.

// Algorithm 1: Stage Partition

// Input: a scheduled DFG (G), resource constraint (R), cycle-time (T), throughput constraint (TP) and (N).
// number of csteps (N).
// Output: a functionally pipelined scheduled DFG, with different clock frequencies in different stages.
1: current_cost = \infty;
2: s_0 = initial_partition(); // initial empty partition
3: s_first = append_to_partition(s_0, 1);
4: S = {s_first};
5: while (S ≠ \Phi) {
6: select an element \( s \in S \);
7: \( S = S \backslash \{s\} \);
\hspace{1em} \text{// first branch}
8: \( s1 = \text{append_to_partition} \ (s, s, s.\text{step} + 1) \);
\hspace{1em} \text{// if satisfy throughput constraints}
9: if (\( s1.\text{current\_stage\_len} \times T \leq 1/TP \)) {
10: \hspace{1em} if (\( s1.\text{low\_bound} < \text{current\_cost} \)) {
11: \hspace{2em} \( S = S \cup \{s1\} \);
12: \hspace{2em} if (\( s1.\text{step} == N \)) {
13: \hspace{3em} \text{current\_best} = s1;
14: \hspace{3em} \text{current\_cost} = s1.\text{low\_bound};
15: \hspace{2em} }
16: \}\}
17: \text{// second branch}
18: \( s2 = \text{start\_new\_partition} \ (s, s.\text{step} + 1) \);
19: if (\( s.\text{resources} + s.\text{usage} (s.\text{step} + 1) \leq R \)) {
20: \hspace{1em} if (\( s2.\text{low\_bound} < \text{current\_cost} \)) {
21: \hspace{2em} \( S = S \cup \{s2\} \);
22: \hspace{2em} if (\( s2.\text{step} == N \)) {
23: \hspace{3em} \text{current\_best} = s2;
24: \hspace{3em} \text{current\_cost} = s2.\text{low\_bound};
25: \hspace{2em} }
26: \}\}
27: \} \}
28: \text{// perform clock period extension}
29: \text{update\_stage\_clock} \ (\text{current\_best} \).

The process adopts a binary branch-and-bound approach. It starts with an empty partition (line 2) that has no \( cstep \) processed. The \( csteps \) are then processed from the first to the last sequentially. A solution is defined as an intermediate partition, in which the stages of some \( csteps \) have been determined while the others are not yet fixed. Given a solution, for each \( cstep \) not yet processed, there are two options that form the two branches of the binary branch-and-bound algorithm. The first option (lines 8–15) is to put the \( cstep \) into the current stage (line 8). This option increases the latency of the current stage, and thus the throughput constraints might be violated. This is the first bound condition for the first branch (line 9). The second option (lines 16–23) is to put the \( cstep \) into a new stage (line 16). This option increases the resources because each pipelined stage consumes its own resources. Thus the resource constraints might be violated and this is the first bound condition for the second branch (line 17).

If no constraint is violated when either the first or the second option is chosen, the lower bound cost of the new solution is calculated. When calculating the cost of a solution \( s \), a dummy solution \( s' \) is derived from \( s \), assuming that each unprocessed \( cstep \) occupies one separate stage. The cost of \( s' \) is the second bound condition for both branches. If it is less than the current cost, the new solution is added to the solution set. We define the cost function as (2), where \( N \) is the number of stages, \( M_i \) is the number of resources in stage \( i \), \( E_{i,j} \) is the energy required for executing the operations, and \( f_i \) is the clock frequency of stage \( i \). The cost function enforces that energy-intensive operations (such as multiplications) are to be assigned to the stages that have lower frequency, so that they have the best chance to be assigned to the low-voltage resources in the multivoltage assignment process described later

\[
\text{cost} = \sum_{i=1}^{N} \sum_{j=1}^{M_i} E_{i,j} f_i. \tag{2}
\]

During the cost calculation, a clock period extension process is performed on each stage. The types of extension include the following: 1) direct extension, which directly extends the clock period to its maximum under the throughput constraint; 2) step merging, which merges consecutive \( csteps \) into one \( cstep \); and 3) deserted step removal, which removes \( csteps \) with no operations due to direct extension and step merging.

