國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

具複雜運算單元之低功率多執行緒資料路徑
的研究與設計

Study on Improving Utilization for Low-Power Multithreaded Datapath with Composite Functional Units

研究生： 卓毅

指導教授： 劉志尉 博士

中華民國九十六年十月

具複雜運算單元之低功率多執行緒資料路徑的研究與設計

Study on Improving Utilization for Low-Power Multithreaded Datapath with Composite Functional Units

研究生：卓毅 Student: Yi Cho

指導教授：劉志尉 博士 Advisor: Dr. Chih-Wei Liu

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
in

Electronics Engineering

October 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年十月

具複雜運算單元之低功率多執行緒資料路徑的研究與設計

研究生：卓　毅　　　　　　　指導教授：劉志尉 博士

國立交通大學
電子工程學系 電子研究所

摘要

在觀察近年來處理器的發展演變中我們發現，簡化指令集處理器(RISC)已成為一大設計主流。其簡單和規律的指令集設計很容易進一步的將指令執行管線化(pipeline)提高處理器效能。然而，因為分派一個指令，只能執行一個動作導致其硬體使用率不高。多指令分發(multi-issue)處理器，即超長指令(VLIW)處理器，利用指令層級平行度(ILP)提高硬體使用率，但它的暫存器檔案面積，隨著運算單元增加而劇烈成長，因而付出沉重的硬體代價。在本論文中，我們提出一個具複雜運算單元(composite FU)的資料路徑，以客製化順序串接多個運算單元的方式，在同一指令中處理連續多個基本運算(primitive operations)，達到硬體使用率的提升。此複雜運算單元不僅可以減輕如 VLIW 的暫存器面積會因功能單元(FU)增加而大幅成長的問題，還因為複雜運算單元可以在抓取運算子後，作多個運算才存回，總暫存器存取次數得到節省，進而得到低功率的好處。此外我們也利用整合管線化設計流程來提升整體效能(操作頻率)，以及搭配交錯多執行緒(interleaved multithreaded)架構來完全地隱藏管線化後所衍生的指令延遲。我們同時提出一個自動化複雜運算單元產生器，藉由分析使用者所輸入的應用程式資料流程圖(data-flow graph)，自動產生出一個最佳化的複雜運算單元。經由對多個典型 DSP 應用分析，複雜運算單元 MSA(串接一個乘法器 M 以及一個移位器 S 和加法器 A)的硬體使用率(operation per cycle)和簡化指令集處理器的 1.00 比較提升為 1.35。使用台積電 0.13um 製程作合成分析，在同樣的運算效能下，複雜運算單元較簡化指令集合的面積約多 10%，但較超長指令減少約 50%。複雜運算單元之功率消耗，較簡化指令集合及超長指令節省 16.6%到 31.6%。

Study on Improving Utilization for Low-Power Multithreaded Datapath with Composite Functional Units

Student: Yi Cho Advisor: Dr. Chih-Wei Liu

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

From the observation of evolution of processor development in recent years, we find that Reduced Instruction Set Computer (RISC) processors have already become main design fashion. The simplicity and regularity of RISC is suitable for pipeline design to boost performance. However, its hardware utilization is low because of it execute only one operation in single instruction issued. Multi-issue (VLIW) processors, takes advantage of the Instruction Level Parallelism (ILP) to promote hardware utilization. But the register file (RF) area of VLIW grows exaggeratedly with the increase of the functional unit number. It pays a great hardware overhead. In this thesis, we propose a datapath with composite functional units (FUs). It cascades several functional units in costumed order to perform continuous multiple primitive operations in single cycle for raising hardware utilization. The read and write port number of the register files of composite FUs only slightly increase by 1 or remain unchanged. It solves the problem of large RF area pressure. In addition, the composite FUs can perform several operations after fetching operands and then write back. The reduction of total register accesses leads to low-power benefit. Besides, the pipeline design is integrated to boost performance up and the Interleaved Multithreaded (IMT) architecture is coordinated to hide instruction latency derived from pipeline design totally. In the mean time, we propose a recursive composite FUs generator which automatically generator a best composite FU by analyzing Data Flow Graph (DFG) input by user. From the analysis of several classic DSP kernels, the hardware utilization of MSA-ordered (cascade a multiplier, a shifter, then an adder) composite FU is 1.35 times higher than 1.00 of RISC. Use the TSMC 0.13um process to do synthesis analysis. Under same performance, the register file area of composite FU is 10% more than RISC and 50% less than VLIW. The power reduction of composite FU is smaller compared with RISC and VLIW ranging from 16.6% to 31.6%.

誌 謝

研究生兩年多的時光轉眼即逝，感謝許多人幫助並鼓勵我完成碩士學業。

首先感謝劉志尉老師。老師的豐富學養及學者風範，不但在專業知識及研究態度上給予提點，也在生活和與人相處的應答給予意見和支持，感謝老師兩年來的指導和關照。

感謝口試委員：周景揚教授，蔡淳仁教授及許騰尹教授。謝謝你們在百忙之中，撥冗參與論文口試，並給予寶貴的意見，讓此篇論文更加完備充實。

感謝林泰吉學長不厭其煩地對我的研究工作前瞻指引，並培養研究態度及應有的能力。以及歐士豪學長給我諸多實現細節上的解惑，還有鄧翔升同學和林彥呈、呂進德兩位學弟對我的研究提出意見和討論，感謝諸位的協助。

感謝其餘所有實驗室成員們。

感謝張彥中、林佑昆、郭羽庭、陳信凱、林禮圳學長以及廖彥欽學姊，在我研究生生涯中的提攜。以及顏于凱、洪正堉、李岳泰、張巍瀚幾位學弟們在研究工作上的幫助。

感謝一起打拼、面對挑戰的同學們：卓志宏、劉士賢、陳慶至、王炳勛。

最後，感謝我的家人。爺爺、爸、媽、妹，感謝你們一路上的支持、體諒及鼓勵。

謹將此篇論文獻給所有曾支持我、協助我的人，衷心的感謝並祝福你們。

卓毅
謹誌於 新竹
2007 秋

Contents

List of Tables

List of Figures

1 Introduction

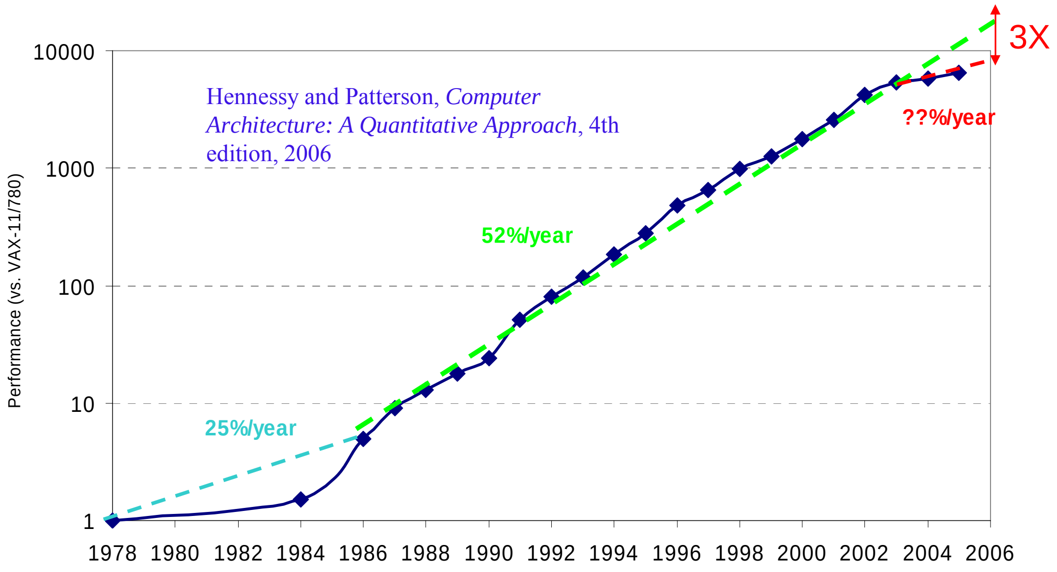
As the desire to the performance for multimedia application growing up day by day, lots of processor design principle showed up for different purposes. We will illustrate the history of processor progress first in section 1-1, and discuss the reasons for the evolution.

Let us focus on the advantages and disadvantages of the datapaths including functional unit architectures. Some decisions and changes are made according to the hardware utilization and the area pressure of register files, or even some power issue.

We propose composite FU to overcome the weakness of RISC and VLIW. And some contributions are described in section 1-2.

Section 1-3 describes the thesis organization.

1.1 History of Processor Progress



Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, 2006

- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

Figure 1-1 Uniprocessor Performance

[1] Figure 1-1 shows the uniprocessor performance measured by SPECint.

◆ **Complex Instruction Set Computer (CISC)**

From 1978 to 1986, the CISC-style processors dominate the processor market. The essence of CISC is to allocate as many hardware as the functions need. Therefore, the CISC processors can perform some specific functions at high speed.

But CISC processors have some well-known drawbacks.

Instructions of CISC processors have very different execution cycles. Single instruction may consume cycles from several to thousands corresponding to its functionality and complexity. The complexity of instruction variation and hardware selection lead to the inefficiency of the compiler. Besides, If pipeline technique is coordinated to boost the performance of CISC processors, it would be hard to pipeline, and the improvement of performance is limited.

In addition, all CISC-style processors suffer a serious problem. Some hardware is idle at most time. DEC-PDP 10 is a famous CISC processor, and some surveys [2] of this processor tell us that 70 instructions account for 99% of operation and 50 instructions account for 95% of operation. Lots of instructions and hardware are rarely used, and the idle hardware implies the waste of power consumption.

◆ **Simple scalar, Reduced Instruction Set Computer (RISC)**

In 1980s, the RISC concept showed up. The essence of RISC is to handle single function within single instruction. Figure 1-2 is an example of RISC, simple scalar. Designers only deploy some primitive hardware to maintain its functionality. All the instructions of RISC have same instruction length, so it is easy to pipeline. Moreover, the performance can be easily enhanced along with the advance of technology. Using some instruction encoding techniques can raise the performance, too. Because of the regularity and simplicity of RISC, it has been widely spread out.

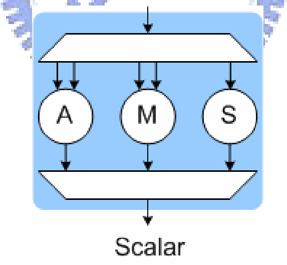


Figure 1-2 Scalar

Here are some disadvantages of RISC processors. The RISC processors have a low hardware utilization problem. They operate one function in single instruction, their hardware utilization is 1. When the number of primitive functional units increases, the low utilization characteristic remains unchanged. What is more, the RISC needs to fetch operands from register or memory first. After perform the single operation, the RISC needs to store the data into register of memory. The accesses per operation of the RISC is very high. It implies the power inefficiency.

◆ Multi-issue (VLIW)

Multi-issue is opposite to single-issue. It means that multiple instructions issued at the same time. The most popular type called Very Long Instruction Word (VLIW) stands for the multi-issue processors since it has been proposed in 1980s. Figure 1-3 shows a 3-way VLIW and a 4-way VLIW.

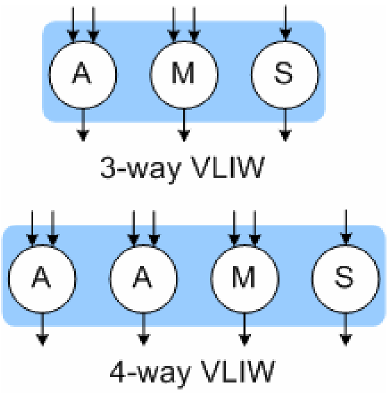


Figure 1-3 VLIW

The VLIW processors exploit the instruction level parallelism (ILP). The functional units operate concurrently. It can reach high performance by taking advantage of ILP.

The greatest problem of VLIW is the register file pressure. Because the VLIW perform multiple functions in the mean time, each functional unit needs corresponding read or write ports to the register file. The port number strongly affects the area of register file. According to [3], in full custom design, for *N* FUs, area and delay are increases as N^{3} and $N^{3/2}$. Besides, the same frequency of register or memory accesses with RISC processor makes the power inefficiency problem remained.

Finally, the performance can't be raised infinitely due to the ILP has its limit. Some other processor architectures are taken into consideration.

- **Multi-core**

After 2000, the performance of VLIW is no longer sufficient for some specific application.

Multi-core processors use several homogeneous or heterogeneous processors to do things at the same time. Hence the higher performance can be achieved. In this thesis, we only concern about the single core processor. The multi-core issue is beyond the scope.

- **Multi-threaded**

[4] In unending quest for computers with higher performance, computer system architects seek to reduce or hide latency, the number of cycles an operation takes from start to finish. A long latency may extend for 10 to 100cycles, forcing the traditional processor to sit idle until the result comes in. Less time is wasted if the latency is reduced or even hidden behind the ongoing execution of another operation.

A popular means of reducing latency is the on-chip cache memory, which can shorten the round trip to data storage from tens of cycles to just one or two. Multithreaded architectures, however, take the tack of hiding latency by supporting multiple concurrent streams of execution, or threads, which are independent of one another. The threads are interleaved on a single processor. When a long-latency operation occurs in one of the threads, another begins execution. In this way, useful work is performed while the time-consuming operation is completed.

Figure 1-4 shows the multithreaded architecture. It includes several parts including computing, selection network, hardware and software context (threads)…etc. The computing part is composed of some functional units, memory/registers, and some interconnection network. Threads are mapped onto hardware context, which each include general-purpose registers, status registers, and a program counter. One context represents a running thread, while the others represent threads that are eligible to run or are waiting on an operation to complete. Because of hardware limits, some threads are not currently mapped. The functional units handle the operations. The memory/registers store some intermediated value to accelerate the whole works. The interconnection network and the context selection hardware maintain the accuracy of each interleaved thread.

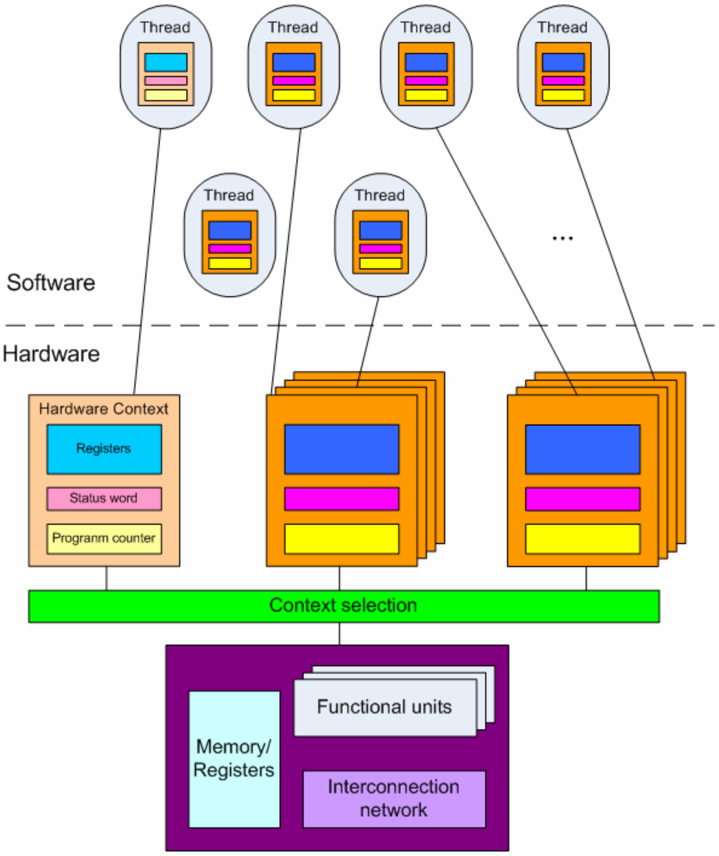


Figure 1-4 Multithreaded architecture

Multi-threaded architectures take advantage of thread level parallelism. Three categories of multi-threaded architectures are coarse-grained multithreading, fine-grained multi-threading, and simultaneous multithreading [5].

- **Coarse-grained (block) multithreading (BMT)**

  The simplest type of multi-threading is where one thread runs until it is blocked by an event that normally would create a long latency stall. Such a stall might be a cache-miss that has to access off-chip memory, which might take hundreds of CPU cycles for the data to return. Instead of waiting for the stall to resolve, a threaded processor would switch execution to another thread that was ready to run. Only when the data for the previous thread had arrived, would the previous thread be placed back on the list of ready-to-run threads.

- **Fine-grained (interleaved) multithreading (IMT)**

  A higher performance type of multithreading is where the processor switches threads every CPU cycle. The purpose is to remove all data dependency stalls from the execution pipeline. Since one thread is relatively independent from other threads, there's less chance of one instruction in one pipe stage needing an output from an older instruction in the pipeline.

- **Simultaneous multithreading (SMT)**

  The most advanced type of multi-threading applies to superscalar processors. A normal superscalar processor issues multiple instructions from a single thread every CPU cycle. In Simultaneous Multi-threading (SMT), the superscalar processor can issue instructions from multiple threads every CPU cycle. Recognizing that any single thread has a limited amount of instruction level parallelism, this type of multithreading is trying to exploit parallelism available across multiple threads to decrease the waste associated with unused issue slots.

SMT is the most complex because of the functionality among threads must be maintained. The most regular is IMT. By the way, the IMT can totally hide instruction latency if enough threads are supported. The hardware cost of IMT, since there are more threads being executed concurrently in the pipeline, shared resources such as caches and TLBs need to larger to avoid thrashing between the different threads. In this thesis, our hide instruction latency technique will focus on IMT.

1.2 Proposed Composite FUs and Contributions

In order to solve the problems mentioned above, we proposed the Composite FUs with the purposes listed below.

◆ Application-specific composite FUs

Composite FU is a cascade datapath. It cascade several primitive FUs to form a composite datapath. Analyze the characteristic of specific application, and find out what operations can be combined into single instruction.

◆ High hardware utilization and high OPs/access

Compared with the RISC processors, the composite FUs perform several functions in single instruction. It means that composite FUs do more things than RISC in a period of time. Hence, the hardware utilization improves.

Besides, the RISC needs a lot of accesses from register or memory. The composite FUs fetch proper operands and perform several operations, then store back to the register or memory. So the total number of accesses is reduced. Register access is a power-consuming action. The composite FUs have high OPs/access that lead to power efficiency potential.

◆ Low register file pressure (limited R/W ports)

Port number	Scalar	Compostie FUs	VLIW
3FUs	2R/1W	3R/1W	5R/3W
4FUs	2R/1W	4R/1W	7R/4W

Table 1-1 Port number of different datapaths

The port number of composite FUs increases one or remains unchanged as the FU number increase. Not like the VLIW processors, every FU needs two or three ports. So the grow-up trend of port number is larger in the VLIW than in the composite FUs. Fewer ports of the composite FUs ease off the register file area pressure. The example of port number is shown in Table 1-1.

◆ Suitable for IMT DSP (zero instruction latency)

When we want to reach higher performance, we will introduce our pipeline design to boost performance. Once the pipeline technique has been used, the instruction latency issue must be taken into consideration. If we ignore the pipeline latency, the data accuracy may have errors. If we just wait until the last work ready, then the performance can't be raised ideally.

There are some techniques to reduce or hide instruction latency, either from software or hardware view. Software method including loop unrolling, software pipelining, etc. and hardware method including forwarding, multithreaded architectures, etc. are all possible solutions.

In this thesis, we choose the IMT architecture to be the way of hiding instruction latency because of it can totally hide instruction latency. And the hardware cost of the composite FUs coordinated with IMT is not much. IMT needs a thread register file for each thread, the register file cost of the composite FUs is acceptable.

1.3 Thesis Organization

The rest of this thesis is organized as follow.

Chapter 2 introduces the background of our work. First, we talk about what is the composite FUs. Second, describe the meaning stand for data flow graph (DFG) and its components. Third, show a covering and match method called ID-based search graphs. Fourth, show the scheduling procedures using list scheduling based method. Fifth, introduce a RF model to estimate the area of register files. Last, illustrate how the interleaved multithreaded (IMT) architecture work.

Chapter 3 describes the comparison of the advantage and disadvantage among scalar, VLIW and Composite FUs from the area and power experiment. For high performance, we have a pipeline designer to speed up the processor with the composite FUs. And we use the IMT architecture to hide instruction latency completely.

Chapter 4 states the difference between classic and our ASIP synthesis flow. Then we propose a flow to recommend a proper composite FU for ASIP designer under certain constraints.

