Scheduling with Multiple Supply Voltages in High-Level Synthesis

Jiang Mengfeng

Graduate School of Information, Production and Systems

University of Waseda

A thesis submitted for the degree of

Master of Engineering

Yet to be decided

Abstract

This paper presents a operation scheduling algorithms called IRMVS (Iterative Refinement Multiple Voltage Scheduling) at the behavioral synthesis stage. The goal is to minimize power/energy consumption under both resource and latency constraints when the resources operating at multiple voltages (5V, 3.3V, and 2.4V). The proposed algorithm can be treated as an iteration containing two phases: In Phase I, a Min-Cut based Voltage Assignment is used. It initializes the operations in the lowest voltages and tries to shorten the nodes in each of the critical paths gradually as little power increase as possible. The result of Phase I will be latency-constrained but no considered on resources constraints. In Phase II, ASAP/ALAP Scheduling is performed first and then do the List Scheduling by the priority which is defined as the latency weighted depth of each node. From the result of scheduling, we count the unallocated resources in each control step and make use of this information in the Voltage Assignment phase of the next cycle, thus we will have a heuristic approach to help us to increase the voltages of useful nodes. After the iteration, we will get a solution of scheduling under both latency constraint and resources constraint at last.

Index Terms: Behavior synthesis, latency-constrained scheduling, low-power design, multiple voltage scheduling, resource-constrained scheduling.

Contents

Chapter 1

Introduction

1.1 Background

In recent years, low power design is compelled by the growing classes of personal computing devices such as portable desktops, audio- and video-based multimedia products, and wireless communication systems, . All these devices require not only high-speed computation and complex functionality, but demand low-power consumption. Another driving factor is for integrating more transistors on a single chip. Unless power consumption is dramatically reduced, the resulting heat will limit the feasible packing and performance of VLSI circuits and systems. Consequently, a great deal of current research is motivated by the need for decreased power dissipation while satisfy the increasing computing requirements.

An effective way to reduce power consumption is to lower the supply voltage level of a circuit, since the switching power is proportional to V_{dd}^2. Reducing the supply voltage, however, causes an increase of the circuit delay and reduction of the throughput. In detail, The propagation delay of CMOS is approximately proportional to $V_{dd}/(V_{dd} - V_T)^2$, where V_T is the transistor threshold voltage.

In order to maintain the throughput, a variety of techniques are applied to compensate for the loss of performance,including reduction of threshold voltages, increasing transistor widths, optimizing the device technology for a lower supply voltage, and shortening critical paths in the datapath by means of parallel architectures and pipelining. But now there is another way to do that, which is to use resources operating at multiple supply voltages.

For techniques, power can be optimized at all levels of the design process: system, algorithm (behavior), architecture, logic, circuit, and processing technology [6].Since any design decision made at earlier stages will have more powerful impacts on the final result, researchers believe that the power minimization should be done at higher abstraction level for more significant power saving [7]. So, scheduling with multiple supply voltages is considered.

There are several scheduling algorithms for multiple-voltage resources in the literature in recent years. These algorithms can be classified into I) only latency-constrained (i.e., latency is a hard constraint and resources are minimized) [1], [2], ii) only resource-constrained (i.e., resource is a hard constraint) [3],and iii) latency and resource constrained (i.e., both latency and resource are hard constraints) [4], [5]. While the (only) latency-constrained and (only) resource-constrained algorithms have polynomial or pseudopolynomial time complexity, the latency and resource-constrained algorithms are based on integer linear programming and have (worst case) exponential time complexity. In this paper, we proposed a heuristic algorithm for latency and resource-constrained scheduling that produces comparable results with only polynomial time complexity.

This paper, consequently, focus on the operation scheduling at the behavior level. Operation scheduling with multiple supply voltages in the High-level Synthesis can save a huge amount of power as well as keep the throughput, since it can really take full advantage of available schedule slack and break through the scheduling bottleneck of non-uniform path length. Details are discussed in later sections.

1.2 Organization of the paper

The paper is organized as follows: The concept of High Level Synthesis is explained in Chapter 2. Chapter 3 introduced the problem and the general methods around the key words of scheduling. Chapter 4 explains our proposed algorithm — Iterative Refinement Multiple Voltage Scheduling Algorithm under both resource and lantency constarints. Chapter 5 describes the simulation results and do some analysis on them. Finally, conclusions are mentioned in chapter 6.

Chapter 2

High Level Synthesis

2.1 What and Why

The high-level synthesis process can be defined as the translation of a behavioral description to a structural description. The high-level synthesis process takes a system in the form of a hardware description language (HDL) as input such as high-level languages (e.g., C) or behavioral hardware description languages (e.g., VHDL). The output must contain both datapath for combinational logic and control signals for sequential logic. In general high-level synthesis tools, the generated output is generally in a low-level HDL, such as structural VHDL. Except the corresponding instructions from the source code of input, a great quantity of information about the detail of architecture is generated such as the number of registers and function units, also a set of constraints and objectives must be satisfied during the translation.

Behavioral or high-level synthesis has become popular because of several advantages it provides, as discussed below:

Continuous and reliable design flow: The high-level synthesis process provides a continuous and reliable automatic translation from high-level specifications to RTL description without manual handling.

Shorter design cycle: Automated design process will make production faster. The shorter design cycle can reduce the number of man-hours used and time to market, and hence the overall cost of the chip.

Fewer errors: The easier the synthesis process can be verified, the fewer errors will occur. Correct design decisions at the higher levels of circuit abstraction can avoid the errors propagating to the lower levels, which are too detailed and costly to correct.

Easy and flexible to search the design space: The designer has more flexibility to choose the proper design, since a synthesis system can produce several designs in a short time.

Balanced degree of freedom for power optimization: Power and performance optimization can be performed at any level of circuit abstraction, from system level to silicon. As the level of abstraction goes lower, the complexity of the circuit increases; additionally, the degrees of freedom, and thus power reduction opportunities, decrease.

Documenting the design process: An automated system can track design decisions and their effects. Thus, the third parties can easily do the design debugging and continuation. This will be useful for macrocell-based design and the sale of designs as intellectual property cores, while the fully custom design is very expensive.

Availability of circuit technology to more people: As design expertise is moved into synthesis systems, it becomes easier for a non-expert to produce a chip that meets a given set of specifications. Hence, the designer can be hired at a lower price, which will reduce the non-recurring cost and overall design cost of the chip.

2.2 A Typical HLS Process

High-Level Synthesis contains Various Phases: Compilation, Transformation, Scheduling, Allocation, Binding and Output Generation. A small example is presented to demonstrate the various phases of behavioral synthesis in detail. Suppose that we want to synthesize hardware to perform the operation $y = (a + b) * (c - d)$. The steps can be seen in Fig. 2.1 to Fig. 2.6.

2.2.1 Compilation

The behavior of a system to be synthesized is usually specified at the algorithmic level using a high-level programming language like C/C++ or a hardware description language (HDL) such as VHDL and Verilog. And the result of compilation is always the data flow graphs (DFGs) and control flow graphs (CFGs). The DFG is a directed graph that represents data movement, whereas the CFG is a directed graph that indicates the sequence of operations. The process of compilation is shown in Fig. 2.1.

2.2.2 Transformation

In the transformation step, the initial DFG is transformed so that the resultant DFG is more suitable for scheduling and allocation. These transformations include compiler-like optimizations such as dead-code elimination, common subexpression elimination, loop unrolling, constant propagation and code motion.

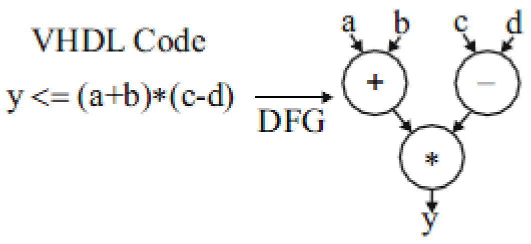


Figure 2.1: Step 1: Compilation and transformation

2.2.3 Scheduling

Scheduling is the process of partitioning the set of arithmetic and logical operations in the DFG into control steps so that the operations in the same control step can be executed concurrently, while taking into consideration possible trade-offs between the total execution cost and hardware cost. In the scheduling step, three important parameters can be determined: the total number of control steps , the minimum number of functional units of each type and the lifetimes of the variables generated during the computation of operations, as is shown in Fig. 2.2.

