CHAPTER 5 A POWER-DRIVEN MULTIPLICATION INSTRUCTION-SET DESIGN METHOD FOR ASIPS

Power consumption has become one of the most important design issues for DSP designs targeted to multimedia and handheld applications. An important trend for low power DSP designs is to customize the instruction set for accommodating program characteristics with ASIPs (Application Specific Instruction-set Processors) [5].

Due to the large area, critical timing and high power dissipation, multipliers are the most critical components in an application-specific design. In [8], a configurable structure was proposed to reduce the power consumption of multiplier design. The key idea of a configurable multiplier-structure [8] to save power consumption is that the multiplier has two configurations. When the smaller multiplication is performed, unused parts of the multiplier are turned off, where "turn off" means gating input signals. This technique has been proven to be very effective in power reduction. Nevertheless, these techniques focus on ASIC (Application Specific Integrate Circuit) designs rather than processor designs. Moreover, the bit-width of the smaller configuration is simply chosen as the half bit-width of the larger multiplier.

However, we have observed that a smaller configuration with half of the maximum bit-width is not necessarily a good choice for the bit-width distribution of multiplication instructions for some specific applications. For example, in *IDCT* of *MPEG2* (video decoder) [52], one of the operands in mostly frequent executed multiplication is constant, whose maximum bit width is not whole or half of the bit width. Figure 5.1 illustrates the bit-width distribution of multiplication instructions by analyzing the *C* source program of an *MPEG2* video decoder. The data is collected by assuming that two operands of a multiplication instruction are the same bit-width.

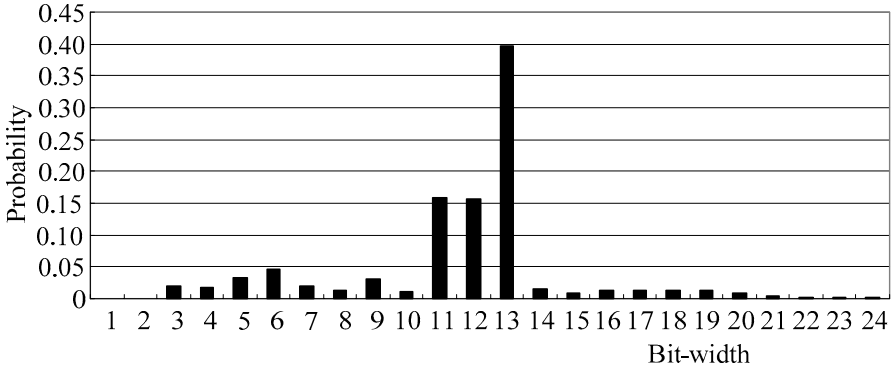

Figure 5.1: The bit-width distribution of multiplication-instructions of an *MPEG2* decoder design.

To achieve low power, one efficient multiplier design is using the configurable multiplier structure described in [8] to realize all multiplication instructions. A straightforward bit-width selection as described in [8] is to have the smaller configuration with the half bit-width of the large one. That is, a 24-by-24 bits multipliers with two configurations, 24-by-24 and 12-by-12. Then, using this straightforward configurable multiplier, the power consumption is $24 \times 24 \times 0.49C_{total}$ (instructions with operands' bit-widths greater than *12*) + $12 \times 12 \times 0.51C_{total}$ (instructions with operands' bit-widths equal to and smaller than *12*) = $355.68 \times C_{total}$ (area × execution cycle count), where C_{total} is the total execution cycle count.

Instead, we can also execute the multiplication instructions with two configurations, 24-by-24 and 13-by-13. By using this selection, the power consumption is $24 \times 24 \times 0.10C_{total}$ (instructions with operands' bit-widths greater than *13*) + $13 \times 13 \times 0.90C_{total}$ (the instructions with operands' bit-widths equal to and smaller than *13*) = $209.70 \times C_{total}$.

Obviously, the latter configurable multiplier (24-by-24 and 13-by-13) is 41.04% better than the straightforward one (24-by-24 and 12-by-12) in terms of power consumption. This motivates us to investigate how to determine the bit-width of a multiplication instruction-set using a dual-&-configurable multiplier (will be depicted

in the following section) for power reduction.