Finally, chapter 5 concludes this thesis and points out the direction for the future researches.

2 Background

First at all, we will give a simple illustration of the composite FUs and talk about how it works.

The composite FUs take the advantage of application characteristic. We must develop a software tool chain to analyze the applications and to find out the possibility of operation combination. We will introduce the format that we concern, DFG. Then use covering and matching technique to recognize new operations after merging. Next, we want to estimate the area of the register files. So we use a list scheduling based method to find a sub-optimal register requirement. Then the estimation is done through a RF model method related to the technology process.

We want to further speed up the performance using pipeline design. It introduces extra instruction latency problem. As mentioned before, we use IMT to hide solve the problem. So we will show how IMT works at the last of this chapter.

2.1 Composite FUs

We propose the composite FU which cascades all the primitive FUs in a customized order by analyzing the DFG (data-flow graph) of the target applications.

A composite FU is a cascade datapath. Figure 2-1 illustrates a MAS composite FU that cascades a multiplier with an adder and then a shifter. On each instruction issue, the maximum number of operations is three, i.e. multiplying operand 1 by operand 2 and then adding the result to operand 3 and finally shifting the sum by a specific value, while the minimum number is one, i.e. either one multiplication or one addition or one shift.

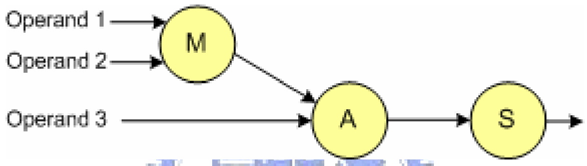


Figure 2-1 Composite FU: MAS

◆ Primitive Operation Set

For simplicity, we define a primitive operation set with three kinds of primitive operations including adder, multiplier, and shifter. Figure 2-2 shows the configuration of (a) adder, (b) multiplier, and (c) shifter.

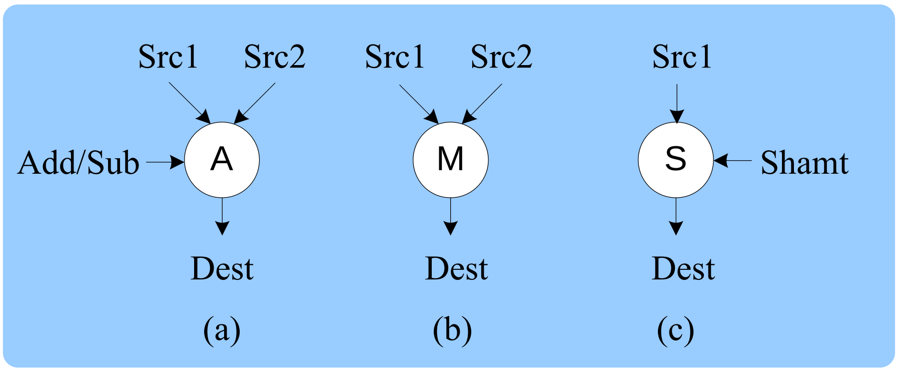


Figure 2-2 Configuration of (a) the adder; (b) the multiplier; (c) the shifter

The adder and multiplier both have two source operands and one destination. On the other hand, the shifter only has one source operands and one destination. There is a **Add/Sub** control signal to tell the adder to perform addition or subtraction. The shifter has a **Shamt** control signal to decide right shift or left shift and shift amount.

The applications which we analyze in the rest part of this thesis are all based on this primitive operation set.

◆ Port constraint

Port number restricts the possible arrangement of the composite FUs.

Figure 2-3 (a) is a full read/write ports version of composite FU MA. It has three read ports and one write port. Figure 2-3 (b) is a reduced version, and it has two read ports and one write port. Figure 2-3 (a) can be reduced to Figure 2-3 (b) through a load/store pair operation. For simplicity, we don't consider about the load/store effect and assume all the composite FUs can get full read/write ports if they need.

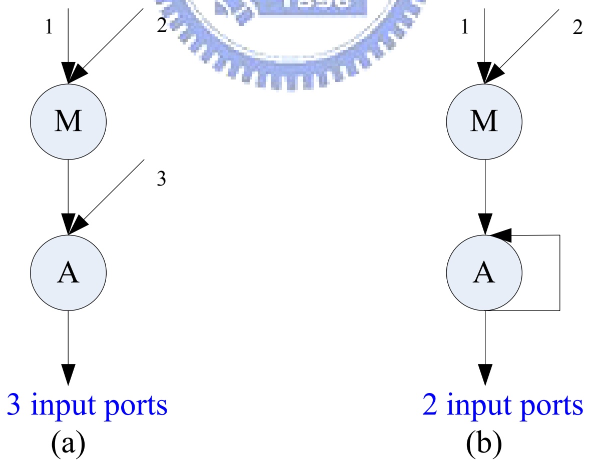


Figure 2-3 Composite FU: MA with (a) full R/W ports (b) reduced R/W ports

There are some techniques used in Sandblaster processors [6] to reduce the number of ports if the hardware doesn't need it at the same time, but it has extra huge overhead to guarantee the accuracy.

2.2 Data-Flow Graph

In mathematics and computer science, graph theory is the study of graphs; mathematical structures used to model pair-wise relations between objects from a certain collection. A "graph" in this context refers to a collection of vertices or 'nodes' and a collection of 'edges' that connect pairs of nodes. A graph may be undirected, meaning that there is no distinction between the two nodes associated with each edge, or its edges may be directed from one node to another.

[7] The data-flow graph captures the data-driven property of DSP algorithm where any node can fire (perform its computation) whenever all the input data are available. It means that a node with multiple input edges can only fire after all its precedent nodes have fired. In data-flow graph (DFG) representations, the nodes represent computations (or functions or subtasks) and the directed edges represent data paths (communications between nodes).

- **Definition**

Node N: computations

Edge E: data dependencies

Graph G = { N, E }

The precedence constraints specify the order in which the nodes in the DFG can be executed. Different representations of the same algorithm may lead to different DFG.

Figure 2-4 is a DFG example of 8 points 1D discrete cosine transform in Lee's algorithm [8].

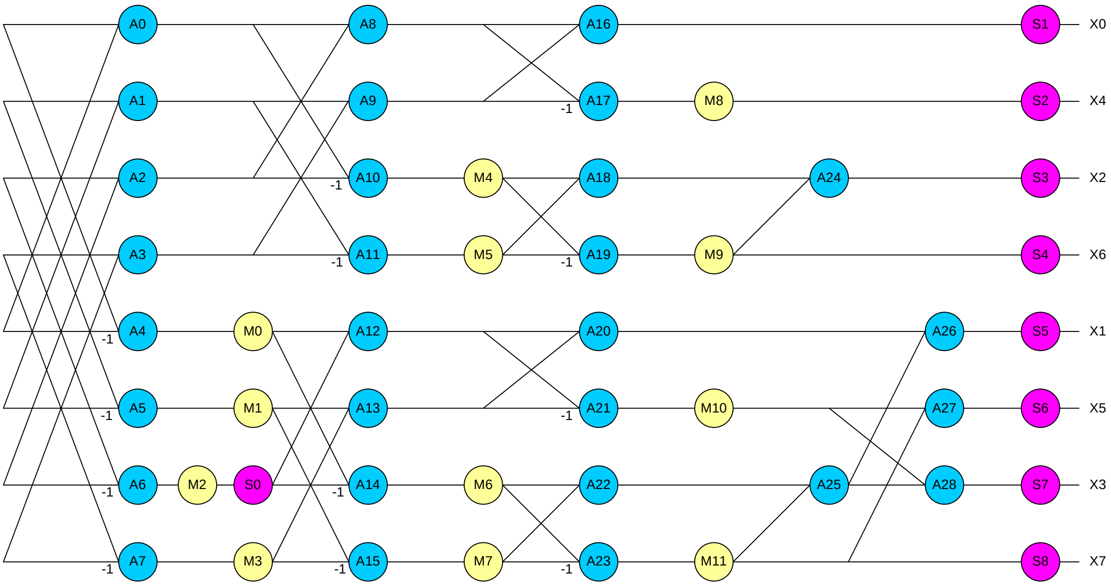


Figure 2-4 DCT (Lee's algorithm)

◆ DFG Description

For convenience, we make a DFG description. The complete DFG includes input/output part and the pure operation part. In our analysis, we suppose the load/store decoupled.

● Input/Output Part

I# (# stands for the number)

O# Src1 (Src1 is the source of output node)

● Operation Part

1. Addition/Subtraction

Syntax: **A# Src1, Src2, addsub**

Description:

Get ***Src1*** and ***Src2*** data from corresponding node and perform addition or subtraction. The ***addsub*** field stands for what the operation the adder does. "+" is for addition, and "-" is for subtraction.

2. Multiplication

Syntax: **M# Src1, Src2**

Description:

Get ***Src1*** and ***Src2*** data and perform multiplication.

3. Shift

Syntax: **S# Src1, shamt**

Description:

Get ***Src1*** data into register ***Src1*** and shift by ***shamt***-bit. The ***Shamt*** fields stand for shift amount and ranges from -8 to 7. The ***shamt*** is a 4-bit field supporting up to 8-bit left and 7-bit right shift.

- **Example**

Figure 2-5 illustrates an example of DFG description of first order biquad filter.

```
I0      A0 M0,M1,+
I1      A1 M2,M3,+
I2      A2 M4,M5,+
I3      A3 I0,A0,-
I4      A4 I1,A1,-
I5      A5 I2,A2,-
I6      A6 M6,M7,+
I7      A7 M8,A6,+
I8
I9      M0 I3,I7
I10     M1 I4,I8
I11     M2 I4,I7
        M3 I5,I8
        M4 I5,I7
        M5 I6,I8
        M6 A3,I9
        M7 A4,I10
        M8 A5,I11

        O0 A7
```

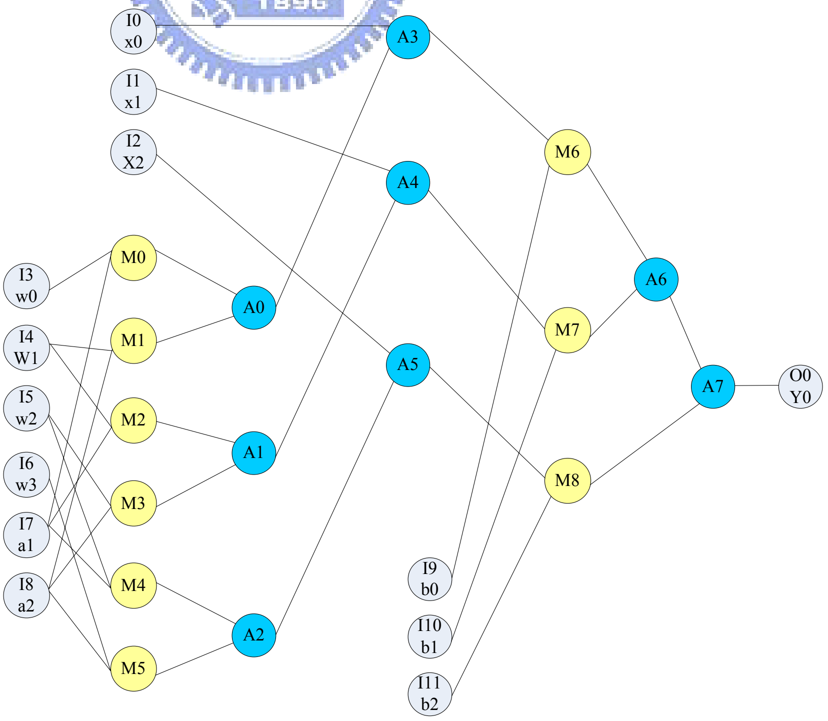


Figure 2-5 DFG description of biquad filter

2.3 Covering

In the mathematical discipline of graph theory a covering for a graph is a set of nodes (or edges) so that the elements of the set are close (adjacent) to all edges (or nodes) of the graph. We are especially interested in finding small sets with this property. The problem of finding the smallest node covering is called the node cover problem and is NP-complete.

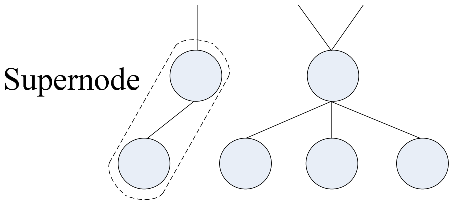


Figure 2-6 Covering and supernode

Figure 2-6 shows a **supernode** merges several nodes together. It inherits the functionality and the dependencies of the replaced nodes.

A **perfect matching** is a matching which covers all nodes of the graph. We want to find a perfect matching of the application using the composite FUs.

◆ Unate and binate covering problems [9]

The classical solving approach for two-level logic minimization in the VLSI literature goes back to Quine’s and McCluskey’s works. It reformulates the problem as a special case of the **Unate Covering Problem** [10] and applies algorithms conceived for the latter, or even for the more general Binate Covering Problem.

Binate (or unate) covering problems is a well known intractable problem. It has several important applications in logic synthesis, such as two-level logic minimization, two-level Boolean relation minimization, three-level NAND implementation, state minimization, exact encoding, and DAG covering [11].

The next paragraph briefly defines the binate covering problem and the notations of typical presentation.

Let f (y_1, …, y_n) be a Boolean function from $\{0, 1\}^n$ into $\{0, 1\}$. Let *Cost* be a function that associates a positive cost with the assignment of variable y_k to 0 or 1. The cost of a n-tuple (v_1, …, v_n) of $\{0, 1\}^n$ is defined as $\sum_{k=1}^{n} Cost\ (y_k = v_k)$.

Definition (Binate covering problem)

The binate covering problem (also called minimum cost assignment problem) consists of finding a minimal cost n-tuple that values f to 1.

- ## Node covering

[12] When deliveries, collections, or visits must be made to (or from) a number of specific (and, often, widely separated) points, the routing problem that must be solved becomes a node-covering one. The demand (or supply) points can then be represented as the nodes on the network model of the urban transportation grid and the question of the order in which to visit these nodes so as to achieve some objective is then addressed.

Our goal is similar to two-level expression, and it is a binate covering problem. [13] The difference is that our operations are not simple logic gates, but three primitive operations, including adder, multiplier, and shifter. There are some studies of the binate covering. We use a covering method called ID-based search graph proposed in IBM’s research [14] and make some modification to facilitate our analysis.

ID-based search graph

A crucial step in the design of Application-Specific Instruction-set Processors (ASIPs) [15] is the instruction-set generation. Methods for automating this process, surveyed in, extract patterns from applications, usually in the form of data-flow graphs, and insert them into a pattern library.

The ID-based search graph introduce a novel organization for pattern libraries that enables a search algorithm with only *O(d)*, where *d* is the size of the pattern sought up to the maximum pattern size in the library. Furthermore, the library organization reveals opportunities to substitute one pattern by another. This may be exploited for more efficient instruction selection and code generation. The method is presented for tree-shaped patterns but can be extended to directed acyclic graphs (DAGs).

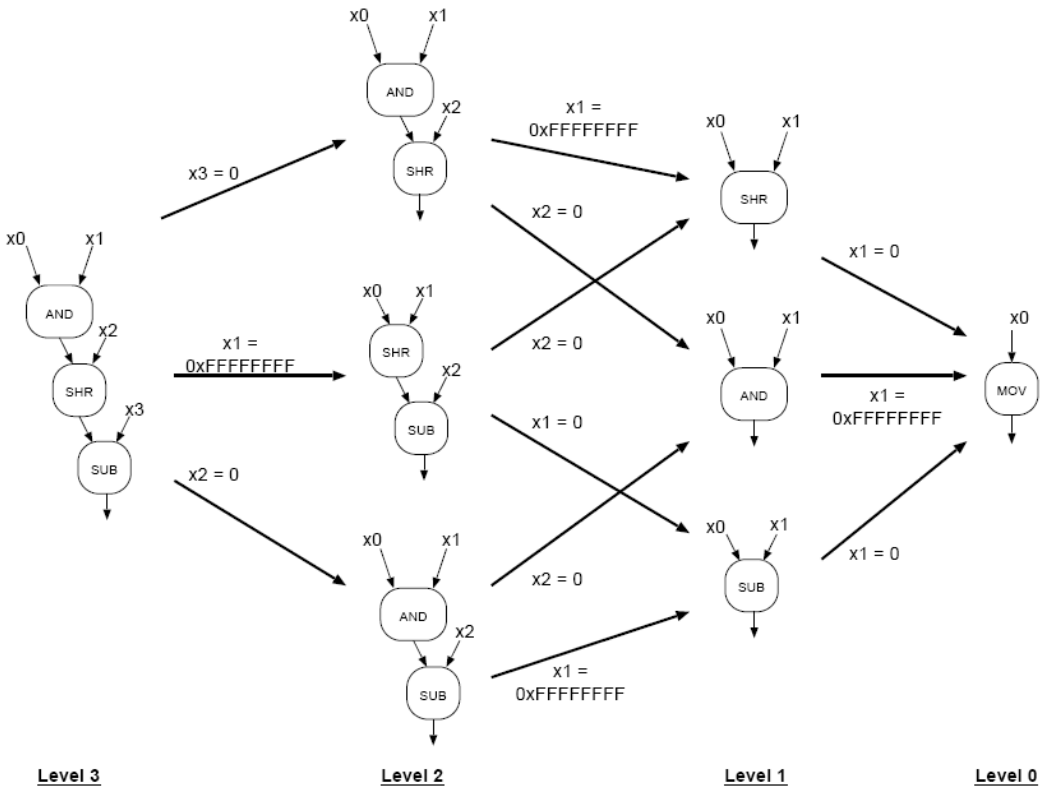


Figure 2-7 ID-Graph of a pattern

● Organizing libraries as identity graphs

In order to create match libraries for a specific application, we decompose the pattern which is used to compare with the application into several sub-levels. Figure 2-7 illustrates the procedures of ID-based search graph to organize libraries. The basic 3-levels pattern is AND-SHR-SUB. It covers an addition, a shift, and a abstraction. The sub-functions of level 2, level 1, and level 0 can be derived from by-passing some functions.

● Searching an ordered library

In order to find all matches, match all the nodes from the highest level to lowest level because of the higher level function or sub-functions can perform more operations in single instruction and execution.

2.4 Scheduling

To estimate the area of register files, we must analyze the register requirement first to find the scale of register numbers. Scheduling and resource allocation can help us to understand the register requirement. In this section, we talk about the basic idea and steps of scheduling first.

[7] Scheduling is to assign the nodes on the DFG to be processed by which functional unit in which time step. Our scheduler schedules the target DFG. Figure 2-8 shows the program flow of the scheduler. Here, we use periodic scheduling for simplicity, where only the intra-iteration data dependency is considered and the edges with delay elements (i.e. dependency across iterations) are removed from DFG first. Then we will make a lifetime analysis of every node. The DFG is scheduled with the ASAP (as soon as possible) and ALAP (as last as possible) scheduling algorithm to

obtain the range to schedule each node. Then, we apply some list scheduling based methods to the steps. At last, the scheduled DFG is output and the register requirement is reported.