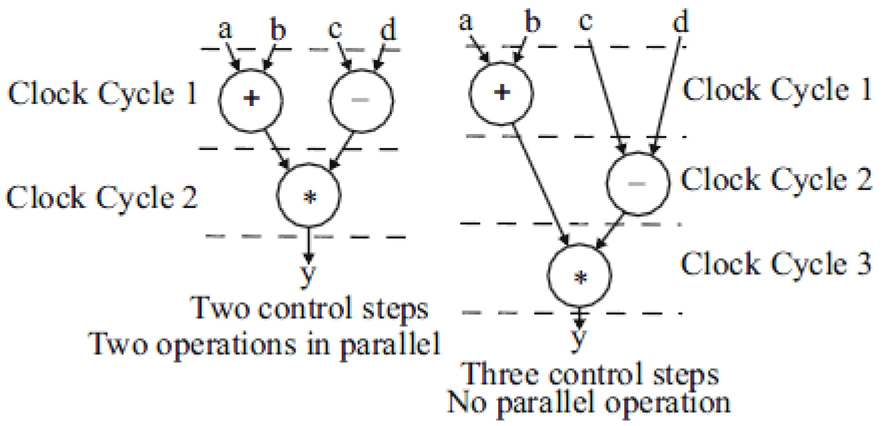


Figure 2.2: Step 2: Scheduling (time or resource constraints)

2.2.4 Allocation

Allocation is the process of choosing resources from the library, which involves tradeoffs according to different features like delay, area, power and leakage. Resource allocation is the process of determining the number of functional units of each type for performing operations, memory units (registers) for storing data values and interconnects for data transportation. Often, the selection and allocation processes are a single task. Allocation is further divided into sub-tasks, such as functional unit allocation, memory unit allocation and interconnect allocation. Resource allocation and binding may share resources so that the same hardware can be used to execute different operations or so that the same register can be used to store more than one variable, as is shown in Fig. 2.3.

2.2.5 Binding

Binding is the process of assigning variables to memory units and data transfers to interconnections. It further determine which resource should be used. It also contains three sub-tasks: functional unit binding, memory unit binding and

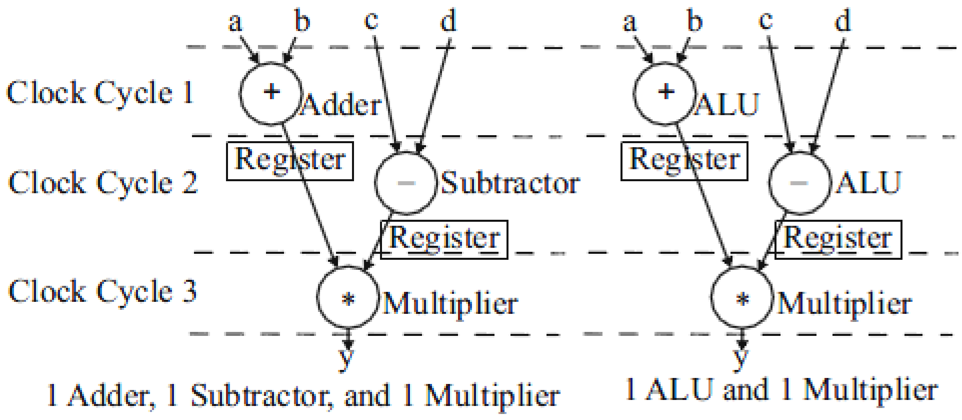


Figure 2.3: Step 3: Allocation (fixed amount and types of resources)

interconnect binding. The first one maps operations onto a set of selected functional units. The second one maps data carriers (constants, variables, arrays) onto storage elements (read-only memories, registers, memory units). The last one maps every data transfer onto a set of interconnection units for data routing, like Fig. 2.4.

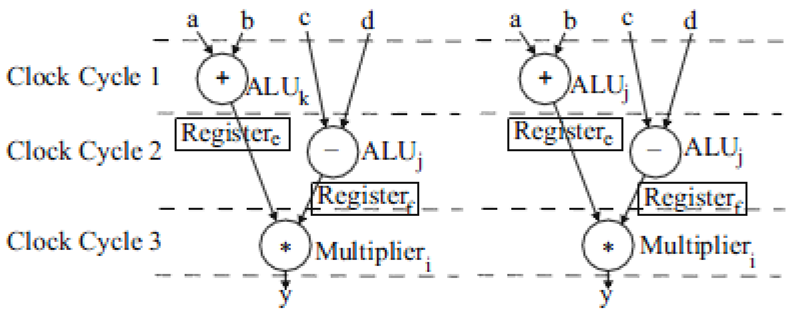


Figure 2.4: Step 4: Binding (which resource will be used by which operation)

2.2.6 Output Generation

Fig.2.5 and Fig.2.6 show the output generation phase. The output should be in a form such that logic-level synthesis tools can optimize the combinational logic and layout synthesis tools can design the chip geometry. The generated output is generally in a low-level HDL, such as structural VHDL.

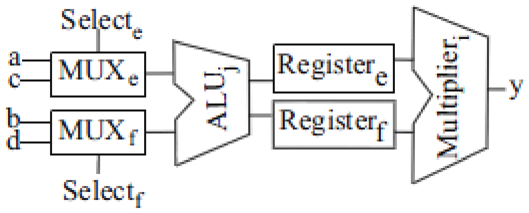
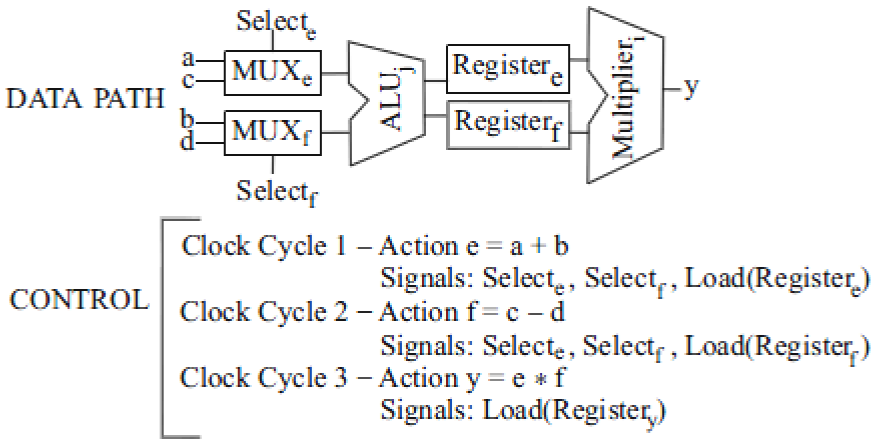


Figure 2.5: Step 5: Connection allocation (communication between resources: bus, buffer, or MUX)



Clock Cycle 1 – Action e = a + b
Signals: Select$_e$, Select$_f$, Load(Register$_e$)
Clock Cycle 2 – Action f = c – d
Signals: Select$_e$, Select$_f$, Load(Register$_f$)
Clock Cycle 3 – Action y = e ∗ f
Signals: Load(Register$_y$)

Figure 2.6: Step 6: Architecture generation (data path and control)

Chapter 3

Scheduling Problem and Method

3.1 What is scheduling problem?

Definition: As introduced in chapter 2 scheduling problem is the assignments of operations to control steps while satisfying data dependency, and it can be seen simply as follows:

- Input:
  - Data Dependency: data-flow graph (DFG)
  - Resource Constraints: # of resources of each operation type can be used in each control steps
- Output: control step for each operation
- Goals: latency, power, leakage, area, etc

The scheduling problem is a non-deterministic polynomial (NP) problem [8]. Behavioral scheduling algorithms may be of various types based on the constraints and optimization schemes, as shown in Fig. 3.1. In this chapter, selected algorithms are discussed generically.