The rest of the section is organized as follows. Section 5.1 presents the multiplier structure. Section 5.2 presents the multiplication-instruction formation method.

5.1 A dual-&-configurable-multiplier structure

Before introducing the dual-&-configurable-multiplier structure, we define the notation $\{M, N\}$ as a multiplication instruction with two operands where the bit-width of the first operand is M and the second one is N.

To speed up performance, most state of art DSP processors provide two multipliers. Table 5.1 lists the summarized results of several processors and their multiplication instructions. The table shows that most DSP processors provide dual multipliers and two multiplication modes. A dual-multiplier can implement two multiplication modes: dual-multiplication and single-multiplication. In the dual-multiplication mode, two multipliers can execute two single multiplication instructions in parallel. In the single-multiplication mode, two multipliers are configured to operate on one single long bit-width multiplication. From the table, we observe that all of multipliers (32 bits) are composed of two smaller multipliers (16 bits, half of maximum bit-width) of equal size. It seems that designers suggest that two equal bit-widths to form a square multiplier is an efficient bit-width in terms of area and parallelism. We would like also to exploit this problem in this thesis.

Table 5.1: The survey of several DSP processors and their multiplication instructions.

Names	Type and number of multiplier	Multiplication instructions
Analog Device TigerSharc	8 16-by-16 multipliers	{16,16} and {32,32}
DSP Group OAK, Teak, Palm	2 16-by-16 multipliers	{16,16} and {32,32}
Lucent Technology DSP 16000	2 16-by-16 multipliers	{16,16}, {16,32}, and {32,32}
PHILIPS REAL DSP	2 16-by-16 multipliers	{16,16}
TI TMS320C6000	2 16-by-16 multipliers	{16,16}
ZSP 400	2 16-by-16 multipliers	{16,16} and {32,32}

In this thesis, we adopt a dual-&-configurable multiplier as our target multiplier structure. Figures 5.2 and 5.3 show an example of the dual-&-configurable structure. Figure 5.2 shows our dual structure that includes two multipliers, *Multiplier1* and *Multiplier2*, to execute {*m, n1*} and {*m, n2*}, respectively, and the control multiplexers. When *control* is set to zero, the multiplexers forward all zero values to carry (borrow) inputs of the adders and subtraters of *Multiplier1*. Hence, the two multipliers can execute two multiplications, {*m*, *n1*} and {*m*, *n2*}, in parallel. When *control* is set to one, the multiplexers select the carry and borrow values of *Multiplier2* to *Multiplier1* and produce one product of a multiplication {*m*, *n1+n2*}.

Figure 5.3 shows the configurable-multiplier structure used in each multiplier which is based on the configurable-multiplier structure described in [8]. A configurable multiplier can execute two or more multiplication instructions with different bit-widths. Figure 5.3 shows a configurable multiplier with two configurations that can perform two multiplication instructions: {*m*, *n1*} is executed in *Multiplier1* and {$m1_{lp}$, $n1_{lp}$} in $Multiplier1_{lp}$. When the *control* is set to one, the entire multiplier (i.e., *Multiplier1*) is activated. When the *control* is set to zero, $Multiplier1_{lp}$ is activated alone. In the following section, this dual-&-configurable multiplier structure is used as our target structure.

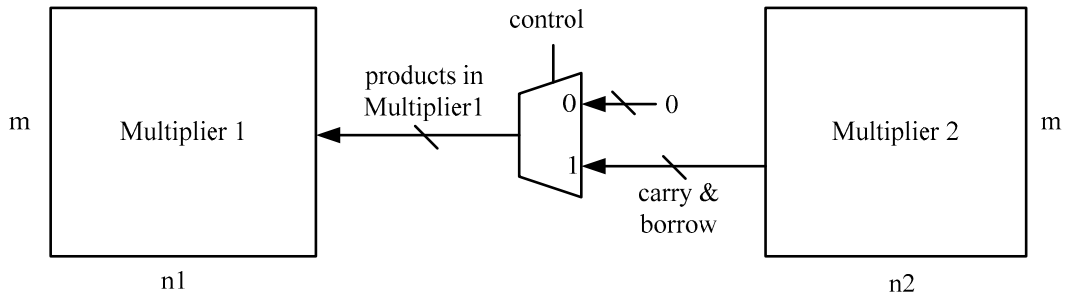

Figure 5.2: The block diagram of a dual-multiplier.