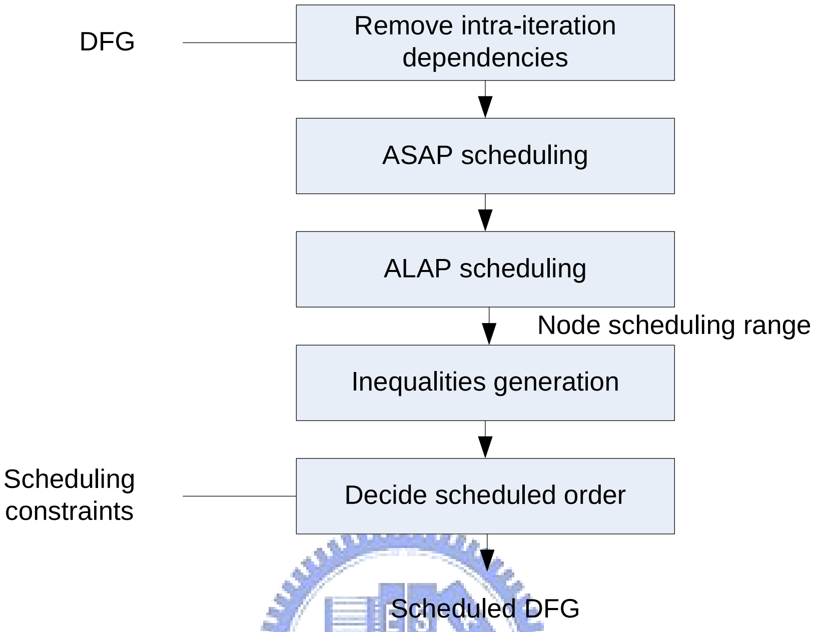


Figure 2-8 Program flow of the scheduler

- **ASAP**

ASAP (as soon as possible) is one of the earliest and simplest scheduling algorithms. ASAP scheduling assumes that the hardware resource (functional units) is unlimited. Nodes are first topologically sorted, that it, if a node n_j is constrained to follow the node n_i with a precedence constraint, then n_j will topologically follow n_i. From the sorted list, nodes are taken one at a time and placed in the earliest available time step, depending on its precedence constraint. Figure 2-9 shows the algorithm of ASAP scheduling. The ASAP scheduling is used to determine the earliest scheduling bound of each node.

```
INPUT: SDFG G=(N,E)
OUTPUT: ASAP SCHEDULE
$TS_0$=1; //SET INITIAL TIME STEP
WHILE (UNSCHEDULED NODES EXIST) {
    SELECT A NODE $N_J$ WHOSE PREDECESSORS HAVE ALREADY BEEN SCHEDULED;
    SCHEDULE NODE $N_J$ TO TIME STEP $TS_J$ = MAX $\{TS_I+(C_I)\}$ FOR ALL $N_I \rightarrow N_J$;
}
```

Figure 2-9 The ASAP scheduling algorithm

- **ALAP**

ALAP (as late as possible) scheduling is similar to the ASAP scheduling; except for the way nodes are placed in the schedule. As the name indicates, ALAP scheduling in Figure 2-10 builds the schedule bottom up and the nodes are topological sorted in reversed order. Therefore, the algorithm must have the information of the iteration period to build the schedule from the bottom up and the iteration period must be long enough to allow all the nodes to be scheduled, otherwise the scheduling will fail.

```
INPUT: SDFG G=(N,E), ITERATION PERIOD:T
OUTPUT: ALAP SCHEDULE
$TS_0$=T; //SET TIME STEP
WHILE (UNSCHEDULED NODES EXIST) {
    SELECT A NODE $N_J$ WHOSE SUCCESSORS HAVE ALREADY BEEN SCHEDULED;
    SCHEDULE NODE $N_J$ TO TIME STEP $TS_J$ = MAX $\{TS_I-(C_I)\}$ FOR ALL $N_I \rightarrow N_J$;
}
```

Figure 2-10 The ALAP scheduling algorithm

◆ ILP-based scheduling

After the ASAP and ALAP scheduling, we describe and solve the scheduling problem by inequalities and priorities. Figure 2-11 gives examples of ASAP, ALAP, and the scheduling range from ASAP and ALAP. When constructing inequalities for constraints, each position (i.e. node *i* at the time step *j*) in the range are associated with a Boolean variable $x_{i.j}$ which indicates whether a node *i* is scheduled into the time step *j*. The following four constraints must be satisfied.

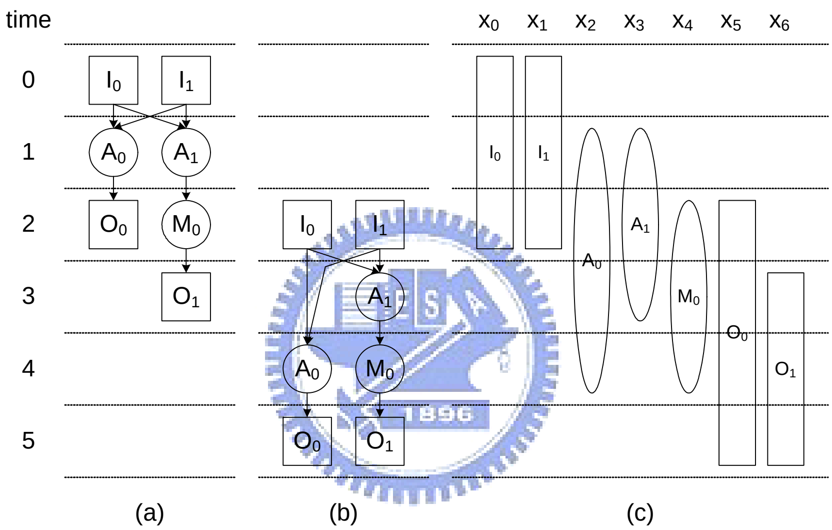


Figure 2-11 Scheduling example (a) ASAP (b) ALAP (c) scheduling range

1. *Resource constraints*

This constraint states that no schedule will have a time step that contains more operations than the available functional units due to the limited hardware resources. Because we assume that the I/O unit and adder are all of one, the inequalities for this constraint of the example in Fig. 2-11 (c) should be:

$x_{0.0}+x_{1.0}\le 1;\quad x_{0.1}+x_{1.1}\le 1;\quad x_{0.2}+x_{1.2}\le 1$ (for input)

$x_{2.1}+x_{3.1}\le 1;\quad x_{2.2}+x_{3.2}\le 1;\quad x_{2.3}+x_{3.3}\le 1$ (for adder)

$x_{5.3}+x_{6.3}\le 1;\quad x_{5.4}+x_{6.4}\le 1;\quad x_{5.5}+x_{6.5}\le 1$ (for output)

2. *Allocation constraints*

This constraint states each node can only be scheduled within the scheduling range bounded by some range obtained from the ASAP and ALAP scheduling and can only appear once in the schedule. The inequalities for this constraint of the example in Fig. 2-11 (c) should be:

$x_{0.0}+ x_{0.1}+ x_{0.2} = 1$

$x_{1.0}+ x_{1.1}+ x_{1.2} = 1$

$x_{2.1}+ x_{2.2}+ x_{2.3}+ x_{2.4} = 1$

$x_{3.1}+ x_{3.2}+ x_{3.3} = 1$

$x_{4.2}+ x_{4.3}+ x_{4.4} = 1$

$x_{5.2}+ x_{5.3}+ x_{5.4}+ x_{5.5} = 1$

$x_{6.3}+ x_{6.4}+ x_{6.5} = 1$

3. *Dependency constraints*

The data dependency in the original DFG should be strictly followed when scheduling. The dependency constraint ensures that DFG remains causal and correct timing sequence. The inequalities for this constraint of the example in Fig. 2-11 (c) should be:

$x_{0.0} + 2\ x_{0.1} + 3\ x_{0.2} - 2\ x_{2.1} - 3\ x_{2.2} - 4\ x_{2.3} - 5\ x_{2.4} \leq -1$

$x_{1.0} + 2\ x_{1.1} + 3\ x_{1.2} - 2\ x_{2.1} - 3\ x_{2.2} - 4\ x_{2.3} - 5\ x_{2.4} \leq -1$

$x_{0.0} + 2\ x_{0.1} + 3\ x_{0.2} - 2\ x_{3.1} - 3\ x_{3.2} - 4\ x_{3.3} \leq -1$

$x_{1.0} + 2\ x_{1.1} + 3\ x_{1.2} - 2\ x_{3.1} - 3\ x_{3.2} - 4\ x_{3.3} \leq -1$

$2\ x_{2.1} + 3\ x_{2.2} + 4\ x_{2.3} + 5\ x_{2.4} - 3\ x_{5.2} - 4\ x_{5.3} - 5\ x_{5.4} - 6\ x_{5.5} \leq -1$

$2\ x_{3.1} + 3\ x_{3.2} + 4\ x_{3.3} - 3\ x_{4.2} - 4\ x_{4.3} - 5\ x_{4.4} \leq -1$

$3\ x_{4.2} + 4\ x_{4.3} + 5\ x_{4.4} - 4\ x_{6.3} - 5\ x_{6.4} - 6\ x_{6.5} \leq -1$

4. *Port conflict constraints*

If the functional units are consisted of single-write instead of *N*-write memory or registers to reduce the hardware complexity, it may introduce port conflicts when multiple functional units simultaneously write their results into the same memory or registers. We can schedule the operations with an identical destination memory or registers into different time slots by incorporating the following port constraints to prevent conflicts. The inequalities for this constraint of the example in Fig. 2-11 (c) should be:

$x_{0.0}+ x_{1.0} \leq 1$; $x_{0.1}+ x_{1.1} \leq 1$; $x_{0.2}+ x_{1.2} \leq 1$; (for adder)

$x_{2.2}+ x_{4.2} \leq 1$; $x_{2.3}+ x_{4.3} \leq 1$; $x_{2.4}+ x_{4.4} \leq 1$; (for output)

According to the assumption above, the composite FUs discussed in this thesis have full read/write ports. The port conflict constraints are mapped to the different situations.

2.5 Complexity of Synthesized Register File

We want to compare the RF area of the composite FUs with the RF area of the VLIW. We need a roughly estimation method to see the scale and the pressure of the RF area. Here is a survey of the relations among RF area and the FU number and port number in [16], and it proposes a simple RF model for 0.18 um process.

Conventionally, the microprocessors have a more efficient and direct data exchange mechanism among the parallel FUs through the register file (RF) than the multi-processors, where a monolithic and centralized RF provides storages for and interconnects to each FU in a general and homogeneous manner. However, the complexity of the centralized RF grows with the number of access ports increasing.

In centralized RF, every read port and write port can directly access any registers in the RF. Prior art has evaluate the complexity of centralized RF in full-custom designs.

The RF consists of register cells similar to SRAM. Generally, the register cell is organized as 6-transistor structure, and the access port of RF requires a word-line and a bit-line which is shown in Figure 2-12. Therefore, the area of single SRAM cell grows in two dimensions and can be described as P^2. For a full-custom centralized RF with n-registers and P-ports, the area and delay are approximated to $n \times P^2$ and $n^{1/2} \times P$ respectively [3].

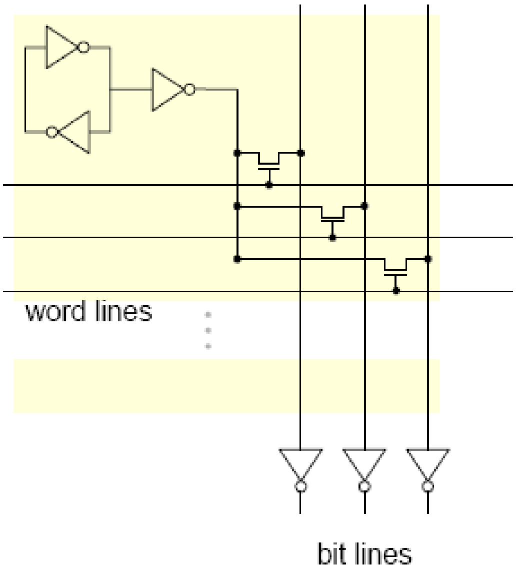


Figure 2-12 A register cell in full custom design

The number of registers and ports required can be estimated from the number of FUs. Assume that each FU requires two ports for two source operands and one port to write the result back to RF, and every FU requires at least one register to store the manipulated data. Consequently, for N FUs, the number of registers and ports is approximated to N and 3N respectively. As a result, the area and delay are increases as N^{3} and $N^{3/2}$.

In this thesis we focus on the cell-based RF organization which consists of flip-flops and switch networks. For a centralized RF with N registers, x read ports, y write ports, and word length W, the complexity of read access network is an N to x crossbar router and every read port has one N to 1 multiplexer. Similarly, the complexity of write access network will be y to N crossbar router with y to 1 multiplexer for each register elements. Figure 2-13 shows the access network of a centralized RF.

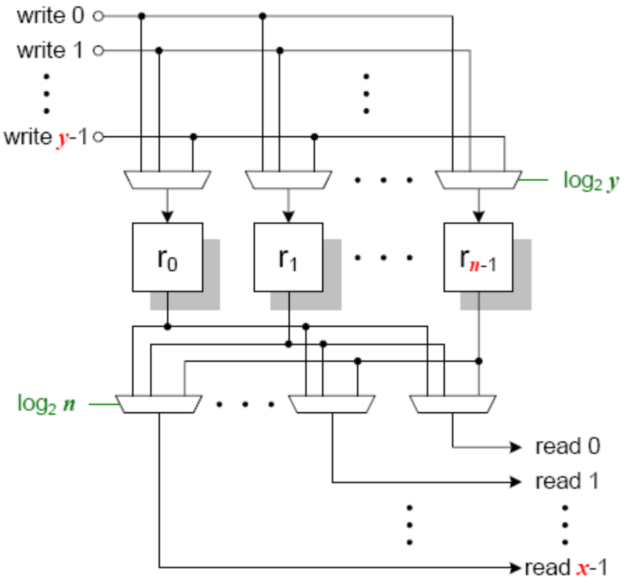


Figure 2-13 Access network of centralized register file

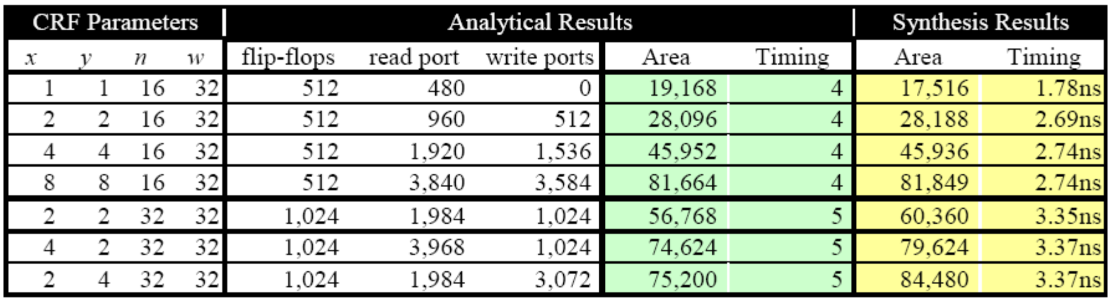
The output of each register has to drive x × N-to-1 multiplexers which results in a very high output loading capacitance. The centralized RF is efficient when executing program with high data-level parallelism only. But it is difficult to guarantee the data-parallelism level of applications, so it is unnecessary to provide the bandwidth for non-common cases with the cost of RF access time and RF area.

We roughly evaluate the area and speed of centralized RF based on the architecture shown in Figure 2-13. The cost function of area complexity for centralized RF is analyzed as follows:

$$
\begin{aligned}
f_{A\,\mathrm{CRF}}\,(x, y, N, W) &= A_{\mathrm{write\_network}} + A_{\mathrm{read\_network}} + A_{\mathrm{registers}} + A_{\mathrm{control}} \\
&\cong [N \times A_{\mathrm{y\text{-}to\text{-}1\_mux}} + x \times A_{\mathrm{N\text{-}to\text{-}1\_mux}} + N \times A_{\mathrm{D\text{-}flipflop}}] \times W \\
&\cong [N \times (y - 1) \times A_{\mathrm{2\text{-}to\text{-}1\_mux}} + x \times (N - 1) \times A_{\mathrm{2\text{-}to\text{-}1\_mux}} \\
&\quad + N \times A_{\mathrm{D\text{-}flipflop}}] \times W
\end{aligned}
$$

This Function assumes all multiplexers are composed of 2-to-1 multiplexers and the control overhead is neglected. This analytical formula is also based on the number of 2-to-1 multiplexers on data path. The experimental results show the analysis is close to synthesis result. Table 2-1 compares the analytical analysis with synthesis results using TSMC 0.18um cell library in different centralized RF configurations. In this thesis, the discussion of timing estimation is out of scope.

The area of analytical results is based on the cell library. The constraint of synthesis is optimized for area to avoid the variance due to the timing optimization technique. The area is over estimated in 16-entry, 1R1W configuration because of the 2-to-1 multiplexer model is actually larger than 4-to-1 multiplexer cell that synthesis tool uses. From the analysis, we can sum up that the area of cell-based CRF is direct proportion to both number of registers and number of ports and the synthesis results prove the point. The natural of high routing complexity of RF will cause high variance in physical implementation, and the variance growing with the number of registers and ports increasing.



CRF Parameters				Analytical Results					Synthesis Results	
x	*y*	*n*	*w*	flip-flops	read port	write ports	Area	Timing	Area	Timing
1	1	16	32	512	480	0	19,168	4	17,516	1.78ns
2	2	16	32	512	960	512	28,096	4	28,188	2.69ns
4	4	16	32	512	1,920	1,536	45,952	4	45,936	2.74ns
8	8	16	32	512	3,840	3,584	81,664	4	81,849	2.74ns
2	2	32	32	1,024	1,984	1,024	56,768	5	60,360	3.35ns
4	2	32	32	1,024	3,968	1,024	74,624	5	79,624	3.37ns
2	4	32	32	1,024	1,984	3,072	75,200	5	84,480	3.37ns

Table 2-1 Comparison of analytical results and synthesis results

2.6 Interleaved Multithreaded (IMT) Architecture

When we use pipeline design to boost the performance of the composite FUs, it will introduce instruction latency. This section shows how the IMT architecture hides the instruction latency totally.

Figure 2-14 illustrates an example of the interleaved threads and dependencies in the pipeline. If an IMT architecture processor has 5 stages and the pipeline latency is 5 cycles, it needs 5 interleaved threads to hide instruction latency totally. These 5 threads are independent tasks.

The operation is to issue the first instruction of thread 1 at cycle 1 into the pipeline, then issue the first instruction of thread 2 at cycle 2, and so on. After first iteration from thread 1 to thread 5 is done, issue the second instruction of thread 1. In this way, the following iterations are executed.

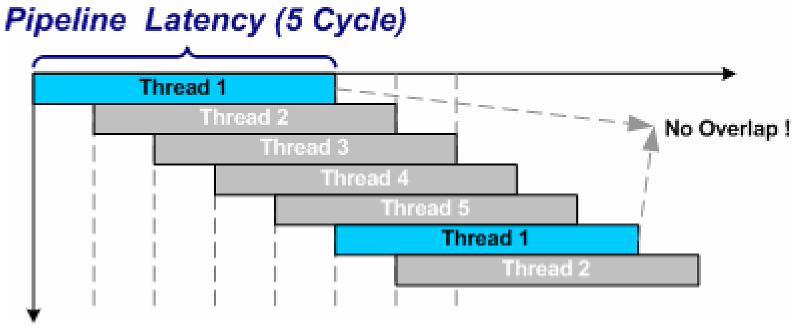


Figure 2-14 Interleaved threads and dependencies in the pipeline

The first instruction and second instruction of thread 1 may have dependency. In the original pipeline design, if their execution time is overlapped, there would be some kind of data hazard. So it must stall the second instruction until the first instruction complete, but this handling method hinders the speedup of performance.

As Figure 2-14, interleaved threads method solves the problem. The execution time of first and second instructions has no overlapped. It hides the instruction latency without any performance loss.

3 The Composite FUs

We want to quantify the hardware utilization of the composite FUs. Once the hardware utilization is upgraded, we analyze the area pressure of register file for different datapaths under different performance constraint.

In section 3-1, we introduce the composite FUs generator to analyze the operations per cycle of all kinds of composite FUs running different applications. After then, the register requirement and the estimated area of register files is calculated by the method in section 2-5 with small modification. Afterward, use pipeline design to boost performance. The pipeline design flow is described in section 3-2. Because of the pipeline design introduces instruction latency, the IMT is used for pipelined composite FUs. We will talk about the hardware cost of IMT.

As a result of operands operated for more than one operation after being fetched, the total register accesses are reduced. We also estimate the power consumption to see where the power is saving.

3.1 The Composite FUs Generator

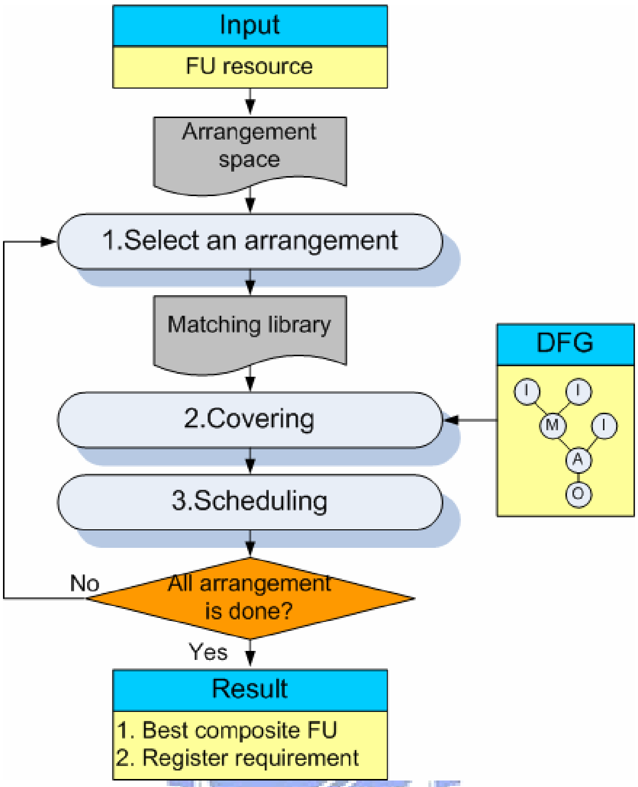


Figure 3-1 The composite FUs generator

We introduce the composite FUs generator in Figure 3-1 for application analysis. When the composite FUs generator receives a FU resource constraint and an application in DFG description, three recursive steps have been taken iteratively.