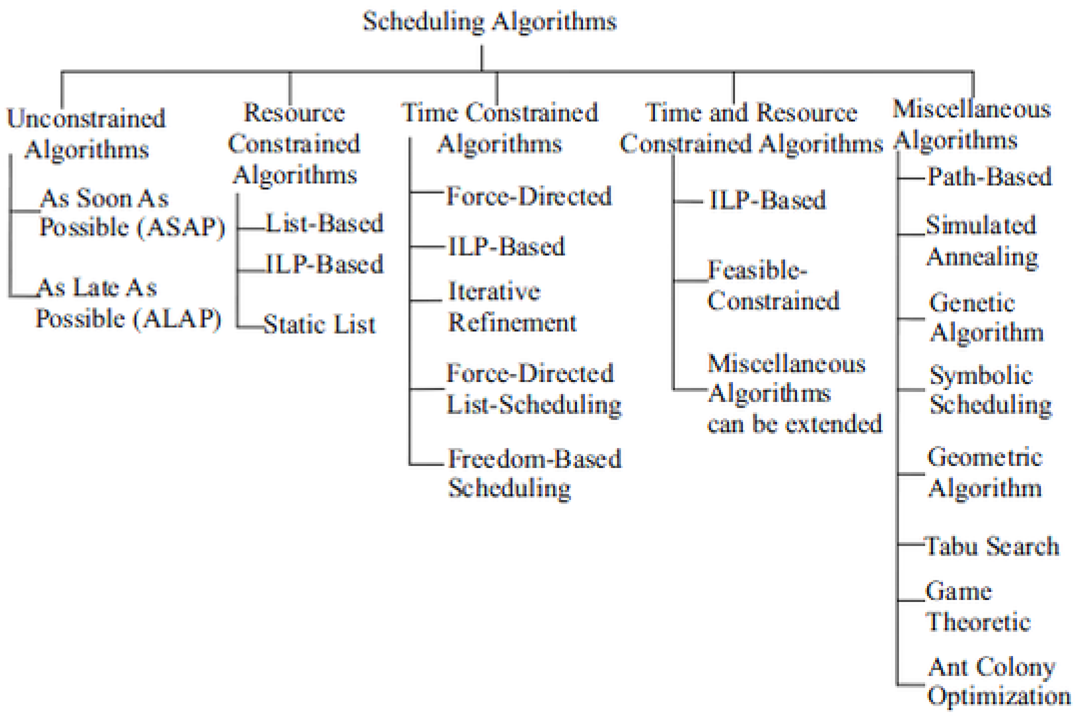


Figure 3.1: Different types of scheduling algorithms

3.2 Basic methods of scheduling

3.2.1 ASAP/ALAP Scheduling and Mobility

As soon as possible(ASAP) scheduling and as late as possible(ALAP) scheduling are the simplest scheduling algorithms. ASAP scheduling places the operations in the sorted topological order by stamping them in the earliest possible control step, while the ALAP scheduling process takes operations in the reverse order and places them in the latest possible control step.

These two algorithms are simple and always the base of most of the advanced scheduling algorithms. The ASAP algorithm is unconstrained, while the ALAP scheduling considers the number of steps resulting from the ASAP schedule as a latency constraint. The time complexity of these algorithms is $O(|V| + |E|)$ [8]. Algorithms 1 and 2, respectively, present the two algorithms.

```
1 For a sequencing DFG $G(V,E)$ time stamp source vertex $v_0$ with $c_0 = 0$;
2 while (Sink vertex $v_N$ is not considered for time stamping) do
3     Select a vertex $v_i$ whose all predecessors are scheduled;
4     Schedule $v_i$ by setting its time stamp as $c_i$ maximum start time of all
      its predecessors plus its delay;
5 end
6 Return schedule of the DFG $\{C_S = c_0, c_1, ..., c_N\}$;
```

Algorithm 1: As Soon As Possible (ASAP) Scheduling

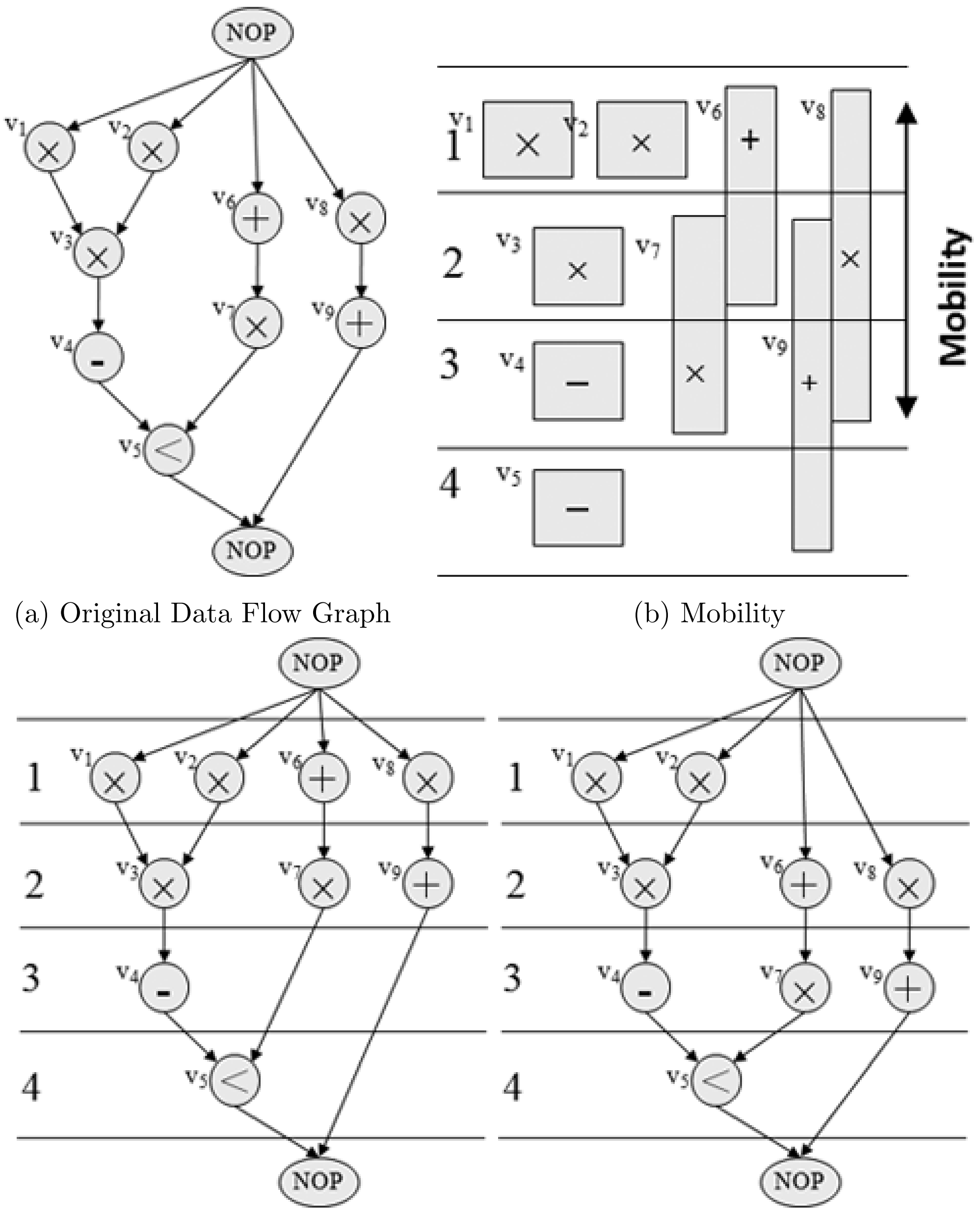
```
1 For a sequencing DFG $G(V,E)$ time stamp sink vertex $v_N$ with $c_N$
  =ASAP Latency Bound+1;
2 while (Source vertex $v_0$ is not considered for time stamping) do
3     Select a vertex $v_i$ whose all successors are scheduled;
4     Schedule $v_i$ by setting its time stamp as $c_i$ minimum start time of all
      its successors minus its delay;
5 end
6 Return schedule of the DFG $\{C_S = c_0, c_1, ..., c_N\}$;
```

Algorithm 2: As Late As Possible (ALAP) Scheduling

As is discussed above, ASAP and ALAP are always useful for other schedule algorithms since they provide lower and upper bounds on possible control steps for scheduling an operation. And the difference between the ALAP and ASAP time stamp is called Mobility. A positive mobility value means the corresponding operation is located in the noncritical path while mobility equal to zero means that the corresponding operation is a critical node in the whole graph. An simple example shows the ASAP/ALAP and mobility in details as in Fig. 3.2.