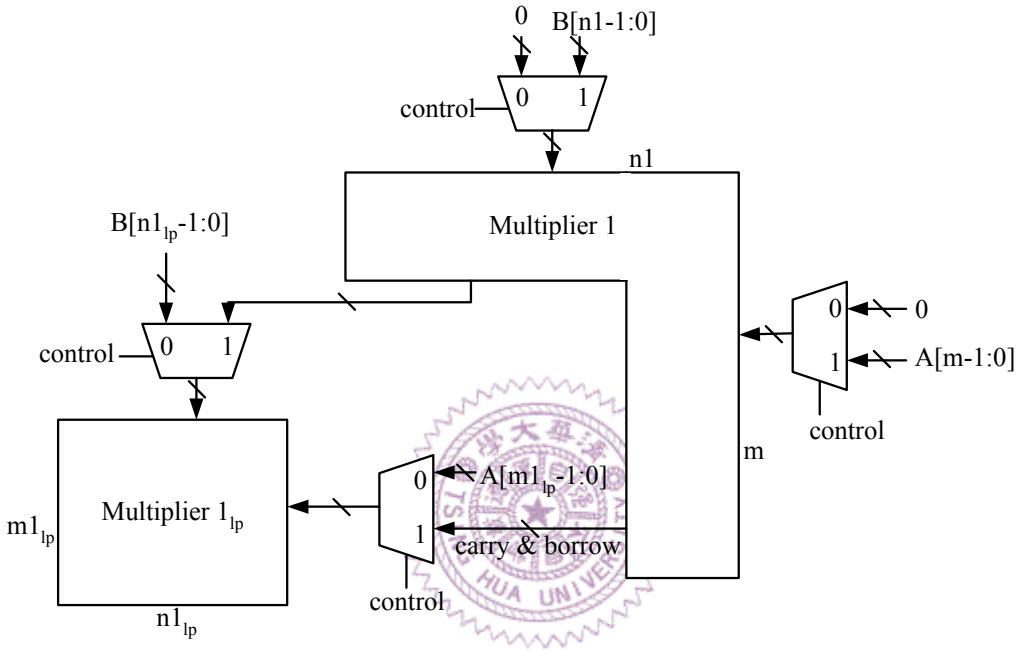

Figure 5.3: The block diagram of a configurable multiplier with two configurations.

5.2. The multiplication instruction-set formation algorithm

This section presents the detailed multiplication instruction-set formation algorithm. Section 5.2.1 gives the problem definition and overview of the multiplication instruction-set formation algorithm. Section 5.2.2 describes the algorithm to determine the bit-widths of a dual-multiplier. Finally, Section 5.2.3 presents the configurable-multiplier formation algorithm.

5.2.1 Problem definition and overview of the algorithm

In an application-specific system design, the bit-widths of multiplication operands usually have a wide variation range from different applications. Often, the behavior of the application-specific system is statically known. Hence, it is feasible to analyze the application programs and utilizing their characteristics to further optimize the application-specific systems. Based on this observation, our problem is defined as follows: Given an application specific program (or a set of programs) and input data set, generate a set of multiplication instructions such that both the power consumption and the execution time of multiplication instructions are minimized.

Figure 5.4(a) presents the design flow of the proposed algorithm that consists of three steps:

Step 1: For a given application program and input data, we analyze the variables of each multiplication statement at instruction level and report the *effective bit-width* [53] of variables. In this thesis, we use the *effective bit-width* proposed in [53], where the *effective bit-width* is calculated as the smallest size that can hold both maximum and minimum values of a variable. We use two methods [54][55] to analyze the *effective bit-width* of variables. One is the static analysis and the other is the simulation-based dynamic analysis. After analyzing multiplication instructions in the program, a set of initial bit-widths for multiplication instructions is determined.