First, select an arrangement from possible candidate formed by FU resource constraint. Second, create an ID-based search graph and match libraries of the arrangement and cover the DFG using the function set of the arrangement. Finally, the register requirement is calculated for further area estimation.

Repeat these steps until all the case of arrangements are done, we can get operations per cycle (OPC) and register requirement for each composite FU. The composite FU with best OPC implies the most proper datapath and the hardware utilization is the highest for the application.

3.1.1 Arrangement Space

For given FU resources, there are many arrangements to form a composite FU.

Number of functional units: ***N***

Number of adders: $\boldsymbol{N_A}$

Number of multiplier: $\boldsymbol{N_M}$

Number of shifter: $\boldsymbol{N_S}$

The total arrangements: $S = \dfrac{N!}{N_A!*N_M!*N_S!}$

In this thesis, we concern about two cases of function units set. One is 3 FUs composed of 1 adder, 1 multiplier, and 1 shifter. The other is 4 FUs composed of 2 adders, 1 multiplier, and 1 shifter. For the former case, NA = 1, NM = 1, NS = 1, so the total arrangements S is 6 as Figure 3-2 shows. For the latter case, NA = 2, NM = 1, NS = 1, so the total arrangements S is 12 as Figure 3-3 shows.

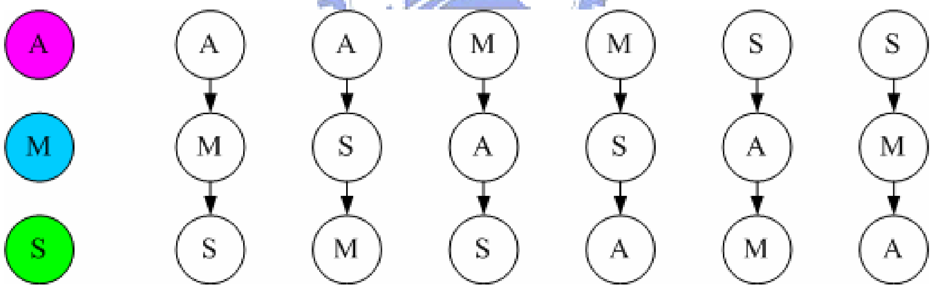


Figure 3-2 The arrangement space of 1 A 1M 1S

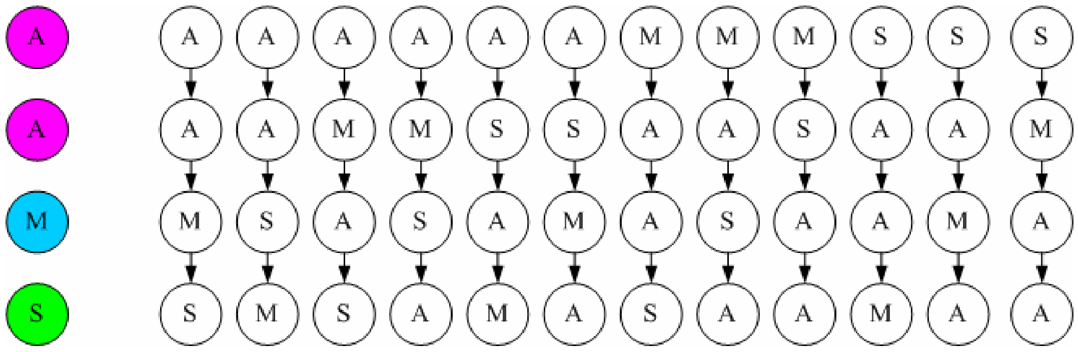

A A A A A A A M M M S S S
A A A M M S S A A S A A M
M M S A S A M A S A A M A
S S M S A M A S A A M A A

Figure 3-3 The arrangement space of 2 A 1M 1S

3.1.2 ID-based Search Graph Covering

According to section 2-3, we map the ID-based search graph from Boolean expression to the analysis of the composite FUs. We use exhaustively tree-structured search principle to simplify selected composite FU.

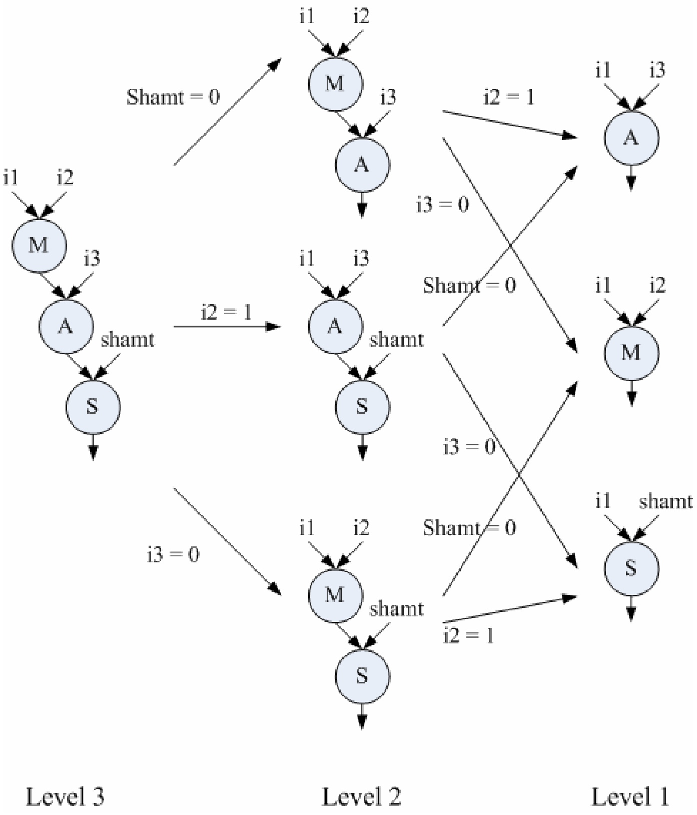


Figure 3-4 The ID-based search graph of MAS

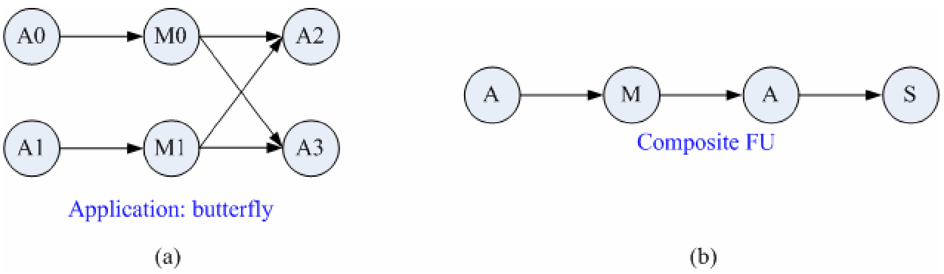
Figure 3-4 illustrates composite FU: MAS and its sub-level functions. First, the MAS-ordered composite FU is categorized into 3 levels of operations. That is, level-3 operation, MAS, performs one multiplication, one addition plus one shift serially. Level-2 operations, including MA, AS, and MS, perform either one multiplication and one addition, or one addition and one shifter, or one multiplication and one shifter serially. Level-1 operations, including M, A, and S, perform either one multiplication, one addition or one shift serially. We can use adding zero, multiplying by one and shifting by zero to bypass the adder, the multiplier and the shifter respectively.

Then, the ID-based covering is performed to cover the nodes of the input DFG by available operations, i.e. level-3, level-2 or level-1 operations. Note that, all covered nodes are assumed to consume one clock cycle. The covering process first searches the whole DFG for any pattern that is matched to the level-3 operation and then for any remaining uncovered node patterns that are matched to the level-2 operation and so on.

◆ Multiple Fan-out Handling and Optimization

● Multiple fan-out problem

Figure 3-5 (a) shows a DFG of a butterfly application, we use Figure 3-5 (b), composite FU: AMAS to cover this DFG. If the A0-M0-A2 is merged together when the match is proceeding in level 3 match for AMA, one of the A3 node can’t get the data from M0 node because of the composite FU AMAS don’t have a write port after multiplication to store the right value into the register file. Hence, the application cannot be fulfilled. Some actions must be taken to prevent the hazard from happened.



(a)

(b)

Figure 3-5 (a) a DFG of butterfly (b) the composite FU: AMAS

● Break

The simplest method to avoid the data hazard is to break at the multiple fan-out nodes. Figure 3-6 illustrates the procedures. The level 2 AM function matches A0-M0 and A1-M1 to form new nodes C1 and C3. The level 1 A function matches A2 and A4 to form new nodes C2 and C4. It takes 4 nodes to perform the butterfly.

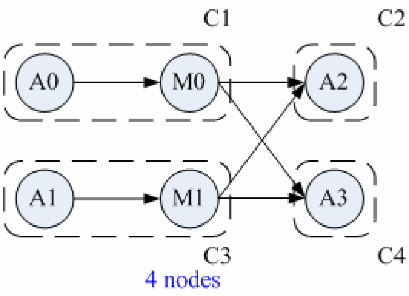


Figure 3-6 Break at multiple fan-out node

- **Node duplication**

For optimization, we use the node duplication technique to minimize the number of matches. Figure 3-7 (a) shows the spirit of node duplication. Breaking at M0 to let A3 get correct source value needs a instruction for A3 only. But we can repeat the operation A0-Mo in the same instruction with A3 because the characteristic of the composite FU allows these work done in the same instruction.

Figure 3-7 (b) illustrates the procedures. The level 3 AMA function matches A0-M0-A2 and A0-M0-A3 to form new nodes C1 and C2. The level 2 AM function matches A1-M1 to form new node C3. It only takes 3 nodes, equals 3 instruction for execution, to perform the butterfly. The node duplication technique reduces the nodes of the covering result form 4 nodes to 3 nodes.

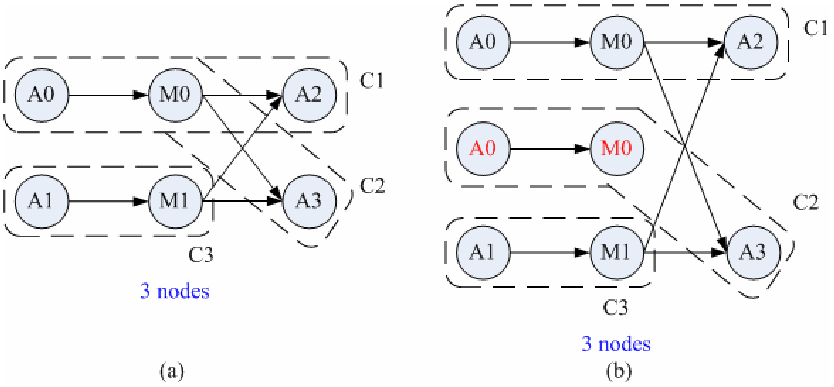


Figure 3-7 Node duplication at multiple fan-out nodes

(a) covering (b) duplicate A0 and M0

3.1.3 Scheduling

Since we want to estimate the area of register files, the scheduling principles with optimizing for register requirement are chosen. But finding out the minimum register requirement is a NP-complete problem. So we use two simple scheduling to get the sub-optimal solution. Some concepts are from [17], forced-directed list-scheduling [18], loop-list scheduling [19], and cone-base list-scheduling [20].

◆ Lifetime Analysis

Based on section 2-4, the lifetime analysis about ASAP scheduling, ALAP scheduling, and range estimation is done first. Then we construct resource, allocation, dependency, and port conflict constraints. With these inequalities, we can put some priorities to solve the linear programming problem and make the scheduled order.

◆ Cutsets

Cut the DFG into branches, called cutsets which have no dependencies of each other. Figure 3-8 cuts the DFG into three independent cutsets. Give the independent cutsets different priorities and finish the cutset one by one can minimize the live registers at the same time space.

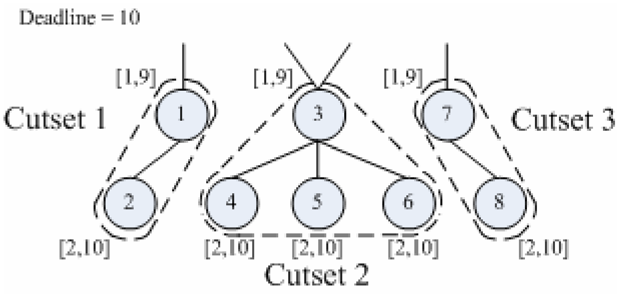


Figure 3-8 Cutset of DFG

For example in Figure 3-8, assume that only one node can be executed in one cycle because of limited hardware resource, and the **hard deadline** of the whole application is 10. The range of lifetime analysis of each node is listed in the square bracket near the node. The value before comma is the earliest time step of ASAP scheduling, and the other value after comma is the latest time step of ALAP scheduling. Node 1 in cutset 1, node 3 in cutset 2, and node 7 in cutset 3 have the same lifetime. It means that their live ranges are completely the same. Without cutset priorities, the scheduling result of cycle 1 is node 1, cycle 2 is node 3, and cycle 3 is node 7, and so on. The register requirement is 3 which happened to store intermediate value of node 1, 3, and 7. On the contrary, set cutset priorities 1 to nodes of cutset 1, 2 to nodes of cutset 2, and 3 to nodes of cutset 3. The scheduling result of cycle 1 is node 1, cycle 2 is node 2, cycle 3 is node 3, and so on. The register requirement is only 1, either store 1, 3, or 7. The register requirement is reduced.

◆ Priorities

Our resource is only one composite FU, if refines the resource constraint. We give different priorities for the independent cutsets first, then execution priorities. We use two methods to get the scheduled order and take the smaller register requirement as result. Additionally, if all the priority is the same, we will do the node with smaller serial number from the DFG description.

Scheduling 1

Based on the first come first serve principle, the execution priority setting start according to the ASAP result. Search the time space of ASAP and mark its order if its ancestor nodes are all in ready list.

Scheduling 2

If the timing constraint is critical, the deadline of each node is highly notified. The execution priority setting start according to the ALAP result, search the time space of ALAP and mark its order if its ancestor nodes are all in ready list.

◆ Example

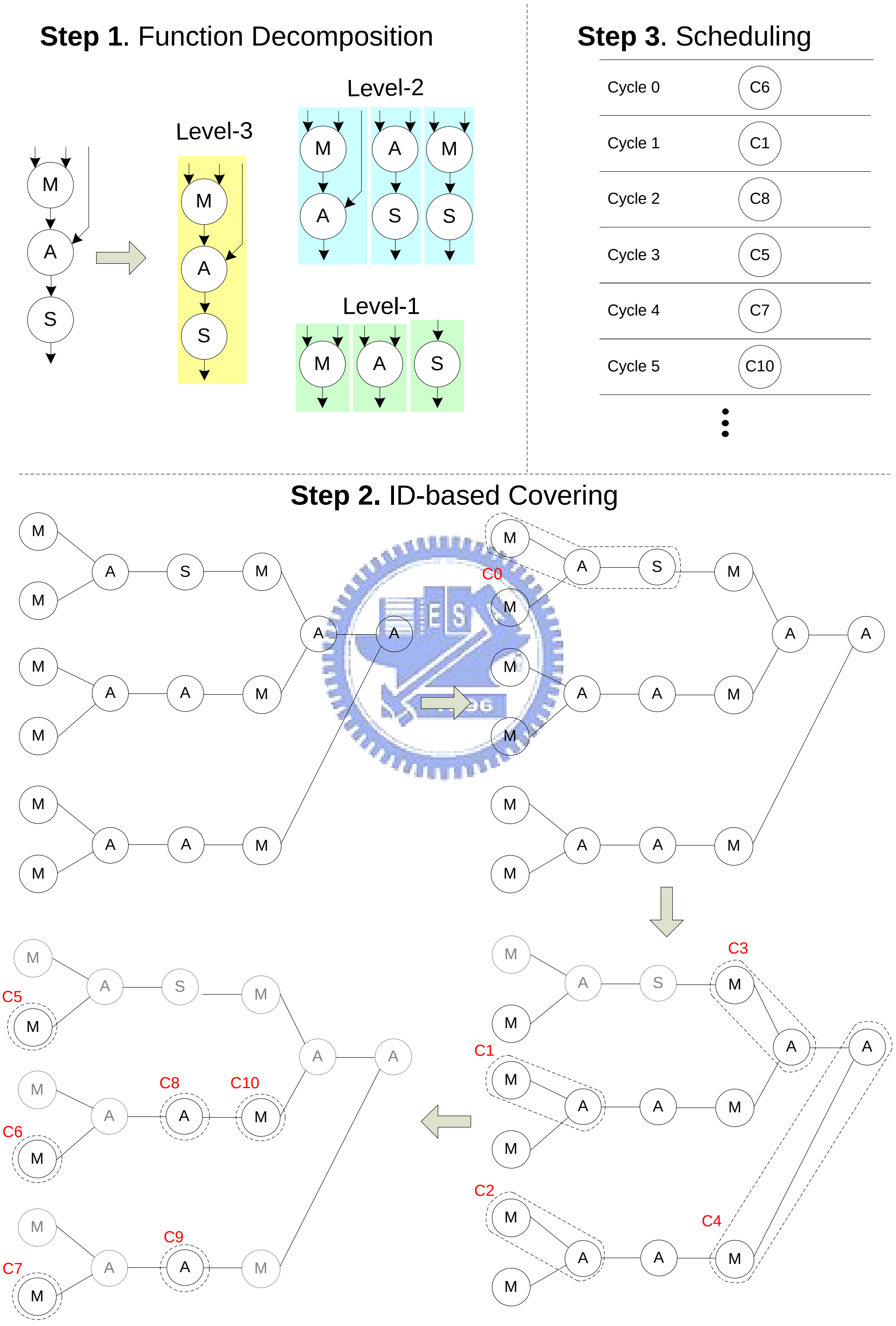


Figure 3-9 Recursive steps of automatic composite FUs generation

Figure 3-9 shows recursive steps of automatic composite FUs generation. The three steps are listed as follow.

- **Step 1. Function decomposition**

The MAS-ordered composite FU is decomposed into 3 levels because it has three functional units. Level-3 operation is MAS. The number of level-2 operations is 3 including MA, AS, and MS. The number of level-1 operation is 3 including M, A, and S. These 7 operations compose of match libraries.

- **Step 2 ID-based covering**

Covering procedures start from the matching of level-3 operation. It finds a MAS match and marks it in C0. Then the matching of level-2 operations starts, and it finds 4 matches and mark it in C1~C4. Finally, the matching of rest of nodes using level-1 operations and make up C5~C10.