3.2.2 List Scheduling

This algorithm is usually a representative resource-constrained scheduling algorithm. List scheduling is essentially a heuristic approach to solve the scheduling problem. In list scheduling, the operations available for scheduling are kept in a priority ordered list for each control step. The priority can be defined in many



(a) Original Data Flow Graph (b) Mobility

(c) ASAP Schedule (d) ALAP Schedule

Figure 3.2: ASAP/ALAP Schedule and Mobility

ways: mobility of the operation, the length of path from the operation to the sink, or even a random priority. An operation on the list is scheduled one by one if the resource needed by the operation is free; otherwise, it is deferred to the next clock cycle.

Since the priority definition makes large impacts on the result of list scheduling, a common and effective strategy is to use the latency weighted depth of the node [9]. The depth of a node n is the length (number of nodes) of the longest path in the dpg from n to some leaf (including n and the leaf.) The latency weighted depth is computed the same way, but the nodes along the path are weighted using the latency of the operation the node represents. The following formula summarizes the priority computation for a node n:

$$priority(n) = max\Big(\forall_{l \in leaves(DPG)} \forall_{p \in paths(n,\ldots,l)} \sum_{p_i=n}^{l} latency(p_i)\Big) \tag{3.1}$$

Dynamic programming can be used to compute the priorities efficiently, and we take into consideration the anti-edges described above:

$$priority(n) = \begin{cases} latency(n) & \text{if } n \text{ is a leaf} \\ \max\Big(latency(n) + \max_{(m,n)\in E}(priority(m)), & \\ \max_{(m,n)\in E'}(priority(m))\Big) & \text{otherwise} \end{cases} \tag{3.2}$$

Since its heuristic method, list scheduling is a relatively fast scheduling. Algorithm 3 shows the pseudo-code of resource-constrained list scheduling that minimizes latency [10]. Figure 2.15 shows the scheduled DFG result for a resource constraint of two multipliers and one ALU. The time complexity of this algorithm is $O(|E| + |V|log(|V|) + |V|T_0)$, where $O(|V|T_0)$ is the worst-case time for creating the list and selecting its first element. This algorithm has better time complexity compared to force-directed scheduling and guarantees results [10]. The time complexity of force-directed scheduling is $O(|V|)^3 C_{max}$, where C_{max} is the time constraint expressed in terms of the number of clock cycles [11]. Thus, list scheduling is a better choice for large and practical designs.

```
1  Initial clock cycle $c = 1$ ;
2  while (For a sequencing DFG $G(V, E)$ time stamp sink vertex $v_n$ is not
   considered for time stamping) do
3      for (Resource of type $k$) do
4          Determine list of vertices of type $k$ whose predecessors are already
           scheduled to finish execution in $c, L_{F_{c,k}}$ ;
5          Determine list of vertices of type $k$ that started at an earlier cycle
           but whose execution is not finished in $c, L_{U_{c,k}}$ ;
6          Select the vertices from list $c, L_{F_{c,k}}$ such that the resource needed
           by these vertices and $c, L_{U_{c,k}}$ do not violate resource constraints;
7          Time stamp the above selected vertices at clock cycle $c$, i.e., $c_i = c$ ;
8      end
9      Increment the clock cycle $c$ ;
10 end
11 Return schedule of the DFG $\{C_{LS} = c_0, c_1, ..., c_N\}$ ;
```

Algorithm 3: Resource-Constrained List Scheduling to Minimize Latency

3.3 Scheduling with Multiple Supply Voltage

For a multiple voltage scheduling, the input is still a data flow graph, input data stream, the only difference is that the resources can operate at multiple voltages. It really causes much different features, the most different thing among them is the delay of the operations, and the datapath interconnection becomes more complex since the insertion of level shifters cannot be neglected. According to the all high voltage representation, lower supply voltage must cause an increasing in the delay of the operations. The length of the DFG becomes longer and the mobility and the ASAP/ALAP are also in different. The minimun length of scheduling becomes far away from the critical length since the longer nodes have more possibility to conflict with each other.

But scheduling with multiple voltages also benefits us in many aspects. In the single supply voltage scheduling one cannot always take full advantage of available schedule slack to reduce the voltage. Nonuniform path lengths, a fixed clock period, and a fixed number of control steps can all lead to schedule slack that is not

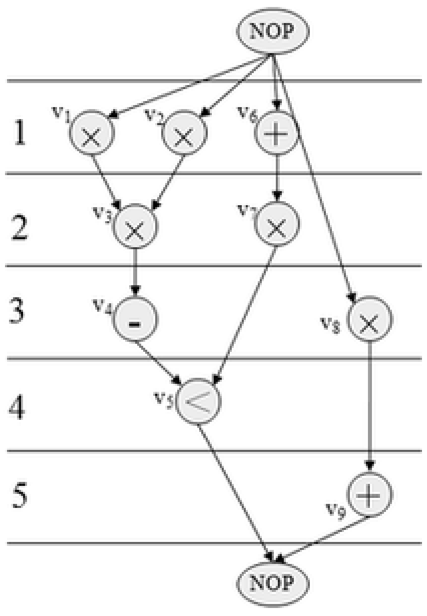


Figure 3.3: Schedule resulting by using a list scheduling algorithm with a resource constraint of two multipliers and one ALU

fully exploited. Fig. 3.4 provides examples of each type of bottleneck. When there are nonuniform path lengths, the critical path determines the minimum supply voltage, even though the shorter path could execute at a still lower voltage and meet timing constraints. When the clock period is a bottleneck, some operations only use part of a clock period. The slack within these clock periods goes to waste. Additional voltages would permit such operations to use the entire clock period. Finally, a fixed number of control steps (resulting from a fixed clock period and latency constraint) may lead to unused clock cycles if the sequence of operations does not match the number of available clock cycles.

3.3.1 Resource-Constrained Multiple Voltage Scheduling

In resource-constrained scheduling, the problem is to schedule the data flow graph, given the resource constraints, such that the number of control cycles is minimum. The resource constraint is specified as follows.

1. The number of computational units for each type is given.

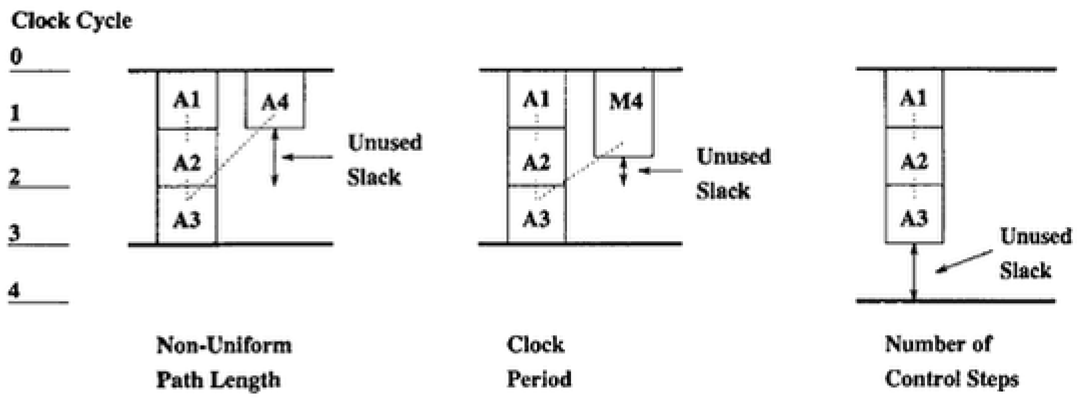


Figure 3.4: Examples of scheduling bottlenecks.