If a solution derived from option 1 or 2 does not break any constraints, its cost will be evaluated. If the cost is acceptable but there are unprocessed operations left, the solution is added as a new branch. If the solution is complete (i.e., all operations have been processed) and its cost is less than the current best cost, the solution is saved and the current best cost is updated (lines 12 and 20). The solution with the best cost is the final solution, of which the clock period for each stage is extended to the maximum under the constraints (line 24). When extending a clock period, the new period is selected from a predefined clock set to limit the number of clocks synthesized.

2) Multivoltage Assignment: After the stage partition, the multivoltage assignment is performed on each stage. The technique is formulated as Algorithm 2, which tries to reduce the supply voltages in a step-by-step manner (line 5) for each resource type. With a set of voltages for a particular resource type, the supply voltages of the resources (\( FUs \)) are reduced gradually from the highest to the lowest voltage by the function reassign \((v_{Hi}, v_{IL}, R, L, t)\) in line 6. The function tries to reduce the voltage of some resources of type \( t \) from \( v_{Hi} \) to \( v_{IL} \) under the resource constraint \( R \) and the latency constraint \( L \) and is described as Algorithm 3.

// Algorithm 2: Multivoltage Assignment

// Input: resource types (RT), resource constraint (RC),
// Latency constraint (L) and number of available voltages (M)
1: for each \( t \in \text{RT} \) {
2: \hspace{1em} \( V = \{v_i| i \in [1, M], v_1 > v_2 > \ldots > v_M\}; // voltage set \)
3: \hspace{1em} \( R = \text{RC}(t); \)
4: \hspace{1em} for each \( r \in R \{r.\text{voltage} = v_1; \} // for each FU \)
5: \hspace{1em} for (\( i = 1; i < M; i++ \)) {
6: \hspace{2em} \text{reassign} (v_i, v_{i+1}, R, L, t); // see Algorithm 3
7: \}

// Algorithm 3: Voltage Reassignment

// Function reassign \((v_{Hi}, v_{IL}, R, L, t)\)
1: \( \text{PR} = \Phi; // resources which have already been processed \)
2: \text{Level1: while (}\( r \in R \& \& r \notin \text{PR} \& \& r.\text{voltage} == v_{Hi} \}) {
3: \hspace{1em} r.\text{voltage} = v_{IL}; // reduce voltage to \( v_{IL} \)
4: \hspace{1em} \text{PR} = \text{PR} \cup \{r\};
5: \hspace{1em} \text{RN} = \Phi; // set of nodes whose voltages are reassigned
6: \hspace{2em} for (\( \text{cstep} = 1; \text{cstep} \leq L, \text{cstep}++ )\} {
7: \hspace{3em} \text{N = get\_match\_nodes} (\text{cstep}, r, v_{Hi});
8: \hspace{3em} // try to scale down the voltage of a node
In general, Algorithm 3 tries to reduce the supply voltage from \(v_{H}\) to \(v_L\) for as many resources as possible (in the loop Level1). In line 3, the voltage of a resource \(r\) is reduced to \(v_L\). This may break the latency constraint \(L\) because the delay of \(r\) is increased due to the lower voltage. In addition, as there is now one more resource with voltage \(v_L\) and one less resource with voltage \(v_{H}\), the resource constraint \(R\) may be violated in some csteps. Therefore, the constraints \(R\) and \(L\) must be checked in every cstep (line 6). The resource constraint might be violated in a situation as illustrated by the example below.

Suppose in Cstep \(A\), all the resources in \(R\) are assigned with \(v_{H}\) and all are used. In Cstep \(B\), a resource \(r \in R\) is reassigned with \(v_L\). This will then cause the violation as now there is one less resource of \(v_{H}\) to be used in Cstep \(A\). This situation is checked by check_resource \((R, cstep, cstep+ latency (n, v_{H} - 1))\) (line 19). One solution to the resource constraint violation is to reduce the voltage of an operation node from \(v_{H}\) to \(v_L\) in the same cstep as well. The function get_match_nodes \((cstep, r, v_{H})\) finds all the nodes \(N\) that match the resource \(r\) in the cstep with voltage \(v_{H}\). Line 9 checks if there exists a node \(n\) whose voltage can be reduced to \(v_L\) without breaking the dependency and latency constraints. The function \(\text{min}_{-}\) voltage \((n)\) returns the minimum voltage of a node \(n\) such that the dependency and latency constraints are still satisfied. If such a node is found, it is reassigned with the voltage \(v_L\) (line 10).