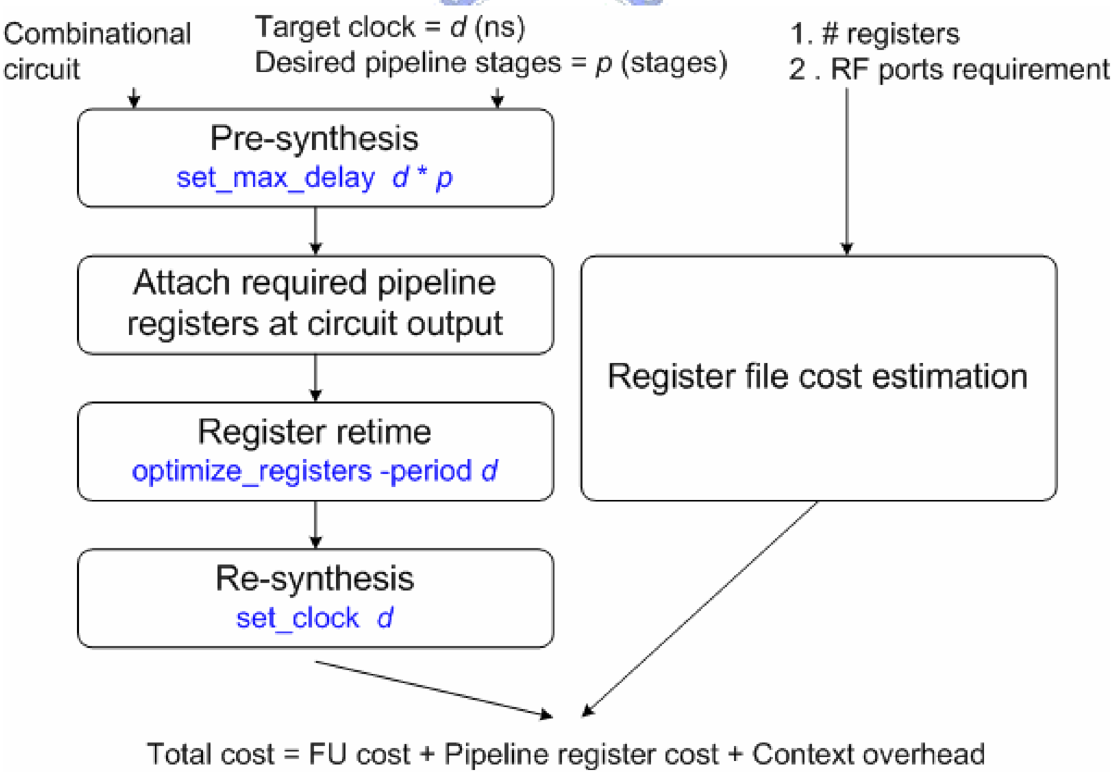
- **Step 3 Scheduling**

After ASAP and ALAP, the lifetime analysis is done and the scheduling result is C6 in cycle 1, C1 in cycle 2, C8 in cycle 3, C5 in cycle 4, C7 in cycle 5, C10 in cycle 6, and so on. And the register requirement is calculated as well.

3.2 Pipeline Design Flow and Hardware Cost of IMT

After the composite FU generation, the pipeline design is performed to decide the required pipeline stages according to the target performance. For example, if the OPC of some specific composite FU is 1.5 regarding a specific application and the target performance is 600MOPS (million operations per second), the required operation frequency is 400MHz.

We use the Synopsys DesignCompiler synthesis tool incorporated with the design flow shown in [] to analyze the various pipeline stage sand the associated area cost. This flow also applies for scalar, VLIW as well as the composite FUs for area comparison in the experiment introduced later.



Combinational circuit
Target clock = *d* (ns)
Desired pipeline stages = *p* (stages)
1. # registers
2 . RF ports requirement
Pre-synthesis
set_max_delay *d* * *p*
Attach required pipeline registers at circuit output
Register retime
optimize_registers -period *d*
Re-synthesis
set_clock *d*
Register file cost estimation
Total cost = FU cost + Pipeline register cost + Context overhead

Figure 3-10 The flow of pipeline design

On the other hand, for analyzing the area cost of datapath and the pipeline registers, we use the pipeline design flow in Figure 3-10. First, if the target clock cycle is d (ns) and the desired number of pipeline stages is p, we pre-synthesize the purely combinational circuit, i.e. MSA, with the input-to-output delay of d*p (ns). Then, (p-1)-stage pipeline registers are attached to the output of the synthesized netlist which is imported into DesignCompiler to perform register retiming with the target clock cycle d (ns) as clock constraint. Finally, the resultant netlist is re-synthesized again and the area cost is recorded.

The main innovations of the pipelining design flow are two-folded. The first is to avoid a over-designed netlist. The conventional method that does not take the target clock rate and the desired number of pipeline stages into consideration may use a tight timing constraint to pre-synthesize the purely combinational circuit and thus results in a over-designed netlist of excessively large area. The other innovation of the proposed pipeline design flow can avoid the situation that insufficient pipeline stages will result in so tight clock timing constraint that the resultant netlist is of excessively large area. In our experience, under some cases, the synthesized netlist with more pipeline stages will has smaller area than that with less pipeline stages.

◆ Hardware Cost of IMT

Because we use interleaved multithreading technique to hide pipeline latency, one more set of context, i.e. register file, is required for one more pipeline stage. Because the port number of the composite is slightly larger than scalar processors and greatly smaller than VLIW processors, the IMT architecture is suitable for the composite FUs. Comparatively, the heavy area pressure hinders the VLIW from using IMT architecture.

4 Application Specific Programmable Processor Synthesis

In classic ASIP synthesis flow, programmers usually design the instruction set architecture (ISA) and micro-architecture first. Then the software programmers and hardware designers develop the software tool chain and hardware architecture according to ISA and micro-architecture separately. This flow starts from high level synthesis and forms a top-down design.

We propose an Application Specific Programmable Processor Flow in section 4-2 to explore the design space of the composite FUs. Then we describes a composite FU selection flow to help users to find out the most proper composite FU.

4.1 High Level ASIP Synthesis Flow

[21] shows typical procedures of high level synthesis. Some instruction set matching and selection techniques are described in [22][23] [24]. [25] introduces the critical path and data path optimization for ASIP design.

High-level synthesis takes some kind of behavioral description of the algorithm, available hardware resources, and a set of constraints and goals, to generate the hardware architecture in register-transfer level (RTL). The set of constraints and goals define the desired performance and characteristics of the final architecture. The most common constraints are area and performance constraints.

Area constrained problems means that given a set of resources (or functional units), try to implement the application using those resources such that it has the highest performance. This is known as ***resource-constrained scheduling***.

The performance constrained problem is known as ***time-constrained scheduling***, where the designer is given a desired sample rate or iteration period and the goal is to minimize the total area of the final architecture.

There are other goals during the synthesis problem depending on the user requirement such as minimizing the number of memory modules, reducing the power consumption, minimizing the number of busses, incorporating reliability and testability into the design, etc.

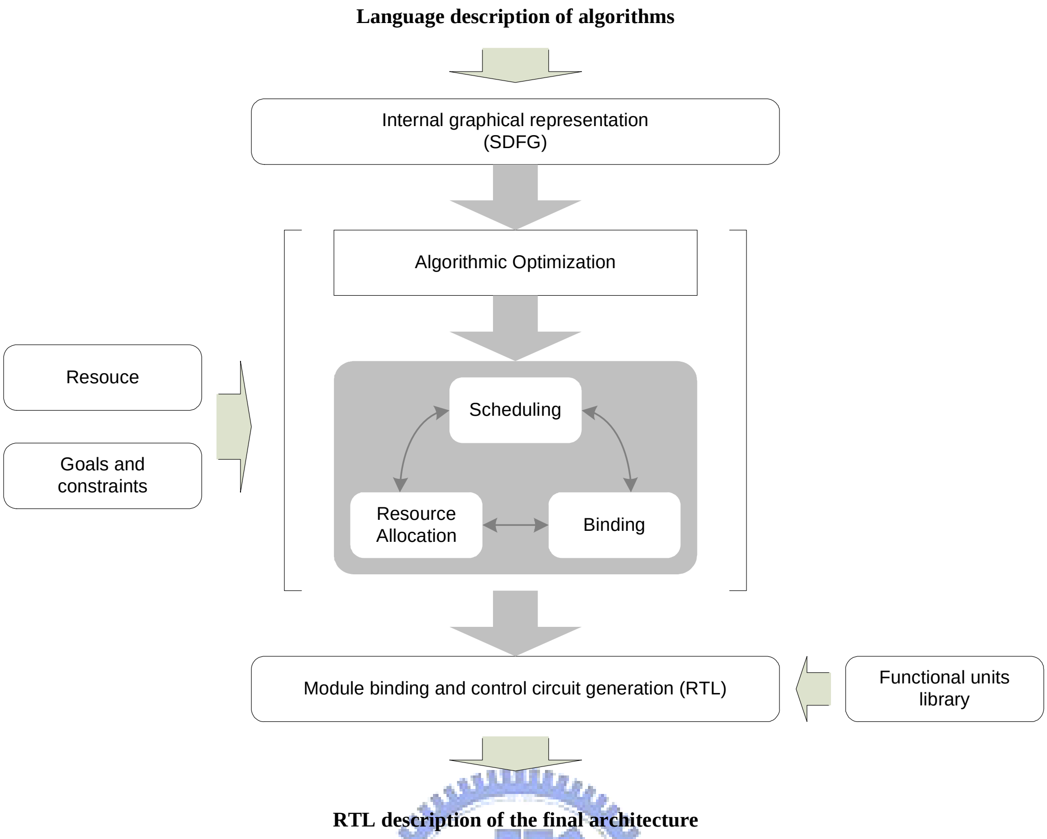

Language description of algorithms
Internal graphical representation (SDFG)
Algorithmic Optimization
Resouce
Goals and constraints
Scheduling
Resource Allocation
Binding
Module binding and control circuit generation (RTL)
Functional units library
RTL description of the final architecture

Figure 4-1 High-level synthesis of DSP datapath

Figure 4-1 illustrates the design flow of the high-level DSP synthesis system. The behavioral description which may be represented in C/C++ is first converted to a graph-based representation, and such as data-flow graph[]. In the DFG representations, the nodes represent computations (or functions or subtasks) and the directed edges represent data paths and each edge has a nonnegative number of delays associated with it. The following tasks in high-level synthesis of DSP datapath include high-level optimization, scheduling, resource allocation, module binding, and control generation. The final architecture produced by high-level synthesis is typically at the synthesizable RTL. Many high-level synthesis systems have been designed and a great deal of progress has been made in finding good techniques for optimizing and exploring design tradeoffs. In addition, the trend towards more automation at higher level of design process is expected to continue.

◆ Scheduling

Scheduling and resource allocation are two important tasks in hardware or software synthesis of DSP system. They are both interrelated and dependent on each other and are among the most difficult problems of high-level synthesis.

Scheduling involves assigning every node of the DFG to time steps. Time steps are the fundamental sequencing units in synchronous systems and correspond to clock cycles.

In general, there are two types of scheduling: one is time-constrained scheduling and the other is resource-constrained scheduling.

Time-constrained scheduling is to minimize the cost of hardware bound by some specific allowed operation time. For example, in many digital signal processing (DSP) systems, the sampling rate of the input data stream dictates the maximum time allowed for carrying out a DSP algorithm on the present data sample before the next sample arrives.

On the other hand, the resource-constrained scheduling problem is encountered in many applications where we are limited by the silicon area. The constraint is usually given in terms of either a number of functional units or the total allocated area. When total area is given as a constraint, the scheduling algorithm determines the type of functional units used in the design. The goal of such an algorithm is to produce a design with the best possible performance but still meeting the given area constraint.

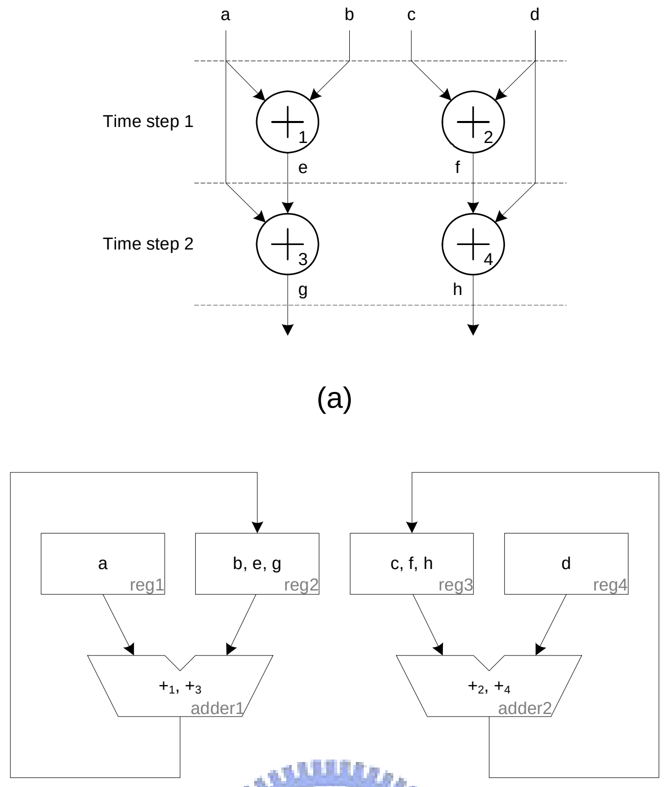
◆ Resource allocation & binding

Resource allocation is the process of determining how many and what types of hardware required to implement the desired behavior at lowest cost. The hardware resources consist primarily of functional units, memory modules, multiplexers, and communication datapaths.

Binding involves the mapping of the variables and operations in the scheduled DFG into the functional, storage, and interconnection units, while ensuring that the design behavior operates correctly on the selected set of components. For the every operation in the DFG, we need a functional unit that is capable of executing the operation. For every variable that is used across several time steps in the scheduled DFG, we need a storage unit to hold the data values during the variable’s lifetime.

Finally, for every data transfer in the DFG, we need a set of interconnection units for the transfer. Besides the design constraints imposed on the original behavior and represented in the DFG, additional constraints on the binding process are imposed by the type of hardware units selected. For example, a functional unit can execute only one operation in any given time step. Similarly, the number of multiple accesses to a storage unit during a control step is limited by the number of parallel ports on the unit.

Figure 4-2 illustrates the mapping of DFG into register transfer components. Figure 4-2 (a) show a scheduled DFG to be mapped and we assume that two adders and four registers are selected. Operation “$+_1$” and “$+_2$” cannot be mapped into the same adder because they must be performed in the same time step 1. On the other hand, operation “$+_1$” can share an adder with operation “$+_3$”, because they are carried out during different control steps. Thus, operation “$+_1$” and “$+_3$” are both mapped into adder1. Variables *a* and *e* must be stored separately because their values are need concurrently in time step 2. Register 1 and 2, where variables *a* and *e* reside, must be connected to the input ports of ADD1; otherwise, operation “$+_3$” will not be able to execute in adder1. Similarly, operation “$+_2$” and “$+_4$” are mapped to adder2. Note that there are several different ways of performing the binding. For example, we can map “$+_2$” and “$+_3$” to adder1 and “$+_1$” and “$+_4$” to adder2.



(a)

(b)

Figure 4-2 (a) Scheduled DFG; (b) mapped operation

4.2 Proposed Application Specific Programmable Processor Synthesis Flow

Not like traditional ASIP synthesis flow, we don’t make the ISA first. The composite FUs take advantages of the application characteristics. Figure 4-3 illustrates a bottom-up design flow described as follow. First, find out the proper datapath under constraints and applications. Second, summary the sub-functions and make a corresponding instruction set. Last, some other details of hardware are implemented. In this thesis, we have already implement ed the lower parts including Composite FUs Generation and Pipeline Design. The System Specification is the target goal and the DFG is the given application. The unsolved problem of Budgeting is waiting for advanced exploration.

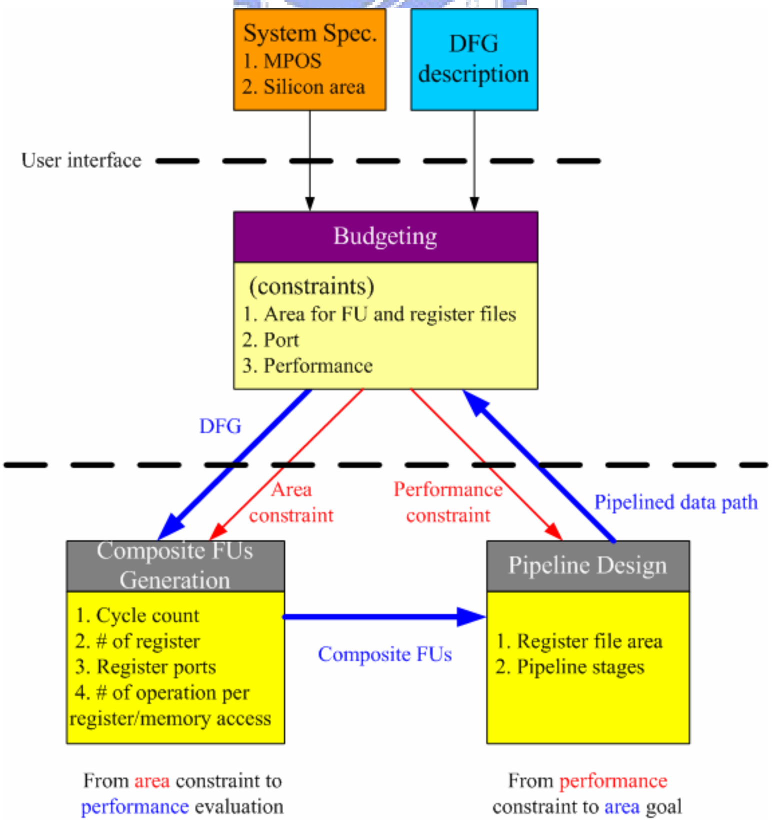


Figure 4-3 Proposed Application Specific Programmable Processor synthesis flow

◆ System Specification

The system may have some constraints. The most usual constraints are performance and area constraints. These two constraints define the analysis space of our composite FUs generator and pipeline design. Besides, the flow may plus some other constraints for other purposes if needs.

Performance constrains: in MOPS

Area constrains: in um^2

◆ Budgeting

Supervisors get system specification through some kind of user interface. Then they decide how to distribute the area for FU and RF. This is another domain of design space exploration.

◆ Composite FUs Generator

Get DFG description and the constraints from Budgeting as inputs. Then, use the composite FUs generator mentioned at chapter 3, we can analyze the DFG and report the corresponding OPC and register requirement. Then, list all results of analysis and send them into Pipeline Designer.

◆ Pipeline Designer

Get the analysis from the Composite FUs Generator and the constraints from Budgeting as inputs. Then, use the pipeline design flow described in section 3-2 and simulate the synthesis results with different pipeline stages. Finally, report the results including the pipeline stages and area in different MOPS to the supervisors to see if the iteration meets the specifications or not.

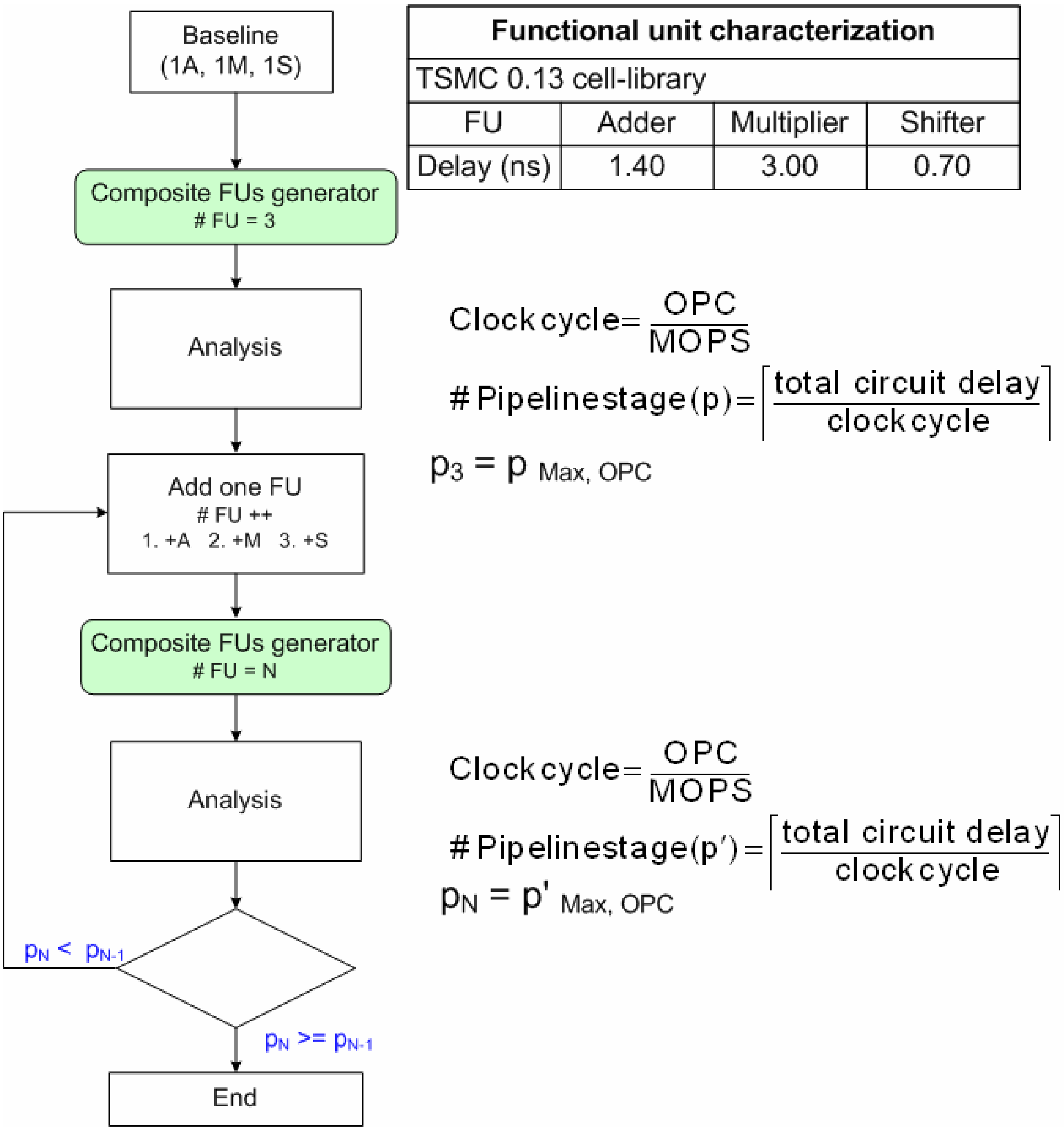
4.3 Composite FU Selection Flow

When we get a specific application, we would like to know what kind of composite FU is the best for this application for different purpose. In this section, we propose an iteration method called composite FU selection flow to find out the most suitable composite FU.