2. The delay of each type of computational unit is also given.

The delay is directly related to the operating voltage. While on the one hand, it is important to minimize the number of control cycles needed to finish a computation because it determines the sample period of the system (and is thus related to power consumption), on the other hand, operating resources with reduced supply voltage reduces the power at the expense of an increase in the number of control cycles. The algorithm presented in this paper tries to balance the conflicting requirements of reducing the number of control cycles and utilizing resources operating at reduced voltage.

3.3.2 Latency-Constrained Multiple Voltage Scheduling

In latency-constrained scheduling, the problem is to assign voltages to the nodes of the DFG, given the timing constraint, such that the power consumption is minimum. Thus, the more nodes that can be assigned to lower voltages, the greater is the power reduction. The statement of assigning voltage to a node is equivalent to assigning voltage to the functional unit onto which the node is mapped.

Chapter 4

Proposed Algorithm - IRMVS

4.1 Preliminaries

4.1.1 Architecture Model Assumption

The architecture model assumed by the algorithm is depicted in Fig. 4.1. All operator outputs have registers. Each operator output feeds only one register at the same voltage as the operator supplying its input. All level shifters, when needed, are performed at operator inputs. In addition, we assume that a level shifter is needed between resource A and resource B only if resource A operates at lower voltage compared to resource B. We do need these assumptions to calculate the number of level shifters during the multiple supply voltage scheduling [7].

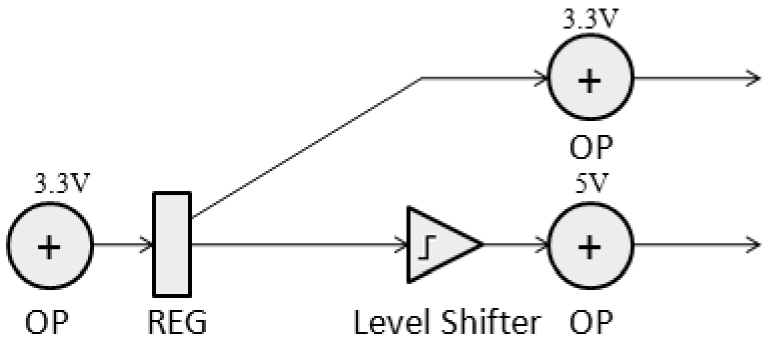


Figure 4.1: Architecture Model Assumption

4.1.2 The Timing and Energy Model

Assume the control steps sharing a basic unit of time in the DFG in behavior level, i.e. the time of the clock cycle is a fixed value, denoted as t_c. Since decreasing the supply voltage level increases the delay of the corresponding operations, an operation may thus become a multi-cycle operation for a given length t_c. For an operation i, let t_i^s be its starting time, a_i the output arrival time, d_i the execution time (delay), t_c the length of the control step, then we have the follows:

$$a_i = t_i^s + d_i$$

$$t_i^s = \max_{(j,i)} \lceil a_j / t_c \rceil * t_c$$

where operation j is a predecessor of operation i.

Table 4.1: Delay Model of a 16-bit operation

Delay(ns)	1.5	2.4	3.3	5
mul16	717	287	175	100
add16	143	57	35.1	20
sub16	143	57	35.1	20

Assume that the dynamic energy dissipation in a functional unit is given by this equation:

$$E_{FU_i} = F_i(\alpha_{i,1}, \alpha_{i,2}) * V_i^2 \tag{4.1}$$

where V_i is the supply voltage of functional unit FU_i and $\alpha_{i,1}, \alpha_{i,2}$ are the average switching activities on the first and second input operations of FU_i, respectively, F_i is a function of $\alpha_{i,1}$ and $\alpha_{i,2}$ and in general may be nonlinear.

[8] provides two methods to calculate E_{FU_i}/V_i^2 for given pair $\alpha_{i,1}$ and $\alpha_{i,2}$, one is based on look-up table and the other is based on energy macro-modeling using a linear equation with random $\alpha_{i,1}$ and $\alpha_{i,2}$. Here we only use their conclusion data. Detail is in shown at Table 4.1 and Table 4.2.

Table 4.2: Energy Model of a 16-bit operation

Energy(pJ)	1.5	2.4	3.3	5
mul16	225	577	1090	2504
add16	10.6	27	51.4	118
sub16	10.6	27	51.4	118

4.1.3 Minimum Schedule Length Estimation

We can easily find the minimum length in an unconstrained scheduling, which is equal to the length of critical path (LOCP) in the DFG. But if we take the resource constraints into account, the problem suddenly becomes much difficult. In fact the problem of minimizing latency under resource constraints or minimizing resources under latency constraints is intractable [8].

Although it is NP-complete, during iterations we can also use several observations to develop a lower bound on the minimum schedule length. Very fortunately, in the voltage assignments the lower bound on the minimum schedule length can be updated very conveniently since it really change the critical paths.

First let us see the single voltage situation, denote the LOCP is cpl(G). Obviously it is a minimum lower bound for the schedule length since any schedule must ensure all dependences in the DFG. Next, we will find all nodes on all the critical paths. And can easily count the critical nodes by control steps, which means that we get the number of the critical operations in each control step. Finding the maximum one and denote it as N.

This is one kind of information in the horizontal view. While the vertical is related to the timing constraints, the horizontal view affects the satisfaction of the resource constraints. If the resource constraints of type i is p_i, the set of the type of operations is T. Then the following equation [9] must hold:

$$\text{minlength(G)} = \max_{i \in T}(cpl(G) + \lceil N/p_i \rceil - 1, N/p_i) \quad (4.2)$$

This equation, which refreshes the lower bound of the scheduling length gradually, also holds for multiple voltage scheduling. Not only that, it can be refreshed iteratively and is a powerful control factor in the heuristic scheduling methods.

4.1.4 Latency Factor

Since it's hard to judge whether there exists a feasible solution to the given resource constraints and latency constraints, we use a relative factor for the latency constraints instead of absolute value.

At first we try to find a way to make this factor working by choosing the critical path as the base line, but soon failed. It's still unable to judge whether the problem has a feasible solution. An extreme example is like this: We have infinite nodes in a DFG, each of which can finish its operation in one control step, therefore the length of critical path is 1. Assume a finite value N is set for the latency factor, and the maximum length of N*1 must hold. But it's impossible since there are infinite nodes at all.

So another value is chosen for substitute, which is the lower bound of the minimum schedule length. Although the lower bound is non-fixed and growing in each cycle of the iteration, we can still use the value to evaluate the quality of our final schedule length, since it is a lower bound of that. Still the extreme example: this time the algorithm can easily find the lower bound of schedule length is infinite according to the Equation (4.2) and stop itself to avoid the dead loop.

4.2 Iterative Refinement Multiple Voltage Scheduling Algorithm

4.2.1 Algorithm Overview

Since the reason that it is not easy to find a constructive scheduling with MSV under both timing constraints and resource constraints, we want to find an iterative way to solve the problem. We call it Iterative Refinement Multiple Voltage Scheduling (IRMVS). The generated solution will be exactly resource constrained. But for the latency constraints, a shrink factor is used to manually guarantee the result not too far away from the optimal scheduling length.

The proposed algorithm contains two main phases: The first is the Algorithm Voltage Assignment (VltAss) and the other is Resource Constrained Scheduling (List Scheduling). The former algorithm initializes all the operations in the lowest voltage and shortens all the critical paths in the DFG as little power increased as possible. The result of Phase I is still a Data flow Graph with each node operating at its own supply voltage, and the critical path of new graph is strictly latency constrained while the power reduction is a optimal solution. But this is not our demanding solution. The latter algorithm which is a resource-constrained scheduling take the output graph from the Phase I as its own input and try to scheduling the operations as shorter as the latency grows. But only execute one time is always not enough. Phase II must feedback some information for Phase I in the next loop circle. In our algorithm, this information is the unallocated resources in each control step. The detail about the data format and the reason why we select this is discussed in later sections.

4.2.2 Algorithm VltAss

In this part of the whole algorithm, a voltage assignment can be generated by a min cut based voltage assignment algorithm [12]. It initializes the operations in the lowest voltages and tries to shorten the nodes in each of the critical paths gradually as little power increase as possible.