Step 2: Based on the analyzed results including *effective bit-width* and profiling execution sequence generated in Step 1, an instruction transition graph is thus constructed. We, from the instruction transition graph, then determine the bit-widths of two multiplication instructions that can be executed by each multiplier, one multiplication instruction that can be executed by both multipliers in parallel, and one multiplication instruction that can be executed by the concatenated multipliers. The

details will be discussed in Section 5.2.2.

Step 3: After determining the bit-widths of the dual-multiplication instructions, we apply the VLIW compiler techniques [56] to parallelize multiplication instructions into dual multipliers. After that, we build an instruction transition graph for each individual multiplier. For each graph, we apply a node-merging procedure as a pre-processing step and then a graph partitioning algorithm [57] to determine the multiplication instruction-set for the configurable-multiplier. The details will be presented in Section 5.2.3. After determining the multiplication instruction set, we generate the dual-&-configurable-multiplier.

Figure 5.4(b) shows a simple example of the design flow. In Step 1, the target program is analyzed. In Step 2, a transition graph is built according to the analyzed result. The four multiplication instructions, *M1*, *M2*, *M3*, and *M4*, and a dual-multiplier are generated after the bit-width determination for the dual multiplier. *M1* performs {*m*, *n1*} in *Multiplier1*, *M2* {*m*, *n2*} in *Multiplier2*, *M3* {*m*, *n1*} and {*m*, *n2*} in the dual multiplication mode, and *M4* {*m*, *n1*+*n2*} in *Multiplier1* + *Multiplier2* by concatenating *Multiplier1* and *Multiplier2*. In Step 3, a transition graph for each multiplier is built according to the parallelized multiplication instruction code and the dual-multiplier. The multiplication instruction-set for each multiplier is generated according to the transition graph. In Figure 5.4(b), configuration number is set to two. In *Multiplier1*, $M1_{lp}$ is generated for low power configuration, while in *Multiplier2*, $M2_{lp}$ is generated.