As the maximum OPC is limited by the characterization of the application itself, we develop a design methodology to find the minimal number of FUs to achieve the required performance. At the same time, the design methodology tries to minimize the number of pipeline stage to reduce thread overhead, i.e. context.

First, we perform FU characterization. Using TSMC 0.13um cell library and Synopsys DesignCompiler, the circuit delay of the 16-bit adder, 16-bit multiplier, and the 16-bit shifter is around 1.40ns, 3.00ns, and 0.70ns respectively. The design methodology begins with a baseline FU set, i.e. one adder, one multiplier, and one shifter. The baseline FU set and the target applications (described by DFG) are used as inputs to the composite FUs generator described in Section II.B.

The minimal number of pipeline stage (p_3) can be thus calculated as Analysis block in Figure 4-4. Then, one more specific functional unit is added, i.e. adding one more adder or multiplier or shifter. Again, after the analysis, if the resultant number of pipeline stage (p_N) is less than (p_{N-1}), we can continue adding one more FU to evaluate the resultant number of pipeline stage. On the other hand, if (p_N) is equal to or larger than (p_{N-1}), the procedure ends.



Functional unit characterization			
TSMC 0.13 cell-library			
FU	Adder	Multiplier	Shifter
Delay (ns)	1.40	3.00	0.70

$$\text{Clock cycle} = \frac{\text{OPC}}{\text{MOPS}}$$

$$\#\,\text{Pipelinestage}(p) = \left\lceil \frac{\text{total circuit delay}}{\text{clock cycle}} \right\rceil$$

$$p_3 = p_{\text{Max, OPC}}$$

$$\text{Clock cycle} = \frac{\text{OPC}}{\text{MOPS}}$$

$$\#\,\text{Pipelinestage}(p') = \left\lceil \frac{\text{total circuit delay}}{\text{clock cycle}} \right\rceil$$

$$p_N = p'_{\text{Max, OPC}}$$

Figure 4-4 Composite FU selection flow

5 Simulation Result

We show the improvement of the hardware utilization first. Then the area and power comparisons are listed in section 5-1 and 5-2.

5.1 Hardware Utilization Improvement and Area Comparison

After the software chain and application analysis flow set up, we start to find out the hardware utilization improving level of the composite FUs. In section 5.1.1, we make a benchmark suite of general media embedded applications. Then the target datapaths are described in section 5.1.2. The OPC is calculated and the single stage area is estimated. We use Synopsys DesignCompiler synthesis tool to synthesize the area of FU and RF separately. Section 5.1.3 shows the total area of different datapaths in different MOPS after pipeline design coordinating with IMT architecture.

5.1.1 Benchmark Suite

We choose nine kernels to make up a benchmark suite. 1~5 and 9 are including in the DSPstone [26] and BDTI [27] benchmark. 6 [28] and 7 [29] are used in the H.264 standard. 8 is used in the MP3 standard [30].

1. *16 taps finite impulse response*
2. *16 taps complex finite impulse response*
3. *16 taps linear-phase finite impulse response*
4. *1st order biquad filter*
5. *(8*8)*(8*1) matrix multiplication*
6. *8 points integer transform*
7. *8 points discrete cosine transform*
8. *12 points inverse modified discrete cosine transform*
9. *8 points fast fourier transform*

Table 5-1 outlines the operations profiling of these nine benchmarks. What we concern are operations of addition/subtraction, multiplication, and shift. Assume the input/output (load/store) operations are decoupled in the analysis. It means the primitive operations can get data from I/O whenever they want without any performance loss.

		# add	# mpy	# sht	# input	# output	# op
FIR	16-tap FIR	15	16	0	32	1	31
CFIR	16-tap complex FIR	62	64	0	64	2	126
LPFIR	16-tap linear-phase FIR	15	8	0	24	1	23
Biquad	1st-order IIR	8	9	0	12	1	17
Matrix	(8*8)*(8*1) matrix multiplication	56	64	0	72	8	120
IT	8-point integer transform (H.264)	32	0	10	8	8	42
DCT	8-point discrete cosine transform	29	12	9	15	8	50
IMDCT	12-point inverse modified DCT	21	11	9	11	12	41
FFT	16-point fast fourier transform	23	10	0	11	14	33
Total		261	194	28			483

Table 5-1 Operations profiling of the benchmark suite

5.1.2 Target Datapaths

We choose 5 kinds of datapths and list them below.

- **3 FUs including 1A 1M 1S**

  1. Scalar

  2. 3-way VLIW

  3. Composite FUs with 3 FUs: 6 arrangements

- **4 FUs including 2A 1M 1S**

  4. 4-way VLIW

  5. Composite FUs with 4FUs: 12 arrangements

Adding one more adder for scalar with 3 FUs is meaningless because of it would be leaved unused.

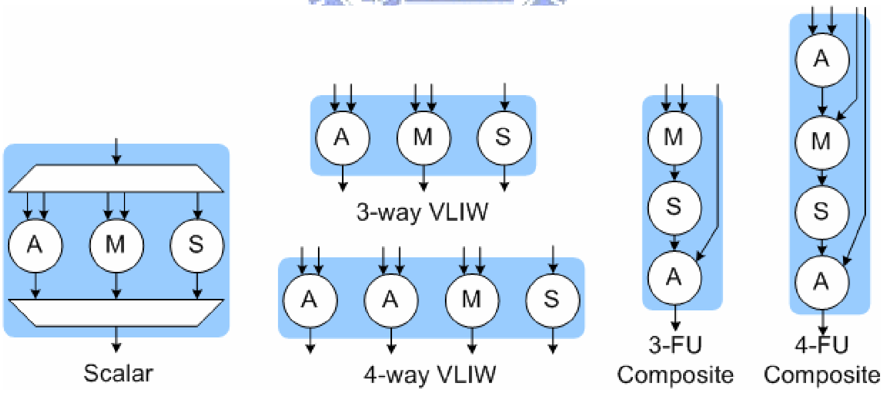


Figure 5-1 5 kinds of datapaths of the scalar, VLIW, and the composite FUs

Table 5-2 outlines the OPC for each datapaths corresponding to different applications. The OPC calculation is based on the assumption that all operations consume one clock cycle. Alternatively, for the scalar processor, the adder, or the multiplier, or the shifter takes one clock cycle to compute one result. Similarly, for the VLIW and the composite FUs, all the FUs can be activated simultaneously in one cycle. The OPC of the VLIW and the composite FUs is normalized to that of the scalar processor first and then the geometric mean is calculated among all input benchmarks. The OPC of scalar is 1.00, and OPC of the other datapaths are normalized to the scalar. The last column, called average OPC is the geometric mean of the former nine columns. The OPC of 3-way and 4-way VLIW are 1.65 and 2.12. The composite FU with best hardware utilization 1.35 and 1.62 for 3 FUs and 4FUs are **MSA** and **AMSA**. Although the average OPC of composite FUs lose to the OPC of the VLIW, the simulation result shows that there are composite FUs with OPC near to the OPC of VLIW, even better.

Figure 5-1 shows the configurations of 5 datapaths. The composite FUs MSA and AMSA with best OPC are chosen for further synthesis and comparison.

# operation/cycle			FIR	CFIR	LPFIR	Biquad	Matrix	IT	DCT	IMDCT	FFT	AVG
3 FU	Scalar		1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	VLIW		**1.82**	**1.91**	**1.53**	**1.70**	**1.79**	**1.31**	**1.61**	**1.86**	**1.43**	**1.65**
	Composite FUs	AMS	1.00	1.00	1.53	1.21	1.00	1.00	1.61	1.52	1.27	1.22
		ASM	1.00	1.00	1.53	1.21	1.00	1.00	1.52	1.28	1.27	1.18
		MAS	1.94	1.34	1.44	1.42	1.36	1.00	1.28	1.24	1.06	1.32
		MSA	**1.94**	**1.34**	**1.44**	**1.42**	**1.36**	**1.31**	**1.14**	**1.28**	**1.06**	**1.35**
		SAM	1.00	1.00	1.53	1.21	1.00	1.31	1.32	1.24	1.27	1.20
		SMA	1.94	1.34	1.44	1.42	1.36	1.31	1.09	1.21	1.06	1.33
4 FU	**VLIW**		**1.82**	**1.91**	**2.30**	**1.70**	**1.79**	**2.47**	**2.63**	**2.28**	**2.36**	**2.12**
	Composite FUs	AAMS	1.29	1.31	1.92	1.70	1.15	1.17	1.79	1.52	1.43	1.45
		AASM	1.29	1.31	1.92	1.70	1.15	1.17	1.67	1.28	1.43	1.42
		AMAS	1.35	1.34	2.88	1.70	1.36	1.17	1.79	1.52	1.43	1.56
		AMSA	**1.35**	**1.34**	**2.88**	**1.70**	**1.36**	**1.62**	**1.67**	**1.64**	**1.43**	**1.62**
		ASAM	1.29	1.31	1.92	1.70	1.15	1.62	1.56	1.32	1.43	1.46
		ASMA	1.35	1.34	2.88	1.70	1.36	1.62	1.56	1.32	1.43	1.57
		MAAS	1.94	1.97	1.44	1.89	1.67	1.17	1.39	1.32	1.18	1.52
		MASA	1.94	1.97	1.44	1.89	1.67	1.62	1.32	1.37	1.18	1.57
		MSAA	1.94	1.97	1.44	1.89	1.67	1.62	1.19	1.32	1.18	1.55
		SAAM	1.29	1.31	1.92	1.70	1.15	1.62	1.39	1.21	1.43	1.43
		SAMA	1.35	1.34	2.88	1.70	1.36	1.62	1.39	1.21	1.43	1.53
		SMAA	1.94	1.97	1.44	1.89	1.67	1.62	1.14	1.17	1.18	1.52

Table 5-2 Operations per cycle

Table 5-3 shows the further comparisons among the Scalar/VLIW/Composite FUs. The 2nd to the 8th columns shows the various the OPC of each architecture regarding the various benchmarks. For the composite FUs, two kinds of OPC are displayed. One is of the MSA-ordered/AMSA-ordered composite FUs and the other is, for each benchmark, the best OPC of all possible composite FUs. For example, the AMS-ordered composite FU results in the best OPC regarding the LPFIR benchmark. Although the MSA-ordered/AMSA-ordered composite FUs is of single-issue processor, their OPCs in all benchmarks are greater than one (of an average of 1.35/1.62) and thus it effectively increases datapath utilization. Besides, the best OPC of the composite FUs (either 3-FU or 4-FU) is comparable to that of the VLIW and even greater than that of the VLIW. The composite FUs have an average performance loss of about 20% in comparison with the VLIW.

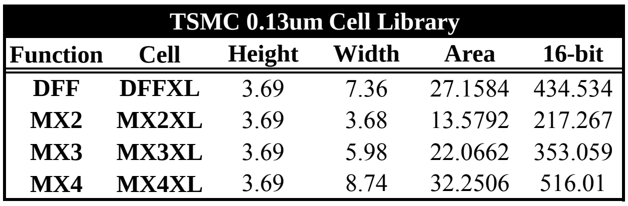
Benchmark	Scalar	3-way VLIW	Composite FU		4-way VLIW	Composite FU	
			MSA	Best		AMSA	Best
FIR	1.0	1.82	1.94	1.94	1.82	1.35	1.94
CFIR	1.0	1.91	1.34	1.34	1.91	1.34	1.97
LPFIR	1.0	1.53	1.44	1.53	2.30	2.88	2.88
Biquad	1.0	1.70	1.42	1.42	1.70	1.70	1.89
Matrix	1.0	1.79	1.36	1.36	1.79	1.36	1.67
IT	1.0	1.31	1.31	1.31	2.47	1.62	1.62
DCT	1.0	1.61	1.14	1.61	2.63	1.67	1.79
IMDCT	1.0	1.86	1.28	1.52	2.28	1.64	1.64
FFT	1.0	1.43	1.06	1.27	2.36	1.43	1.43
Average	1.0	1.65	1.35	-	2.12	1.62	-

Table 5-3 OPC comparison

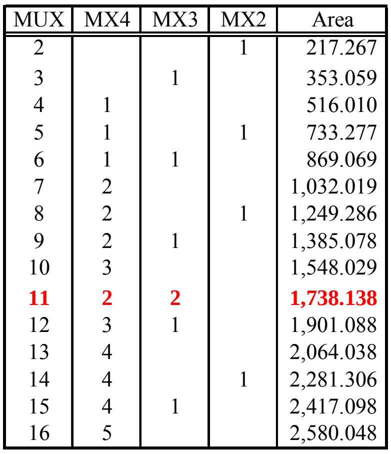
5.1.3 Register File Model Modification

The RF area model in section 2-5 is used in TSMC 0.18um cell library. The area of the 4-to-1 MUX is about three times larger than area of the 2-to-1 MUX. Using 2-to-1 MUX to estimate all the N-to-1 MUX is acceptable while the area is almost the same and the errors are the slightly difference of timing and control network area.

Table 5-4 (a) shows the area of basic MUX cells in TSMC 0.13um cell library. The area of 4-to-1 MUX is no longer three times the area of 2-to-1 MUX, it is about 2.5 times. The RF model must be modified. Otherwise, the estimation area could be over-estimated.



TSMC 0.13um Cell Library					
Function	**Cell**	**Height**	**Width**	**Area**	**16-bit**
DFF	**DFFXL**	3.69	7.36	27.1584	434.534
MX2	**MX2XL**	3.69	3.68	13.5792	217.267
MX3	**MX3XL**	3.69	5.98	22.0662	353.059
MX4	**MX4XL**	3.69	8.74	32.2506	516.01

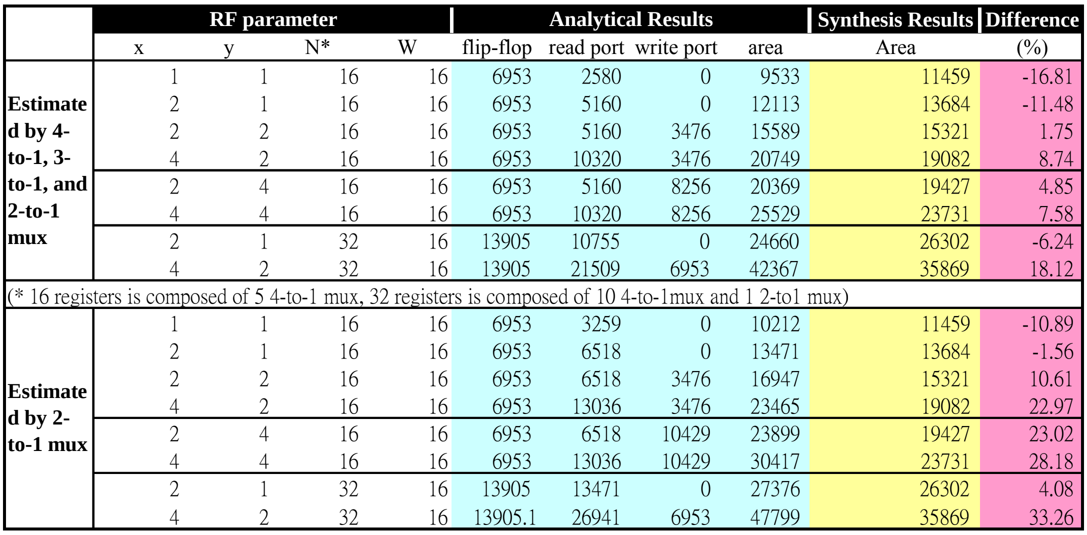


MUX	MX4	MX3	MX2	Area
2			1	217.267
3		1		353.059
4	1			516.010
5	1		1	733.277
6	1	1		869.069
7	2			1,032.019
8	2		1	1,249.286
9	2	1		1,385.078
10	3			1,548.029
11	**2**	**2**		**1,738.138**
12	3	1		1,901.088
13	4			2,064.038
14	4		1	2,281.306
15	4	1		2,417.098
16	5			2,580.048

Table 5-4 (a) Area of basic MUX cells in TSMC 0.13um cell library (unit: um2)

(b)Area of modified MUX model (unit: um2)

We use 2-to-1 MUX, 3-to-1 MUX, and 4-to-1 MUX to form the N-to-1 MUX and to find the minimum area as Table 5-4 (b) shows. Table 5-5 illustrates that the difference between analytical model and synthesis result. The upper part of the annotation is the results of modified RF model, and the lower part of the annotation is the results of the original RF model. It shows that our modified method have lower errors from -6% ~ 20%. Except that the difference with few ports is under-estimated because of the MUX area is relatively small.



	RF parameter				**Analytical Results**				**Synthesis Results**	**Difference**
	x	y	N*	W	flip-flop	read port	write port	area	Area	(%)
Estimated by 4-to-1, 3-to-1, and 2-to-1 mux	1	1	16	16	6953	2580	0	9533	11459	-16.81
	2	1	16	16	6953	5160	0	12113	13684	-11.48
	2	2	16	16	6953	5160	3476	15589	15321	1.75
	4	2	16	16	6953	10320	3476	20749	19082	8.74
	2	4	16	16	6953	5160	8256	20369	19427	4.85
	4	4	16	16	6953	10320	8256	25529	23731	7.58
	2	1	32	16	13905	10755	0	24660	26302	-6.24
	4	2	32	16	13905	21509	6953	42367	35869	18.12
(* 16 registers is composed of 5 4-to-1 mux, 32 registers is composed of 10 4-to-1mux and 1 2-to1 mux)										
Estimated by 2-to-1 mux	1	1	16	16	6953	3259	0	10212	11459	-10.89
	2	1	16	16	6953	6518	0	13471	13684	-1.56
	2	2	16	16	6953	6518	3476	16947	15321	10.61
	4	2	16	16	6953	13036	3476	23465	19082	22.97
	2	4	16	16	6953	6518	10429	23899	19427	23.02
	4	4	16	16	6953	13036	10429	30417	23731	28.18
	2	1	32	16	13905	13471	0	27376	26302	4.08
	4	2	32	16	13905.1	26941	6953	47799	35869	33.26

Table 5-5 Modified RF model in TSMC 0.13um cell library

5.1.4 Area Comparison

◆ Estimation Results

From the register requirement conducted from the composite FUs generator, using the modified RF model and get the results in Table 5-6. The “# port” column shows the read/write port number. The “MAX” column stands for the maximal of register requirement of architectures. The “Est. Area” column means the estimation according to register number form MAX. We can see that the area of 3-way VLIW is the largest one in the 3-FU fields, so as to the area of 4-way VLIW is the largest one in the 4-FU fields.

For more general purpose and simulation simplicity, we round the MAX into 16 for convenience. This is reasonable because the variation of register requirement is highly different according to applications and scheduling methods. We can see that the area pressure of VLIW is relatively high to scalar and the composite FUs. The area of composite FU is slightly larger than scalar.