4.2.2.1 Calculate the global sensitivity

A control value called global sensitivity is used to control the shorten process. It's defined as $\Delta power/\Big(max_{i\in p}length(p) - max_{i\in p}length'(p)\Big)$ [12]. But in order to reuse the information feed back by the second phase Resource Constrained Scheduling, in our algorithm a Resource and Latency related factor (RL Factor) is involved into the original definition of global sensitivity and is divided by the original equation. So the new definition is like this:

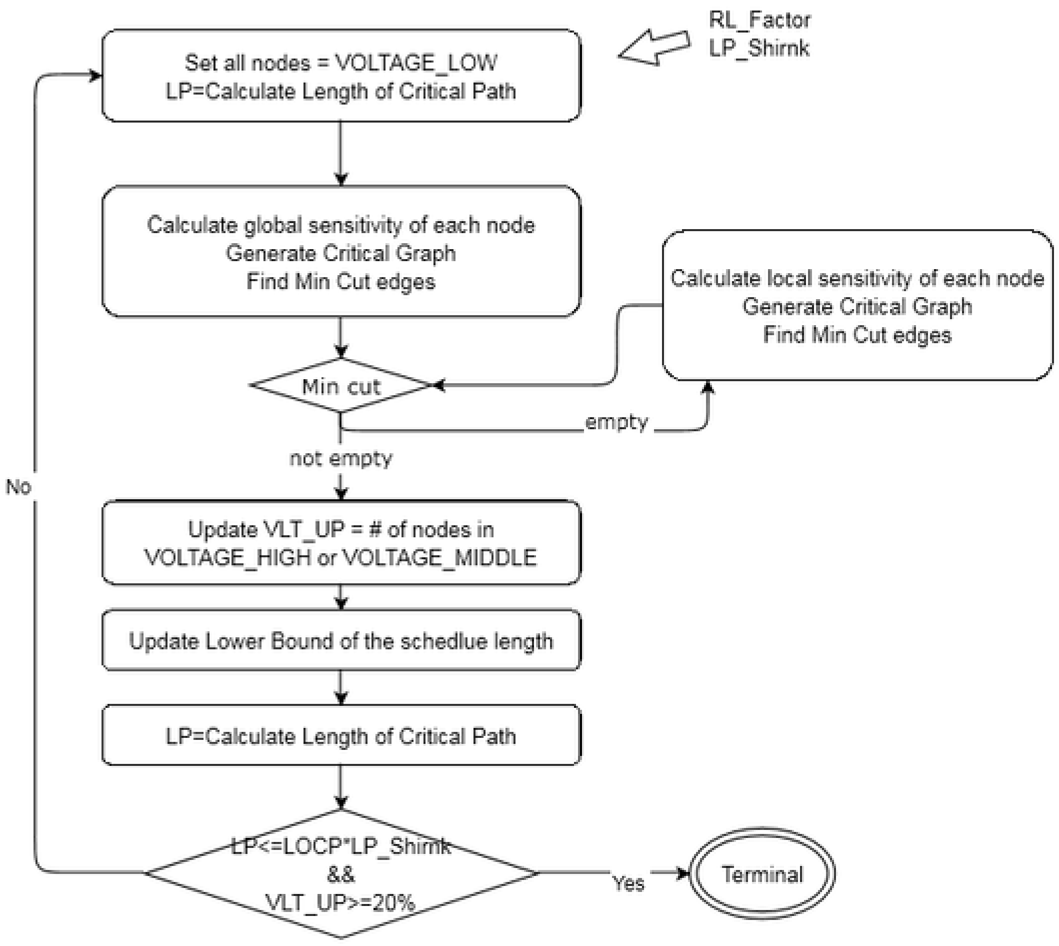

RL_Factor
LP_Shirnk
Set all nodes = VOLTAGE_LOW
LP=Calculate Length of Critical Path
Calculate global sensitivity of each node
Generate Critical Graph
Find Min Cut edges
Calculate local sensitivity of each node
Generate Critical Graph
Find Min Cut edges
Min cut
empty
not empty
No
Update VLT_UP = # of nodes in VOLTAGE_HIGH or VOLTAGE_MIDDLE
Update Lower Bound of the schedlue length
LP=Calculate Length of Critical Path
LP<=LOCP*LP_Shirnk && VLT_UP>=20%
Yes
Terminal

Figure 4.2: Chart Flow of Algorithm VltAss

$$GlobalSensitivity(i) = \frac{\Delta power}{\Big(\max_{i \in p} length(p) - \max_{i \in p} length'(p)\Big) * RLFactor(i)} \tag{4.3}$$

where i is a critical node

The key point of Algorithm VltAss is the decreasing of the LOCP. Global Sensitivity is used to choose what nodes should be upgraded. Assume RL Factor(i)=1 in this stage, and the reason of the definition is very simple to understand. We wonder LOCP decrement as much as possible and power increment as little as possible. So these two values is just set for numerator and denominator. Find the nodes with minimum global sensitivity , upgrading them and then the algorithm will enter another loop circle. The detailed calculation of Global sensitivity is shown in Algorithm 4.

```
   Input: A critical operation $v_i$
   Output: The global sensitivity ($\Delta power/\Delta length$) of $v_i$
1  $item2 = d(i) - d'(i)$ ;
2  $item1 = (v_i$ is primary input ?0 : $\infty)$ ;
3  $item3 = (v_i$ is primary output?0 : $\infty)$ ;
4  forall the fanin $v_j$ of $v_i$ do
5      $\Delta d(j,i) = (v_s^i == v_s^j ?-$ lc delay: 0) ;
6      $item1 = \min\{item1, s(j) + \Delta d(j,i)\}$ ;
7  end
8  forall the fanout $v_z$ of $v_i$ do
9      $\Delta d(i,z) = (v_s^i$ is one level below $v_s^z$ ? lc delay : 0) ;
10     $item3 = \min\{item3, s(z) + \Delta d(i,z)\}$ ;
11 end
12 if $item1 + item2 + item3 \leq 0$ then
13     return $\infty$ ;
14 end
15 else
16     return $\Delta$ power/ (item1 + item2 + item3) ;
17 end
```

Algorithm 4: calculate the global sensitivity

4.2.2.2 Min Cut of Generated Critical Graph

Since the scheduling solutions are strong relative to the non-critical path, researchers should focus on all the critical paths to avoid the situation that upgrades a node's voltage but no LOCP decrease. The way to do this is to upgrade a group of nodes simultaneously. A special way based on Min Cut of Generated Critical Graph is introduced here.

First the critical nodes on the current critical paths are selected, and a subgraph including only those nodes can be generated. And then take a node-split technique to using edge weight instead of the node weight which is the delay of the corresponding operation. Set the other edges' weight to infinite, so only new edges between the nodes of same operation have finite weight, each of which can be mapped onto a operation node in the original DAG. Fig. 4.3 is an example help to illustrate.

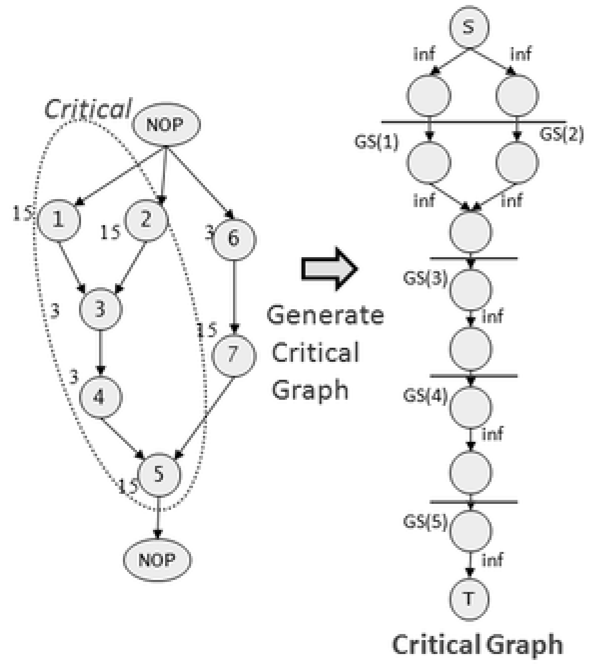


Figure 4.3: Critical Graph and Cuts on it

A cut in a graph G is a subset of all the nodes that is neither empty nor the complete set of nodes. The weight of a cut is the sum of the weights of the edges having exactly one end node in the cut. A minimum cut is a cut of minimum weight. Here the only useful information to us is the cut edges, neither the cut nodes nor the cut value is insignificant in our algorithm, but we still can use some popular min cut algorithms from Flow Network Problems. In the proposed algorithm, a $O(|V||E| + |V|^2 log|V|)$ calculation of Min Cut() from LEDA is used. In Fig. 4.3, the min cut edges will be decided by the minimum value of $\{GS(1) + GS(2) = 172, GS(3) = 160, GS(4) = 93, GS(5) = 108\}$. So node 4 is chosen to upgrade. The upgrade sequence will be $4, 3, 5, \{2, 1\}, 6, 6, 7$, and the assignment is {M,M,M,M,M,H,M}.