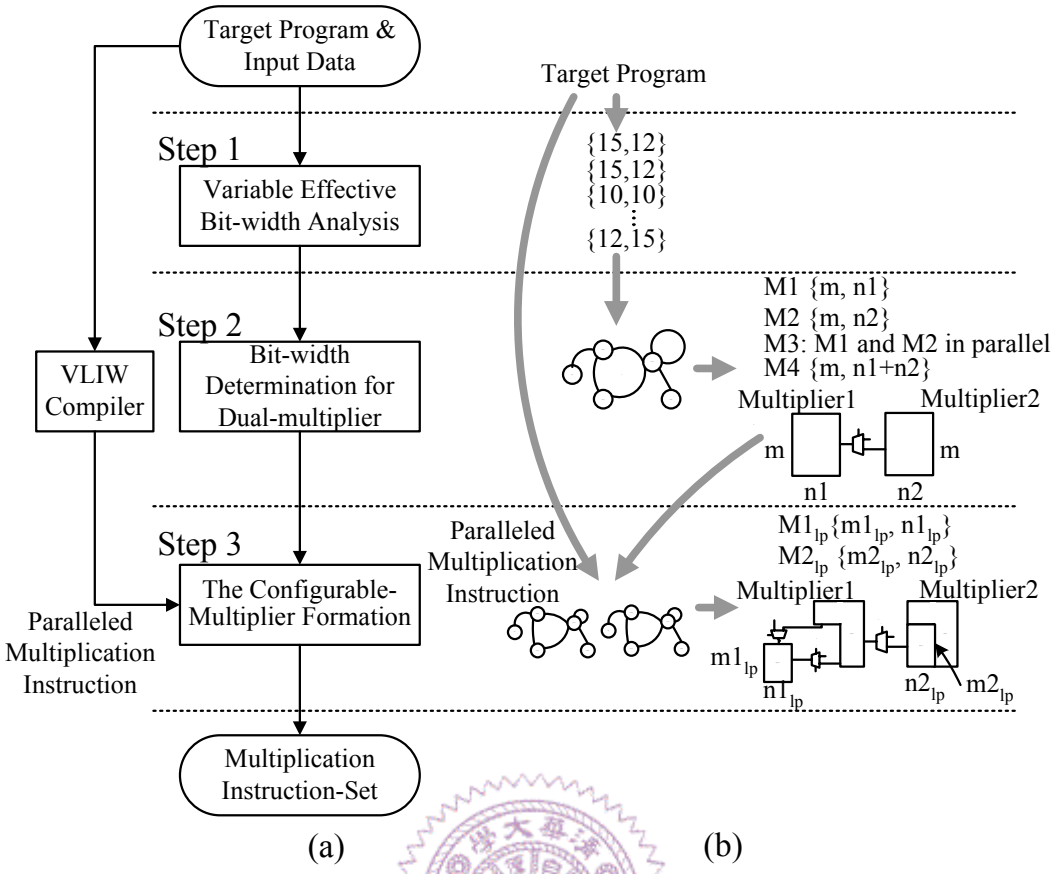

Figure 5.4: The design flow of the multiplication instruction formation algorithm.

5.2.2 Bit-width determination for the dual-multiplier

In Step 2, bit-width for dual multiplier will be determined. First, we construct a weighted graph to model the execution frequency and transitions among multiplication instructions with different bit-widths. The weighted graph is constructed from the profiling of the multiplication instruction execution sequences generated from the target application programs. Let $G=(V, WV, E, WE)$ be a weighted graph, where V is a node set, WV weights on the nodes, E an edge set, and WE weights on the edges. A node v in V represents one multiplication instruction in the program, the weight on the node, v, represents the execution frequency of the instruction v, an edge between node v_1 and node v_2 represents that instruction v_1 is

executed before instruction v_2 or v_2 is executed before v_1, and the weight on the edge represents the transition frequency between v_1 and v_2.

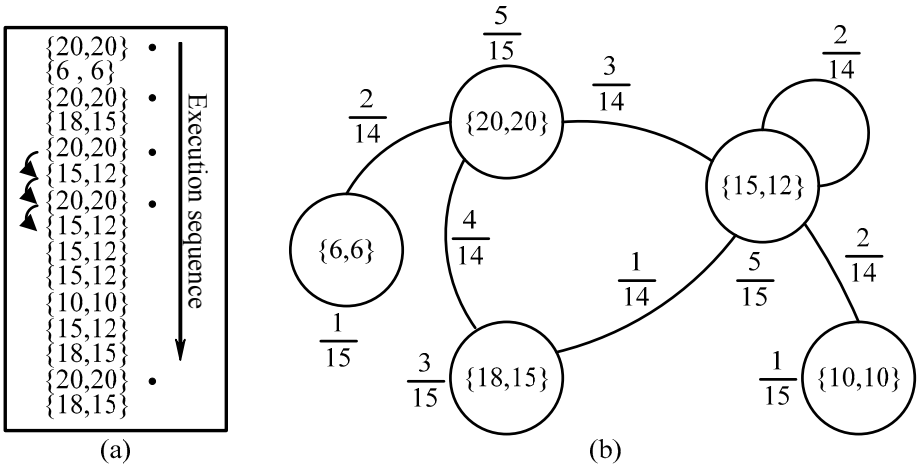

Figure 5.5: A transition graph example.

Figure 5.5 shows a transition graph example. In Figure 5.5(a), there are 5 types of multiplication instructions in the execution sequence. Hence, there are 5 nodes in the graph as shown in Figure 5.5(b). The numbers next to the nodes are the execution frequencies of the instructions. For example, {20, 20} is executed 5 times (marked by dots in Figure 5.5(a)) and there are 15 multiplication instructions in total. As a result, the execution frequency of node {20, 20} is 5/15. The edges between nodes and the weights on the edges are constructed also by tracing the multiplication instruction execution sequence. For instance, in Figure 5.5(a) the three arrows show that the {20, 20} is executed before {15, 12} twice and {15, 12} before {20, 20} once and the total number of transition is 14. Hence, there is an edge between nodes {20, 20} and {15, 12} and the weight of the edge is 3/14.