			FIR	CFIR	LPFIR	Biquad	Matrix	IT	DCT	IMDCT	FFT	# port	MAX	Est. Area	final	Area
3 FU	Scalar		2	4	3	6	8	11	10	9	7	2R/1W	11	8256	16	12113
	VLIW		**2**	**3**	**3**	**4**	**4**	**9**	**10**	**8**	**7**	5R/3W	**10**	**15616**	**16**	**25502**
	Composite FUs	AMS	2	4	3	4	8	16	12	7	10	3R/1W	16	14693	16	14693
		ASM	2	4	3	4	8	16	11	7	10		16	14693		
		MAS	1	3	3	3	4	16	10	8	12		16	14693		
		MSA	**1**	**3**	**3**	**3**	**4**	**14**	**10**	**8**	**12**		**14**	**12927**		
		SAM	2	4	3	4	8	14	11	7	10		14	12927		
		SMA	1	3	3	3	4	14	10	7	12		14	12927		
4 FU	**VLIW**		**2**	**3**	**4**	**4**	**4**	**10**	**10**	**7**	**10**	7R/4W	**10**	**20342**	**16**	**33269**
	Composite FUs	AAMS	3	3	5	6	8	14	10	8	8	4R/1W	14	15209	16	17273
		AASM	3	3	5	6	8	14	10	8	8		14	15209		
		AMAS	3	3	1	3	6	14	10	8	8		14	15209		
		AMSA	**3**	**3**	**1**	**3**	**6**	**12**	**10**	**7**	**8**		**12**	**12819**		
		ASAM	3	3	5	6	8	12	10	8	8		12	12819		
		ASMA	3	3	1	3	6	12	10	8	8		12	12819		
		MAAS	3	2	4	3	4	14	10	9	11		14	15209		
		MASA	3	2	4	3	4	12	10	9	11		12	12819		
		MSAA	3	2	4	3	4	10	10	9	11		11	11732		
		SAAM	3	3	5	6	8	10	10	7	8		10	10537		
		SAMA	3	3	1	3	6	10	10	7	8		10	10537		
		SMAA	3	2	4	3	4	10	10	7	11		11	11732		

Table 5-6 Register requirement and estimated RF area

◆ Synthesis Results

● Un-pipelined

Figure 5-2 explains the area cost of the various architectures under the performance requirement of 100MOPS, 150MOPS, 200MOPS, and 300MOPS respectively. We use Synopsys DesignCompiler with the TSMC 0.13um cell-library to synthesize the various architectures. All synthesized architectures are un-pipelined.

The composite FUs (MSA or AMSA) save about 10% ~ 25% of area compared to the VLIW regarding 100/150/200 MOPS due to less number of ports of centralized register file. However, when the performance requirement is above 300 MOPS, the composite FUs have to be pipelined to achieve the comparable performance to the VLIW.

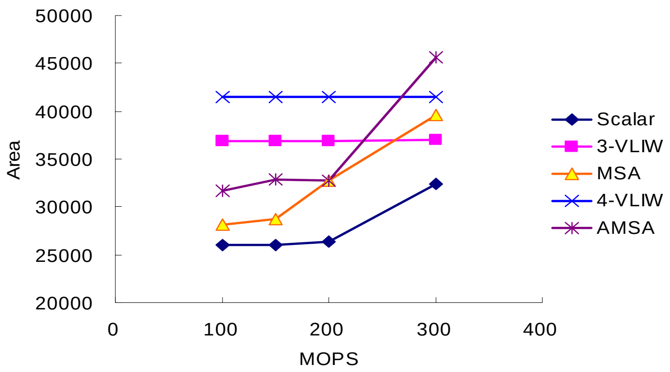


Figure 5-2 Area analysis of single stage

The total area is composed of the area of FUs and the area of RF. The area of RF changes slightly with the frequency increasing, but the area of FUs grows exaggeratedly when the frequency increases and the combinational delay constraint decrease. Actually, the cost of the RF area of the composite is quite equal to the cost of the scalar.

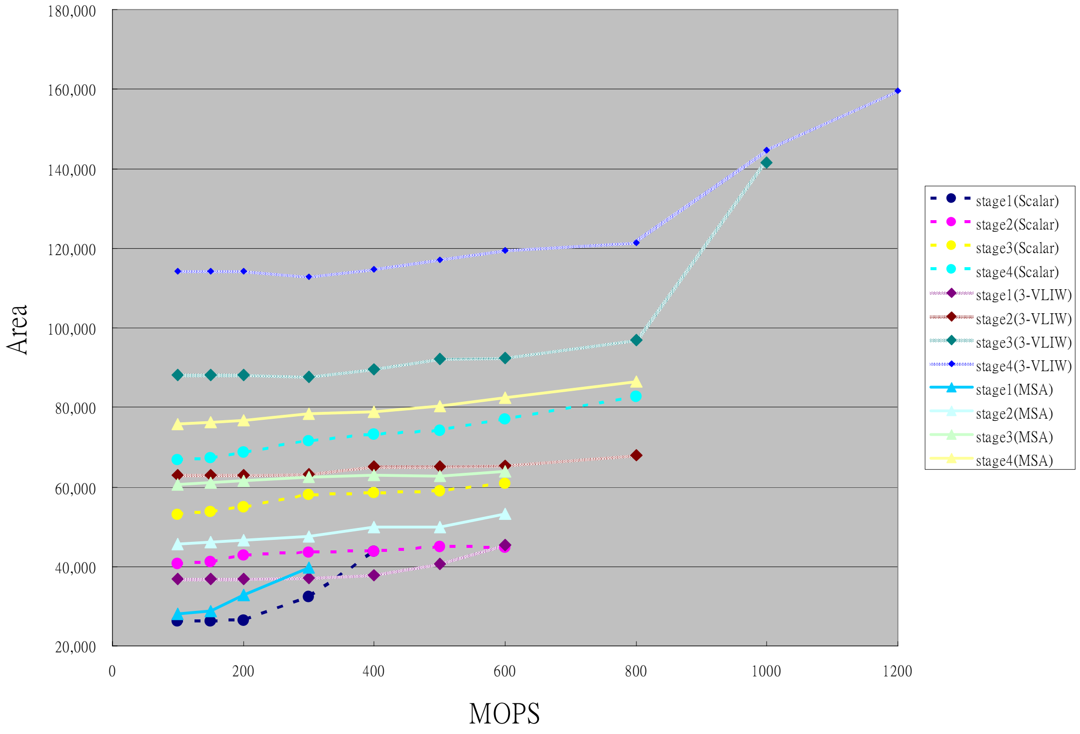


Figure 5-3 Area analysis with 1 to 4 pipeline stages (3 FUs)

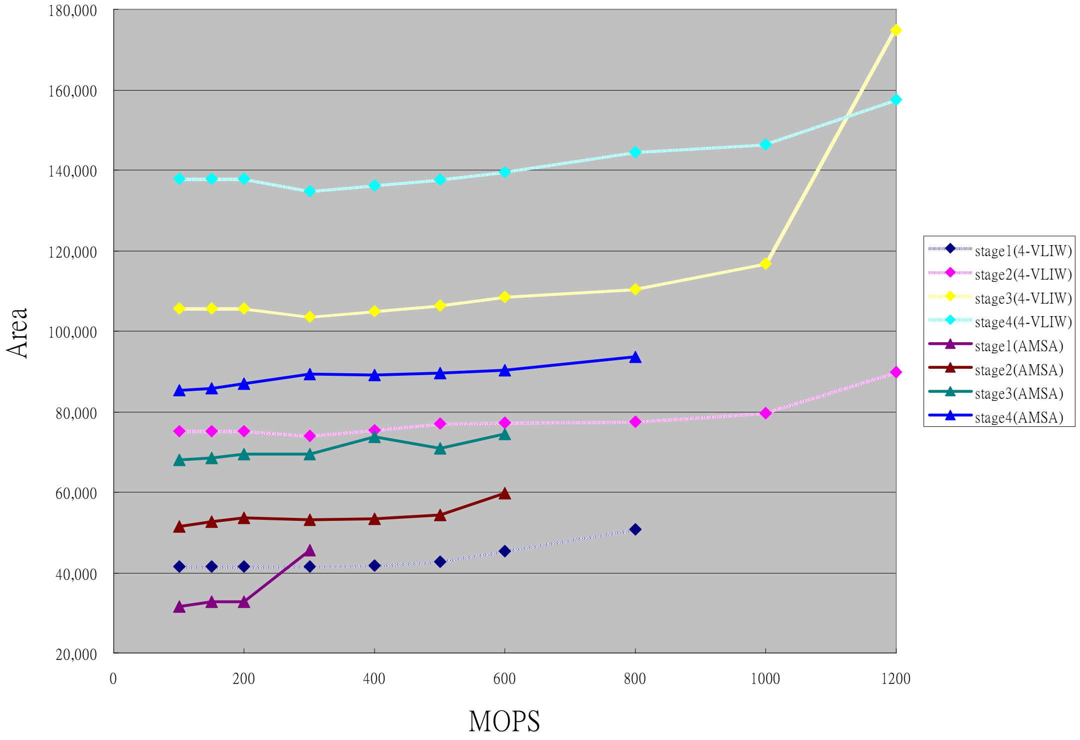


Figure 5-4 Area analysis with 1 to 4 pipeline stages (4 FUs)

- **Pipelined**

After pipeline design introduced in section 3-2, we use ***N*** threads IMT to hide ***N*** stages instruction latency totally. For each thread, there must be a thread register file supported to keep the functionality of IMT architecture.

Figure 5-3 shows the area analysis of the datapaths with 3 FUs and 1 to 4 pipeline stages. These lines stand for total area composed of the area of the FUs, the pipeline register, and the thread register files. Figure 5-4 shows the area analysis of the datapaths with 4 FUs and 1 to 4 pipeline stages.

We can see that if VLIW using IMT, the cost is exaggeratedly large than composite FU and scalar. On the other hand, the composite FUs cost a slightly more area than the scalar by around 10%.

Because the peak performance of the composite FUs is lower than the VLIW under same FU resource. VLIW cooperate with IMT can reach specific MOPS goal with less pipeline stages and thread number. However, to our knowledge, there are several ways to reduce the multithread-related cost such as the register file architecture using master latch sharing described later.

We will recommend the designers if they want to use VLIW architecture, they should use some other techniques to hide instruction latency under limited area constraint. Nevertheless, these techniques need overhead of hardware complexity and the re-compile of the software efforts.

5.2 Power Estimation

In this section, we quantify the reduction of register accesses. Then we conduct an experiment to see the degree of power saving.

5.2.1 Register Accesses Per Operation

The total accesses of the composite FUs are reduced because of several operations performed in single instruction. It apparently cuts the read and write accesses.

# register access/operation(actual)			FIR		CFIR		LPFIR		Biquad		Matrix		IT		DCT		IMDCT		FFT		AVG
			t	N	t	N	t	N	t	N	t	N	t	N	t	N	t	N	t	N	N
3 FU	Scalar		3.00	1.00	3.00	1.00	3.00	1.00	3.00	1.00	3.00	1.00	2.76	1.00	2.82	1.00	2.78	1.00	3.00	1.00	1.00
	VLIW		3.00	1.00	3.00	1.00	3.00	1.00	3.00	1.00	3.00	1.00	2.76	1.00	2.82	1.00	2.78	1.00	3.00	1.00	1.00
	Composite FUs	AMS	3.00	1.00	3.00	1.00	2.30	0.77	2.65	0.88	3.00	1.00	2.76	1.00	2.06	0.73	2.10	0.75	2.58	0.86	0.88
		ASM	3.00	1.00	3.00	1.00	2.30	0.77	2.65	0.88	3.00	1.00	2.76	1.00	2.14	0.76	2.34	0.84	2.58	0.86	0.90
		MAS	2.03	0.68	2.49	0.83	2.39	0.80	2.41	0.80	2.47	0.82	2.76	1.00	2.28	0.81	2.39	0.86	2.77	0.92	0.83
		MSA	**2.03**	**0.68**	**2.49**	**0.83**	**2.39**	**0.80**	**2.41**	**0.80**	**2.47**	**0.82**	**2.29**	**0.83**	**2.46**	**0.87**	**2.23**	**0.80**	**2.77**	**0.92**	**0.82**
		SAM	3.00	1.00	3.00	1.00	2.30	0.77	2.65	0.88	3.00	1.00	2.29	0.83	2.34	0.83	2.39	0.86	2.58	0.86	0.89
		SMA	2.03	0.68	2.49	0.83	2.39	0.80	2.41	0.80	2.47	0.82	2.29	0.83	2.54	0.90	2.17	0.78	2.77	0.92	0.82
4 FU	**VLIW**		3.00	1.00	3.00	1.00	3.00	1.00	3.00	1.00	3.00	1.00	2.76	1.00	2.82	1.00	2.78	1.00	3.00	1.00	1.00
	Composite FUs	AAMS	2.55	0.85	2.52	0.84	2.04	0.68	2.18	0.73	2.73	0.91	2.41	0.87	1.90	0.68	2.05	0.74	2.24	0.75	0.78
		AASM	2.55	0.85	2.52	0.84	2.04	0.68	2.18	0.73	2.73	0.91	2.41	0.87	1.98	0.70	2.28	0.82	2.24	0.75	0.79
		AMAS	2.48	0.83	2.49	0.83	1.70	0.57	2.18	0.73	2.47	0.82	2.41	0.87	1.90	0.68	2.05	0.74	2.24	0.75	0.75
		AMSA	**2.48**	**0.83**	**2.49**	**0.83**	**1.70**	**0.57**	**2.18**	**0.73**	**2.47**	**0.82**	**1.95**	**0.71**	**1.96**	**0.70**	**1.98**	**0.71**	**2.24**	**0.75**	**0.73**
		ASAM	2.55	0.85	2.52	0.84	2.04	0.68	2.18	0.73	2.73	0.91	1.95	0.71	2.04	0.72	2.23	0.80	2.24	0.75	0.77
		ASMA	2.48	0.83	2.49	0.83	1.70	0.57	2.18	0.73	2.47	0.82	1.95	0.71	2.04	0.72	2.23	0.80	2.24	0.75	0.75
		MAAS	2.00	0.67	2.02	0.67	2.39	0.80	2.06	0.69	2.20	0.73	2.41	0.87	2.17	0.77	2.23	0.80	2.47	0.82	0.76
		MASA	2.03	0.68	2.02	0.67	2.39	0.80	2.06	0.69	2.20	0.73	1.95	0.71	2.24	0.79	2.21	0.80	2.47	0.82	0.74
		MSAA	2.03	0.68	2.02	0.67	2.39	0.80	2.06	0.69	2.20	0.73	1.95	0.71	2.39	0.85	2.16	0.78	2.47	0.82	0.74
		SAAM	2.55	0.85	2.52	0.84	2.04	0.68	2.18	0.73	2.73	0.91	1.95	0.71	2.19	0.78	2.37	0.85	2.24	0.75	0.78
		SAMA	2.48	0.83	2.49	0.83	1.70	0.57	2.18	0.73	2.47	0.82	1.95	0.71	2.19	0.78	2.37	0.85	2.24	0.75	0.76
		SMAA	2.03	0.68	2.02	0.67	2.39	0.80	2.06	0.69	2.20	0.73	1.95	0.71	2.46	0.87	2.26	0.81	2.47	0.82	0.75

Table 5-7 Register accesses per operation

Table 5-7 outlines the register accesses per operation. First, count all accesses for each case. The accesses of adder of scalar and VLIW are 2R/1W. The accesses of multiplier of scalar and VLIW are 2R/1W. The accesses of shifter of scalar and VLIW are 1R/1W. The accesses of composite FUs with 3 FUs (1A1M1S) are 3R/1W. The accesses of composite FUs with 4 FUs (2A1M1S) are 4R/1W. Furthermore, when the composite FUs use sub-functions to execute the applications, the corresponding accesses are taken into consideration. Then, all the data is normalized to scalar 1.00. The last column is the geometric mean.

Table 5-8 shows the respective register access per operation of the various FU configurations regarding the used benchmarks. On average, the 3-FU and the 4-FU composite FUs reduce about 18% and 27% of register accesses per operation compared to the scalar and the VLIW.

Benchmark	**Scalar**	**3-way VLIW**	**Composite FU (MSA)**	**4-way VLIW**	**Composite FU (AMSA)**
FIR	1.00	1.00	0.68	1.00	0.83
CFIR	1.00	1.00	0.83	1.00	0.83
LPFIR	1.00	1.00	0.80	1.00	0.57
Biquad	1.00	1.00	0.80	1.00	0.73
Matrix	1.00	1.00	0.82	1.00	0.82
IT	1.00	1.00	0.83	1.00	0.71
DCT	1.00	1.00	0.87	1.00	0.70
IMDCT	1.00	1.00	0.80	1.00	0.71
FFT	1.00	1.00	0.92	1.00	0.75
Average	1.00	1.00	**0.82**	1.00	**0.73**

Table 5-8 Outline of register accesses per operation

5.2.2 Target Simulated Architecture

Figure 5-5 is an overview when the FU and RF are mapped into a real architecture. Assume that there are instruction memory, data memory, and load/store unit which can help the application really work.

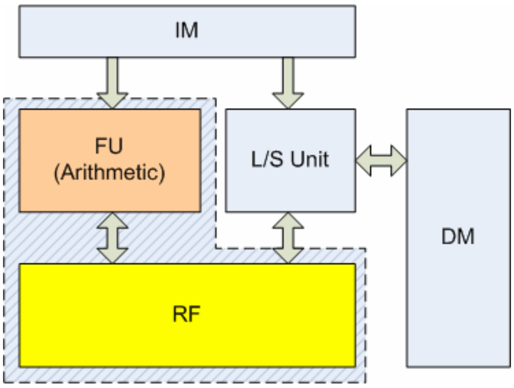


Figure 5-5 Simulated architecture

The shadow area surrounded by the dotted line includes FU and RF. These two parts are what we concern. In our experiment, we replace these two part using the FU and RF pairs of scalar, VLIW, and composite FU (MSA) to keep track of the power consumption. Remember that the L/S is independent of the FU because of the assumption made before for software analysis.

5.2.3 Ping-Pong Register File

Based on the analysis assumption and results before, the load/store handling is decoupled. Now we want to estimate power carefully, so we should reconsider of the load/store effect again because of the power estimation should be closed to real situation.

There is two methods to handle the load/store operation.

(1) Increase additional RF port to let the FUs get the right data transparently in the execution sequence.

(2) Increase load/store cycles.

The latter is simple, but it needs extra cycles because of the I/O pattern of register files must be considered.

Without changing the scenario, we use ping-pong register to keep simplicity without opening new ports (2) and performance loss (1).

Assume the load bandwidth is full capable of getting enough data. For example, when executing a MAC needs to load a coefficient and a data input, the load bandwidth is twice the arithmetic bandwidth. However, there is some control hardware overhead if the register file is capable of two values written at the same time using the same MUX.

We mirror a 16-bit RF with 16 registers and use the ping-pong mode execution. As Figure 5-6 shows, write ports of ping-pong RFs are interleaved access from load/store unit or FU. The ports of FU and RF for different datapaths are listed below.

FU (Input / Output)

Scalar: 2 / 1

VLIW: 5 / 3

MSA: 3 / 1

RF (Read / Write)

Scalar: 2 / 1

VLIW: 5 / 3

MSA: 3 / 1

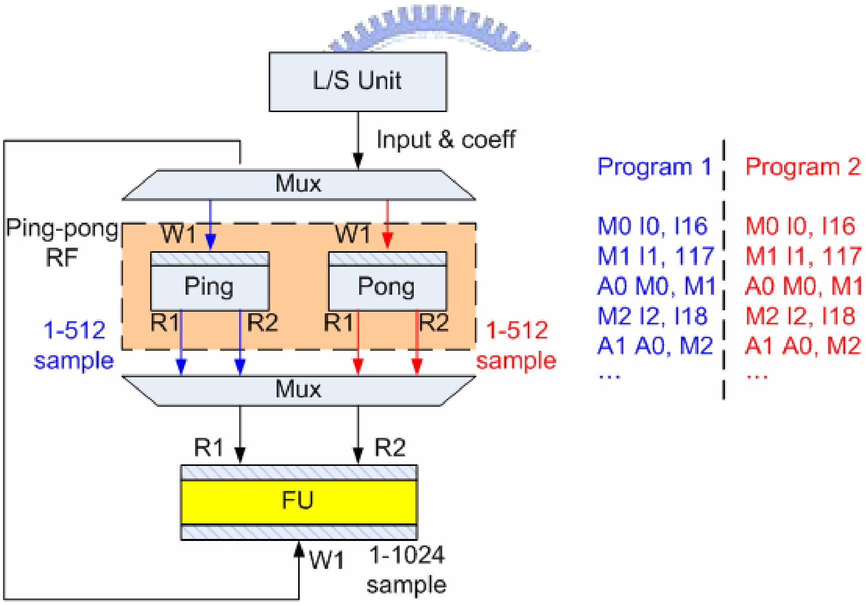


Figure 5-6 Data flow of FU and RF

Besides, power is strongly related to activity pattern. The power consumption is almost from the transition of the logic network. Figure 5-7 shows the access pattern derived from Figure 5-6. We can see that the hardware can get right data in corresponding cycles. The ping-pong execution covers the problems derived from load/store cooperation. These two threads make up the total application execution.