4.2.2.3 Mechanism to jump out of TRAP

The delay of critical gate is decreased if its supply voltage is upgraded, but it may result to the insertion of lever converts, so the length of critical paths may not be shorten, so TRAP generates. To jump of TRAP, sensitivity of critical gate is relaxed without considering the possibly inserted level converter as follows:

$$LocalSensitivity(i) = \frac{\Delta power}{\Delta delay(i) * RLFactor(i)} \tag{4.4}$$

4.2.3 Constrained Scheduling — List Scheduling

As is described in section 3.2.2 List Scheduling is a representative resource-constrained scheduling algorithm. We apply the algorithm here with the priority defined as the latency weighted depth of each node a combination of this part, here we only want to discuss the feedback information.

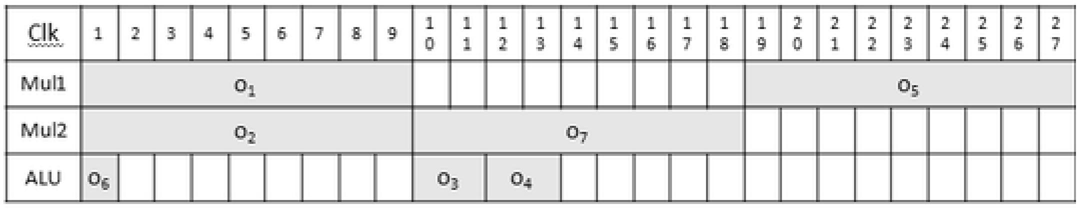
4.2.3.1 RL Factor — feedback to Algorithm VltAss

If we use result in the Algorithm VltAss for the example in Fig 4.3 and apply List Scheduling on them. A schedule result of length 28 will be generated, as shown in Fig.4.4. Then the problem is very simple. Which nodes should be made shorter and which nodes may become longer in the next voltage assignment? This decision is made in the Phase I but here we can now answer it manually. Those nodes placed in a congested position along the direction of control step grows. It should be made shorter to decrease LOCP under resource constraints. For instance here we wonder to shorten o_5 so as to shorten the schedule length. In order to describe the definition simply and clearly, we denote Sch(i,j) if o_i is scheduled to control step t_j. Then a RL Factor can be calculated as follows:

RL Factor(i) $= 1 + \sum_{\forall_j Sch(i,j)}$ (the number of unallocated resources of j). (4.5)

RL Factor represents the how "critical" a node keep under resource constraints. As the longer the operation is, the possibility it can be accumulated will be larger. Only when all the control steps for an operation placing with all the available resources exhaust, the value of RL Factor(i) become zero and meanwhile shortening the node becomes insignificance to help to shorten the length of schedule.

Since RL Factor is in the denominator of the calculation of the Global Sensitivity. High value of RL Factor will increase the possibility to upgrade. It can help us to increase the voltages of useful nodes and better the last solution for the approach.



Clk	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Mul1	o_1																		o_5								
Mul2	o_2									o_7																	
ALU	o_6									o_3		o_4															

Figure 4.4: An example to illustrate the use of RL Factor

4.2.4 Algorithm Analysis

The whole view of the algorithm structure is as shown in Fig 4.5.

The time complexity is determined by both the Algorithm VltAss (Phase I) and List Scheduling (Phase II), and the iteration numbers $nIter$. The upper bound of Algorithm VltAss is $O(kn^3\sqrt{m+n})$ [12], but in practice it is only $O(m+n)$; The List Scheduling takes also $O(m+n)$ and the RL Factors updating is $O(lgn)$,which runs when the List Scheduling finishes. From those above , the time complexity is $O(k(m+n))$ for which k is the iteration number.

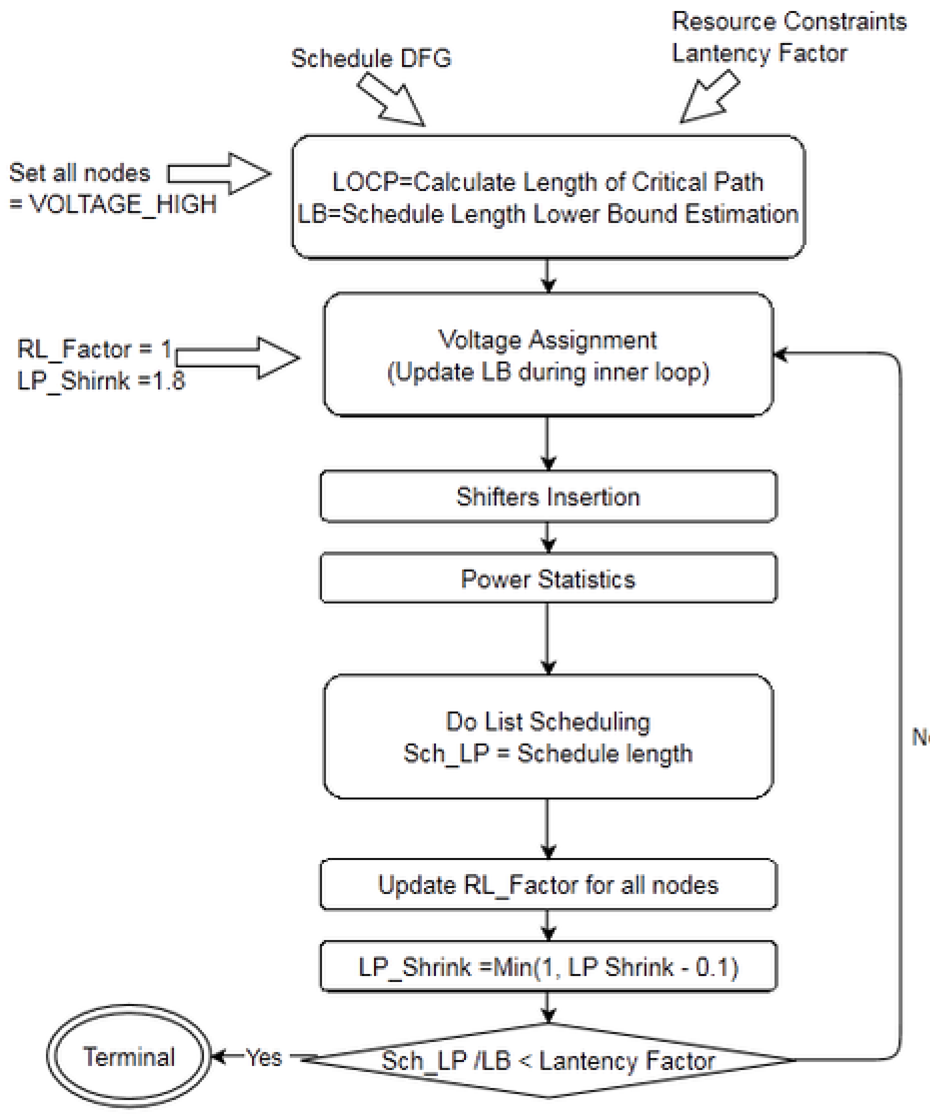

Schedule DFG
Resource Constraints
Lantency Factor
Set all nodes = VOLTAGE_HIGH
LOCP=Calculate Length of Critical Path
LB=Schedule Length Lower Bound Estimation
RL_Factor = 1
LP_Shirnk =1.8
Voltage Assignment
(Update LB during inner loop)
Shifters Insertion
Power Statistics
Do List Scheduling
Sch_LP = Schedule length
No
Update RL_Factor for all nodes
LP_Shrink =Min(1, LP Shrink - 0.1)
Terminal
Yes
Sch_LP /LB < Lantency Factor

Figure 4.5: Chart Flow of Algorithm IRMVS

Chapter 5

Experimental Results

5.1 Experimental Environment

The proposed algorithm IRMVS has been implemented in C on a Linux work station with 2.66 GHz CPU and 16 GB memory.