After generating a transition graph, we determine the bit-widths of two individual multiplication instructions for the dual-multiplier. The goal of selecting the bit-width is to minimize the total energy consumption while taking execution time into consideration. To that end, the first objective is to minimize the total area and hence power consumption. The second objective is to allow as many multiplication

instructions with small bit-widths executed in parallel as possible. In this case, the total execution time is minimized.

To determine the bit-width of the dual-multiplication instructions, a straightforward algorithm is to enumerate all possible configurations. This enumeration takes exponential time in terms of the number of possible bit-widths, i.e. the number of nodes in the transition graph. Instead, we consider only the configurations presented in the nodes and edges of the instruction transition graph. For a node *v* representing a multiplication instruction of {*m1, n1*} in *G*, we consider a dual-multiplication instruction that allows two *m1*-by-*n1* multiplication instructions to be executed in parallel. For an edge *e* representing transition between multiplication instruction of {*m1, n1*} and multiplication instruction of {*m2, n2*} in *G*, we consider a dual-multiplication instruction that allows multiplication instructions of *m1*-by-*n1* and of *m2*-by-*n2* multiplications to be executed in parallel. Moreover, when a dual-multiplier is in single-multiplication mode, the bit-width must be large enough to hold the largest operands of the multiplication instruction.

Therefore, the configuration of the dual-multiplier is determined as follows. Let *max_m* and *max_n* be the maximum bit-width of the first operand and the second operand of all multiplication instructions. For the configuration corresponding to a node *v* representing multiplication instruction of {*m1, n1*}, we have a dual-multiplication instruction with two individual multipliers of *max_m*-by-*n1* and *max_m*-by-(*maximum* of *max_n*-*n1* and *n1*). This dual-multiplier, in the single-multiplication mode, can support one multiplication instruction of {*max_m, max_n*} and in the dual-multiplication mode, can support two multiplication instructions of {*max_m, n1*} and {*max_m, n1*}. Similarly, for the configuration corresponding to an edge *e* representing multiplication instruction of {*m1, n1*} and multiplication instruction of {*m2, n2*}, we have a dual-multiplication instruction with

two individual multipliers of *max_m*-by-*n2* and *max_m*-by-(*maximum of max_n-n2* and *n1*). This dual-multiplier can support {*max_m, max_n*} multiplication instruction in the single-multiplication mode and support two multiplication instructions of {*max_m, n1*} and {*max_m, n2*} in the dual-multiplication mode. Figure 5.6 shows a configuration for an edge connecting nodes of {*m1, n1*} and {*m2, n2*}.

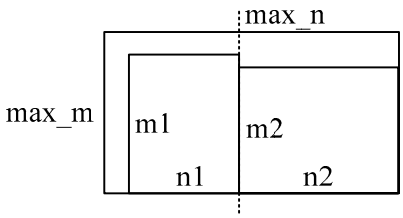

Figure 5.6: A configuration-determination example.

For each configuration *c*, we compute the cost function, *Energy*(*G*,*c*). When *c* corresponds to a node, *v*, we have

$$Energy(G,c) = \left(\sum_{v} wv_v \times parallel_v \right) \times timing \times area_{\max},$$

where wv_v represents execution frequency of instruction *v*, $parallel_v$ is 1 if the bit-width of multiplication instruction at node *v* is smaller than or equal to one single multiplier, $parallel_v$ is 2 otherwise, *timing* is the critical timing of the generated multiplier and $area_{max}$ is the area of this dual-multiplier. Similarly, when *c* corresponds to an edge, *e*, we have

$$Energy(G,c) = \left(\sum_{e} we_e \times parallel_e \right) \times timing \times area_{\max},$$

where we_e represents transition frequency on the edge *e*. If two nodes of an edge can be executed in parallel, $parallel_e$ is 1. Otherwise, $parallel_e$ is 2.