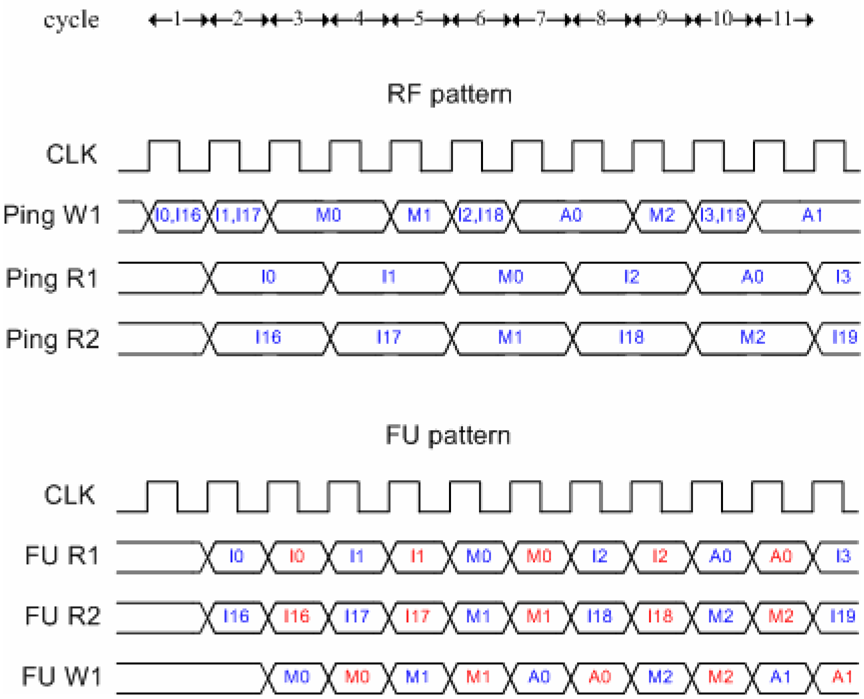


Figure 5-7 Access pattern of FU and RF

5.2.4 Simulation Results

This experiment measures the power consumption of the scalar, the 3-way VLIW, and the MSA-ordered composite FU. We take Synopsys PrimePower as simulation tool. It simulated the power consumption of gate-level files.

The application is executed in streaming process. And the Remez 16-tap FIR with 1,024 Gaussian-distributed random input patterns is used as the test real application. Table 5-9 illustrates the total cycles for test application of each datapaths.

Number of cycle	Scalar	MSA	3-VLIW
random input	1024	1024	1024
remez filter	16	16	16
FIR(once)	31	16	17
total	31744	16384	17408

Table 5-9 Execution cycles

First of all, let us see the factors which may affect the power *P*. Three main factors are the operation frequency *f*, capacitance *C*, and the voltages *V*. Their relations to power is $P \propto f \cdot C \cdot V^2$. The voltage *V* is constant in TSMC 0.13 cell library. The *f* is related to the cycle time. The capacitance *C* is related to the area. Table 5-10 is derived from the OPC analyzed before and the performance goal. It is used as the synthesis timing constraint. And Table 5-11 shows the synthesis area based on the cycle time in Table 5-10. Because the critical path has limited synthesis timing constraint, the area of MSA-ordered composite FU explodes in 400MOPS.

Cycle time (unit:ns)	Scalar	MSA	3-VLIW
100MOPS	10.00	19.38	18.23
200MOPS	5.00	9.69	9.12
400MOPS	2.50	4.84	4.56

Table 5-10 Cycle time

Area (unit: um2)	Scalar				MSA				3-VLIW			
	FU	RF			FU	RF			FU	RF		
		Ping	Pong	Sum		Ping	Pong	Sum		Ping	Pong	Sum
100MOPS	12724	16047	16047	32094	12296	18892	18892	37784	12289	31733	31733	63466
200MOPS	13012	16047	16047	32094	12875	18892	18892	37784	12289	31733	31733	63466
400MOPS	30204	16047	16047	32094	21920	18892	18892	37784	13092	31733	31733	63466

Table 5-11 Synthesis area

Table 5-12 shows the power consumption of every part we concern. The power of MSA is smaller in most case because of the longer cycle time, lower frequency. The power of FU part of composite FU grows exaggeratedly in 400MOPS because of the area explosion.

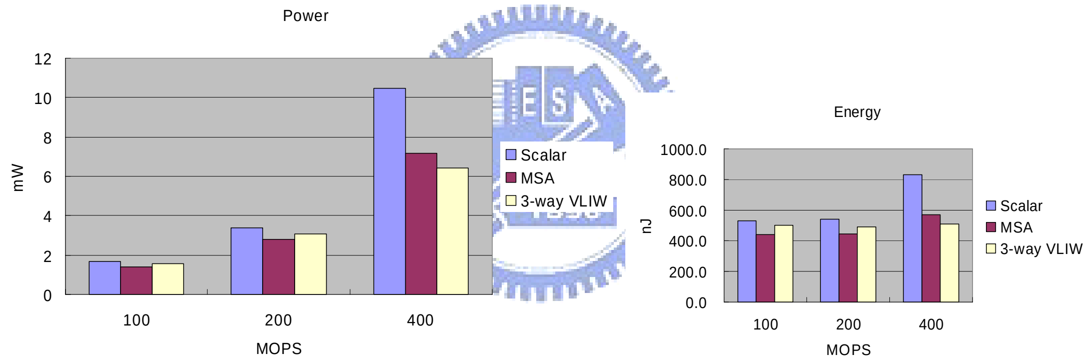
Power (unit: mW)	Scalar					MSA					3-VLIW				
	FU	RF			Total	FU	RF			Total	FU	RF			Total
		Ping	Pong	Sum			Ping	Pong	Sum			Ping	Pong	Sum	
100MOPS	0.454	0.608	0.604	1.212	1.666	0.264	0.566	0.560	1.126	1.390	0.234	0.701	0.703	1.404	1.638
200MOPS	0.994	1.208	1.202	2.410	3.404	0.579	1.120	1.107	2.227	2.806	0.460	1.405	1.403	2.808	3.268
400MOPS	5.670	2.406	2.397	4.803	10.473	2.699	2.237	2.228	4.465	7.164	1.203	2.812	2.807	5.619	6.822

Table 5-12 Power consumption

Compared with the scalar and the VLIW, the composite FU saves 16.5% ~ 31.6% of power consumption under the 100 ~ 200 MOPS performance requirement as Table 5-13 shows. The power saving comes from the less number of RF's port required by the composite FUs and less register access per operation of the composite FUs. Figure 5-8 shows the bar charts of power and energy comparison.

Power improve (Normalized to scalar) (%)					
MSA			3-VLIW		
FU	RF	Total	FU	RF	Total
-41.8	**-7.1**	**-16.6**	**-48.4**	**15.8**	**-1.7**
-41.7	**-7.6**	**-17.6**	**-53.7**	**16.5**	**-4.0**
-52.4	**-7.0**	**-31.6**	**-78.8**	**17.0**	**-34.9**

Table 5-13 Power improvement



(a)

(b)

Figure 5-8 Comparison of (a) power (b) energy

6 Summary & Future Works

In this thesis, we propose the composite functional units which cascades all the primitive FUs in a customized order by analyzing the DFG (data-flow graph) of the target applications to improve datapath utilization. The composite FU with 3 primitive functional units achieves an OPC of 1.35 on average and has comparable OPCs to that of the VLIW in several benchmarks.

Besides, the composite FU reduces 10% to 25% of area compared with the VLIW and saves 16.5% to 31.6% of power consumption compared with both the scalar and the VLIW under the performance target ranging from 100 to 300 MOPS.

Although the composite FUs may result in long critical path, pipelining technique can be applied to raise the clock rate feasibly. A flexible pipelining design flow is also proposed to assist in FU pipelining. Additionally, the interleaved multithreading can be applied to hide pipeline latency totally if enough number of threads is supported.

The area comparison in section 5-1 shows that the hardware cost of the thread increase is small for composite FU, so the IMT architecture is good for the composite FUs. Relatively, the hardware cost of VLIW cooperated with IMT is too high. Some other methods to hide instruction latency must be taken.

In chapter 4, we proposed an Application Programmable Processor Synthesis Flow to design a processor based on composite FUs. The composite FU selection flow helps user to find a proper composite FU for a specific application.

◆ Future Work

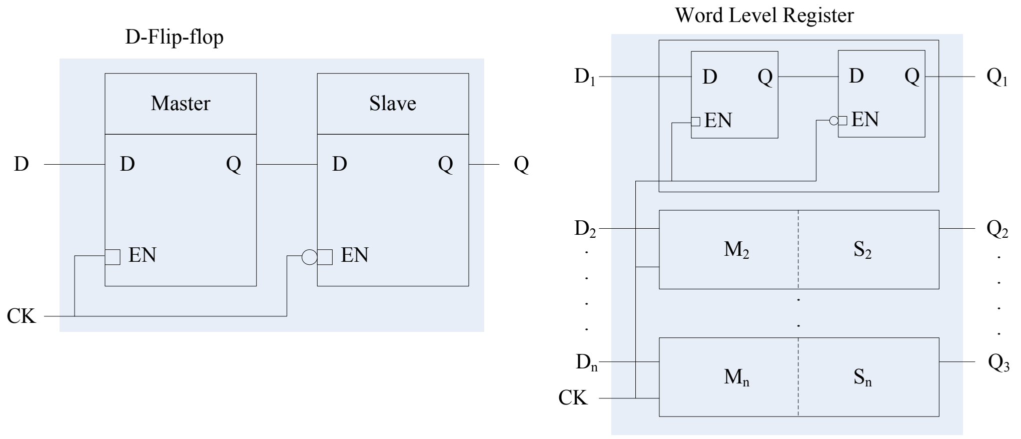
● Thread Register File Reduction

Although multithreading, in our experience, will incur large context overhead. Because of the IMT architecture needs a thread register file for each thread, the area and overhead of hardware increase with thread number. There are other approaches such as the register file architecture using master latch sharing described in [] to lessen the side effect.   In the future, we will continue studying on how to reduce the multithreading-incurred overhead.

■ Share master latching

[31] introduces a method of reducing area and power consumption of a synthesizable register tile by using a single master latch shared by a number of slaves. Simulation results show that, depending on the size of the register tile, reduction of power consumption of more than 50% is achievable.

Data stores are an important power critical part of resource sharing architectures [7] or processing units, like application specific instruction set processors (ASIPs). They are preferably implemented as a synthesizable register file described as part of the design on register transfer level, because of a high effort required for timing verification of RAM.



(a) (b)

Figure 6-1 (a) D Flip-filop (b) Word Level Register

Figure 6-1 (a) is a typical D flop-flop. It is composed of master latch and slave latch. Several D flip-flops make up a word level register in Figure 6-1 (b). The real data is stored in the slave latch.

This method also aims at reduction of capacitance connected to the data bus.

This is achieved by splitting up the master-slave flip-flops into the master latches and the slave latches. If clock gating is applied, slave latches of registers in the register tile can share one master latch. Thus the number of master latches connected to the data bus is decreased. Additionally savings in area can he expected.

Figure 6-2 (a) shows conventional register file with flip-flops and Figure 6-2 (b) shows modified register file with shared master latches.

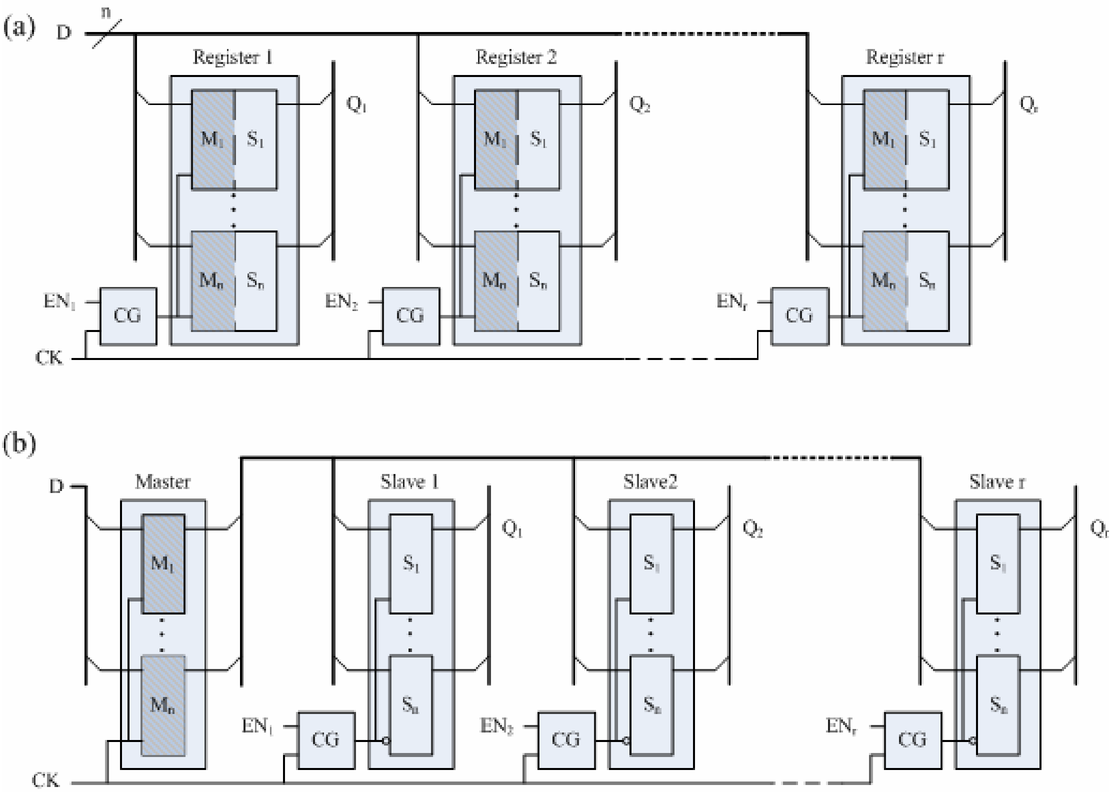


Figure 6-2 (a) Conventional register file with flip-filops

(b) Modified register file with shared master latches.

- **Fused FU and VLIW Cooperation**

We demonstrate that the composite FU effectively increases datapath utilization in this thesis. However, the long critical path will incur other overheads, i.e. pipeline latency or register file complexity. Another research direction will focus on the fused FU such as MAC (multiply and add). We can cascade two or three primitive operations to be a fused FU that is frequently used and then design a VLIW processor with several fused FUs as primitive FUs. Compared to the conventional VLIW processors with the same computing resources, multi-issue fused FUs still demand less number of register file ports. We expect that the multi-issue fused FUs will explore more operation parallelism and thus achieve higher OPC. At the same time, it can avoid long critical path.

Merge Shifter Into Multiplier

Because the shift operations are small relatively to the total operations, the shifter is usually idle. Shift operations often occur between different application and data transformation for alignment. Maybe we can merge the shifter into the multiplier by by-passing the corresponding bits form the product of multiplier with little hardware effort. By the way, the software analysis may need modification.

Reference

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture – A Quantitative Approach*, 4th edition, Morgan Kaufmann, 2006

[2] *DECsystem-10/DECSYSTEM-20 Processor Reference Manual*, DEC, 1982

[3] S. Rixner, et al., "Register organization for media processing, " in *Proc. HPCA*-6, pp.375-386, 2000

[4] G. T. Bryd and M. A. Holliday, "Multithreaded processor architectures," *IEEE SPECTRUM*, August 1995

[5] T Ungerer, B ROBIC, and J SILC, "A survey of processors with explicit multithreading," *ACM Computing Surveys*, Vol. 35, No. 1, pp. 29-63, March 2003

[6] J Glossner, "Sandblaster Low Power DSP," *IEEE Custom Integrated Circuits Conference*, 2004

[7] K. K. Parhi, *VLSI Digital Signal Processing Systems – Design and Implementation*, John Wiley & Sons, 1999

[8] A. V. Oppenheim, R.W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing*, 2nd edition, Prentice Hall,1999

[9] X. Y. Li, M. F. Stallmann, and F Brglez, "Effective bounding techniques for solving unate and binate covering problem," *ACM IEEE Design Automation Conference*, 2005

[10] R. Cordone, F. Ferrandi, D. Sciuto, and R. W. Calvo, "An efficient heuristic approach to solve the unate covering problem," *Proc. Design Automation and Test in Europe*, pp. 364-371, 2000

[11] O. Coudert, "On solving binate covering problems," In *The Proceedings of the Design Automation Conference*, pages 197-- 202, June 1996

[12] R. C. Larson and A. R. Odoni, *Orban Operation Resarch*, 2nd edition, Dynamic Ideas, 2007

[13] S. Liao, S. Devadas, K. Keutzer, and S. Tjiang, "Instruction selection using binate covering for code size optimization," In *Proceedings of International Conference on ComputerAided Design*, 1995

[14] Gero Dittmann, "Organizing libraries of DFG patterns," Proceeding of the DATE, 2004

[15] Gero Dittmann. "Organizing pattern libraries for ASIP design, " IBM Research Report RZ3488, April 2003

[16] T. J. Lin, P. C. Hsiao, C. W. Liu, and C. W. Jen, "Area-efficient register organization for fully-synthesizable VLIW DSP cores," *International Journal of Electrical Engineering*, May 2006 (EI)

[17] P. Chretienne, E.G. Coffman, Jr., J.K. Lenstra, and Z.Liu, *Scheduling Theory and its Application*, Wiley, June 1995.

[18] P. G. Paulin and J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC's," *IEEE TRANSACTIONS ON CAD*, Vol 8, No. 6, June 1999.

[19] Y. N. Chang, C. Y. Wang, and K. K. Parhi, "Loop-list scheduling for heterogeneous functional units," In *6th Great. Lakes Symposium on VLSI*, pages 2–7, March 1996

[20] S. Govindarajan and R. Vemuri, "Cone-based clustering heuristic for list-scheduling algorithms," In *Proc. of European Design & Test Conference (ED&TC)*, pages 456–462, March 1997

[21] D. D. Gajski, N. D. Dutt, A. C. Wu, and S. Y. Lin, *High-Level Synthesis – Introduction to Chip and System Design*, Kluwer Academic Publishers, 1992

[22] C. Liem, T. May, and P. Paulin, "Instruction-set matching and selection for DSP and ASIP code generation," *IEEE European Design and Test Conference, EDAC* ,1994

[23] J. Shu, T. C. Wilson, and D. K. Banerji, "Instruction-set matching and GA-based selection for embedded-processor code generation," *9th International Conference on VLSI Design*, January 1996

[24] J. V. Praet, G. Goossens, D. Lanneer, and H. D. Man, "Instruction set definition and instruction selection for ASIPs," *Proc. 7th IEEE/ACM Int. Symp. On High-Level Synthesis*, Niagara-on-the-Lake, May 1994

[25] G. Dittmann and A. Herkersdorf, "Multilayer intermediate representation for ASIP design and critical-path optimization," Technical Report RZ 3484, IBM Research, February 2003

[26] *DSPStone - A DSP oriented Benchmarking Methodology*, In ICSPAT, Aachen University of Technology, 1994

[27] BDTI, http://www.bdti.com

[28] S. Gordon, "Simplified use of 8x8 transform," Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, doc. JVT-I022, San Diego, USA, September 2003.

[29] Independent JPEG Group, http://www.ijg.org

[30] MAD: MPEG Audio Decoder, http://www.underbit.com/products/mad/

[31] M. Wroblewski, M. Mueller, A. Wortmann, S. Simon, W. Pieper, and J. A. Nossek, “A power efficient register file architecture using master latch sharing,” in Proc. ISCAS, May 2003

作者簡歷

卓毅，1983 年 4 月 21 日出生於台北市。2005 年取得國立交通大學電子工程學系學士學位，並繼續在國立交通大學電子工程研究所攻讀碩士。2007 年在劉志尉教授指導下，取得碩士學位。本篇論文「具複雜運算單元之低功率多執行緒資料路徑的研究與設計」為其碩士論文。