5.2 Benchmark Suites and Resource Constraints

Table 5.1: Benchmarks of DEGs used in experiments

Benchmark	Description	#Node	#Edge	T_{cri}
ad	Audio decoding	46	113	111
ae	Audio encoding	53	141	145
ar	AR Lattice filter	28	40	21
diff2	Differential equation	11	16	13
ellip	Fifth-order elliptical filter	34	65	27
fft	Fast fourier transform	129	240	29
MPEG	Motion compensation of MPEG decoding	53	112	46
rand0	Random DFG	91	134	33
rand1	Random DFG	631	941	52

The benchmark DFGs that are used in the experiments are obtained from the Internet and generated by TGFF tool [13]. It covers Motion compensation of MPEG decoding, ADPCM Audio decoding/encoding, FFT, some typical DSP filters, and two random DFGs. Table 5.1 shows these benchmarks of input data and the Critical path is calculated by the highest voltage. Table 5.2 shows the resource constraints and delay in different voltages. Also the delay of a level shifter can be normalized to 0.05 cycles and it isn't neglected in our calculation.

Table 5.2: Resource Constraints Data of Experiments

Operation type	Resource Type	Clks	#Units
*	MUL	5,9,15	2
+	ALU	1,2,3	1
−	ALU	1,2,3	1
&	ALU	1,2,3	1
<	ALU	1,2,3	1
<<	ALU	1,2,3	1

5.3 Energy Reduction and Running Time

Table 5.3 is using in the power dissipation (in pJ) in a 16-bit level shifter per voltage transition and Table 5.4 gives the energy for operations. The estimatee method is given in section 4.1.2. The delay is normalized for easy to deal with control steps, Multiplier occupies 5,9,15 cycles during different voltages while ALU occupies 1,2,3 cycles instead.

Table 5.5 lists experiment results of our algorithm, where the average power saving is 22.5% under the time constraint T_{cons}=1.5T_{LB} while the power saving can be 46.21% when T_{cons}=2T_{LB}.

Table 5.3: Energy of a 16-bit Level Shifter

$x/y(pJ)$	2.4V	3.3V	5V
2.4V	0	64.0	128.0
3.3V	49.6	0	142.4
5V	88.0	104.0	0

Table 5.4: Energy of a 16-bit operation

Energy(pJ)	2.4V	3.3V	5V
MUL16	577	1090	2504
ALU16	27	51.4	118

Table 5.5: Energy Reduction and Running Time for Benchmarks

benchmark	T_{cons}=1.5T_{LB}				T_{cons}=2T_{LB}			
	$E_{alg}(pJ)$	$redn\%$	$\frac{E_{LS}}{E_{alg}}\%$	time	$E_{alg}(pJ)$	$redn\%$	$\frac{E_{LS}}{E_{alg}}\%$	time
ad	72350.8	26.5	4.48	0.29	39656	59.73	4.66	0.04
ae	73231.6	32.7	4.26	0.28	43842.6	59.72	3.86	0.04
ar	33846	18.4	0.76	0.15	22309.6	46.22	3.13	0.03
diff2	12911.4	17.3	3.09	0.07	9864.4	36.82	8.37	0.04
ellip	19365.2	16.2	10.25	0.13	11373.2	50.77	11.38	0.03
fft	74320.6	9.4	7.92	0.66	51292	37.47	12	0.19
MPEG	17523.4	30.9	16.24	0.4	14574.4	42.49	18.02	0.13
rand0	39101.2	23.8	10.44	0.32	31203.8	39.17	12.29	0.15
rand1	238642	24.9	12.35	44.44	179619.4	43.49	15.5	2.13
average	-	22.5	8.214	5.19	-	46.21	9.91	0.31

Chapter 6

Conclusions

In this paper, we proposed a new multiple voltage schedule method which is called Iterative Refinement Multiple Voltage Scheduling (IRMVS), based on voltage reassignment by min cut calculation. The proposed algorithm can generate a multiple voltage schedule for each slot under given resource constraints and latency constraint factors. It also give a renewer lower bound of the length of Multiple Voltage Schedling to estimate the length of output. The time complexity is $O(k(m+n))$, for which k is the iteration number. From the experiment results, the average power saving is 22.5% under the time constraint T_{cons}=1.5T_{LB} while the power saving can be 46.21% when T_{cons}=2T_{LB}.

Acknowledgements

This thesis would not have been possible without the support, hard work and endless efforts of a large number of individuals and institutions. I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

At the very first, I am honored to express my deepest gratitude to my dedicated supervisor, Professor Yoshimura, with whose constant encouragement and patient guidance I could have worked out this thesis. He has offered me many valuable ideas, suggestions, and his profound knowledge, rich experience and serious attitude towards scientific research will inspire me all my life.

Secondly, I would like to extend my deepest appreciation to Dr. Chen Song for his great encouragement and invaluable suggestions. I cannot make this thesis without his valuable suggestions that improved the quality of this thesis.

Next, I wish to extend my thanks to all the friends in our laboratory, for their support of my research.

What's more, my special appreciation goes to my parents, Mr. Hua Jiang and Ms. Qingyun Meng, who always kept me away from family responsibilities and encouraged me to concentrate on my study. Without their encouragement and support, this thesis would not have been completed.

Finally, I would like to acknowledge the girl who is always believing in me and giving me the confidence to do anything. Thank you.

References

[1] S. Raje and M. Sarrafzadeh, "Scheduling with multiple voltages," *Integr.VLSI J., pp. 37-60,*, 1997

[2] Jui-Ming Chang and Massoud Pedram, "Energy Minimization Using Multiple Supply Voltages," *IEEE Trans. VLSI Syst, vol. 5*, 1997

[3] W.-T. Shiue and C. Chakrabarti, "Low power scheduling with resources operating at multiple voltages," *IEEE Trans. Circuits Syst. II, vol. 47, pp. 536-543,*, 2000.

[4] M.C. Johnson and K. Roy, "Datapath Scheduling with Multiple Supply Voltages and Level Converters," *ACM Trans. Design Automation Electron. Syst, vol. 2, no 3, pp. 227-248*, 1997.

[5] Y.-R. Lin, C.-T. Hwang, and A. C.-H. Wu, "Scheduling techniques for variable voltage low power design," *ACM Trans. Design Automation Electronic Syst., pp. 81-97,*, 1997

[6] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE, vol. 83, pp. 498-523*, 1995

[7] O.S. Unsal and I. Koren, "System-level power-aware design techniques in real-time systems," *Proceedings of the IEEE, vo1. 91, pp. 1055-1069*, 2003

[8] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., "Introduction to Algorithms, second edition," *The MIT Press*, 2001

[9] Keith D. Cooper, Philip J. Schielke, and Devika Subramanian, "An Experimental Evaluation of List Scheduling," *Rice Computer Science Techcnical Report 98-326*, 1998

[10] Gerez, S.H., "Algorithms for VLSI Design Automation," *Wiley*, 2004.

[11] Paulin, P.G., Knight, J.P., "Force directed scheduling for the behavioral synthesis of ASIC," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 8(6),661-679*, 1989.

[12] Tao Lin, Sheqin Dong, Song Chen, Yuchun Ma, Ou He, and Satoshi Goto, "Novel and Efficient Min Cut based Voltage Assignment in Gate Level," *Quality Electronic Design (ISQED), 2011 12th International pp. 1-6* , 2011.

[13] D. Rhodes, R. Dick, and K. Vallerio., gTgff: Task graph for free,hhttp://ziyang.eecs.umich.edu/dickrp/tgff/, 2008.