Note that resource constraint violation might also be introduced in the new lifetime \([cstep, cstep + latency(n, v_{H} - 1)]\) of \(n\). This is because in some csteps in the new lifetime of \(n\), all resources of \(v_L\) may have already been used, leaving no resource of \(v_L\) to execute \(n\). This is checked in line 11, where the function latency \((n, v_{H})\) finds the latency of \(n\) under the voltage \(v_{H}\). If the check fails, the voltage of \(n\) is rolled back. Otherwise, line 18 is performed to check the possible resource violation due to the reallocation of \(r\) with \(v_L\). The checking is on the current cstep only. The following csteps will be checked in the subsequent iterations. In this manner, the voltage reduction is performed on as many resources and nodes as possible while maintaining the constraints throughout.

In short, in the core process, a multifrequency multivoltage functionally pipelined datapath is synthesized from the original scheduled DFG. To further reduce the energy, an iterative process is proposed in the next section.

### B. The Iterative Process

The iterative process is based on the simulated annealing technique. It aims to further optimize the energy consumption by searching for alternative schedules of the DFG, hence different stage partitions and multivoltage assignments, leading to better energy consumption. In the process, four types of local transformation are used to iteratively refine the solution. One of the transformations is randomly selected in each iteration of the simulated annealing process. The four transformations are as follows.

1. **i)** Push up: It moves the first cstep of stage, to stage\(_{-1}\).
2. **ii)** Push down: It moves the last cstep of stage, to stage\(_{+1}\).
3. **iii)** Split stage: There are some cases where extra resources are available because the total resource usage decreases after a series of transformations. In these cases, the extra resources are allocated to the stage that can provide the maximum cost reduction and the stage is split into two stages, providing further clock period extension and voltage reduction.
4. **iv)** Reschedule a stage: It reschedules all the operations of a pipelined stage. Working together with the other transformations, it can transform the entire schedule and totally alter the influence of the original schedule on the total energy.

### IV. Experimental Results

In our experiments, we used the FUs in Table I. The energy of the level converters is from [4], i.e., the low-to-high converters consume the energy of 178.1, 139.4, and 53.04 pJ for 3.3–5, 2.4–5, and 2.4–3.3 V conversion, respectively.

#### A. Influence of Resource and Throughput Constraints

In this experiment, four different resource constraints \((A \rightarrow D)\) were used while the throughput \((TP)\) was fixed. The results obtained are shown in Table II. It can be seen that the energy decreases as the resource constraint is relaxed because the loose constraints promote...
better stage partition, clock cycle extension, and hence voltage supply reduction. To study the influence of throughput, the resource constraint ($R$) was fixed and the throughputs were varied. The results are shown in Table III, where $TP(A)$ is the throughput (data arrival interval) for Experiment $A$. The throughput was then relaxed in 50-ns steps for each of the subsequent experiments from $B$ to $E$. It shows that the energy decreases when the throughput is relaxed. The loosen throughput constraints promote the clock period extension and hence voltage supply reduction.

**B. Comparisons**

To find the improvement of energy achieved by MuVoF and to compare with other published works, we chose some typical resources and find the maximum throughput (TP) in nonpipelining single-voltage synthesis (SVSF) under the resource constraints ($R$) using list scheduling. We then used the same $R$ and TP for MuVoF. Table IV shows the results obtained. The allowed frequencies are of multiples of ten, starting from 1/30 ns and the maximum number of frequencies allowed is 5. The run time is obtained with MuVoF implemented in Java and running on a 2.4-GHz Pentium PC with 256 MB of RAM. Averagely, the energy is reduced by 50.0%.

**V. CONCLUSION**

In this paper, MuVoF, a multivoltage multifrequency behavioral synthesis algorithm for functionally pipelined datapath, is proposed. Given a DFG, the resource, and throughput constraints, it synthesizes a low-energy functionally pipelined datapath. MuVoF consists of a core process and an iterative process. The results obtained from experiments show that MuVoF is effective in reducing energy consumption.

**REFERENCES**