The reason behind this cost function is to minimize the total energy consumption. The first two terms represent the execution time required to complete the job with achievable maximum parallelism and the third term is the power consumption of each clock cycle estimated by the total area. The total number of configurations considered is (|*e*| + |*v*|). The configuration that results in the minimum cost is selected.

5.2.3 The configurable-multiplier formation

After determining the bit-width of dual-multiplier, we need to determine the low power configuration of each multiplier in the dual-multiplier. To decide the configuration of each multiplier, a possible execution sequence on each multiplier needs to be known. Since a dual multiplier allows instructions to run in parallel, the source code is recompiled by the VLIW tool, *Trimaran* [56], to parallelize the executions of multiplication instructions.

Now, for each multiplier, we have an instruction execution sequence. The next step is to build an instruction transition graph described in Section 5.2.2 for each individual multiplier according to the generated instruction execution sequence. The next step is applying the multilevel partitioning algorithm [57] to the transition graph. To determine the low power configuration, total energy cost instead of cut size is used as the cost function of the partitioning algorithm. The cost is defined as follows. Suppose the reduced graph G is partitioned into $\pi = (G_1, G_2, \ldots, G_n)$, where n is the number of partitioning. Let GSF_i denote the frequency that transitions will bring an instruction of G_i to G_i itself and GCF_{ij} the transition frequency occurs between instructions in G_i and instructions in G_j (where $i \neq j$). Hence, GSF_i is calculated as the sum of transition frequencies (weight) of edges that connect two instructions in G_i, and GCF_{ij} the summation of transition frequencies of edges that cross two sub-graphs, G_i and G_j. The energy cost is defined as

$$Energy\_Cost = \sum GSF_i \times Garea_i + \sum_{i,j} GCF_{ij} \times (Garea_{i|j} + overhead),$$

where $Garea_i$ represents the area of the maximum bit-width instruction that can hold all instructions in the group i, $Garea_{i|j}$ the area of the configuration multiplier for group i and j and *overhead* the selection multiplexer overhead. The partitioned result has two properties. One is that group with smaller area will have a higher self

transition. The other is that the cross transition between groups is minimized. Clearly, multiplication instructions with high self-transition frequencies, low cross-transition frequencies, and low energy are desirable.

Based on the results produced by partitioning algorithm, low power multiplication instructions and low power configurations are generated for each multiplier. For each partitioned group, a low power configuration and its corresponding low power multiplication instruction is generated. The bit-width of the configuration is set to be the maximum bit-width of the multiplication in that group.

5.3 Experimental results

We have conducted two sets of experiments by using the *MediaBench* [52] as the target application. Our experimental platform used the *Trimaran* infrastructure [56].

In the first experiment, we compared the average power consumption and area overhead with/without applying the configurable-multiplier formation algorithm in Section 5.2.3. In the second experiment, we compared the average power consumption, execution time, and area of equal-size partitioning square-multipliers design method and our proposed bit-width determination algorithm described in Section 5.2.

We used three representative benchmarks from the *MediaBench* suite: one for video decompression (*MPEG2*), one for audio codec (*G721*), and one for image compression (*EPIC*). We obtained the input data of each program from [65][66][67]. Based on the profiling result on these programs, a 24-by-24 bit was selected as the maximum bit-width.

After determining the bit-width of multipliers, we generated the *Verilog* design description. The final circuit was generated by the *Synopsys Design Compiler* with the *TSMC 0.25um* cell library. Throughout the entire experiments, we used *Synopsys*

PrimePower to calculate average power consumption. Both dynamic and leakage power are considered. The area and timing data was reported by *Design Compiler* with the *report_area* and *report_timing* options.

5.3.1 The configurable-multiplier formation algorithm

In this experiment, we evaluated our method for the configurable multiplier with the configurable-multiplier formation algorithm proposed in Section 5.2.3.

Figure 5.7 shows the power consumption and area comparisons with different configuration numbers and timing constraints on the three benchmarks. Since the results of the three benchmarks show the same trend, we only used *MPEG2* (Figure 5.7(a)) as a demonstrated example. The timing constraints ranging from 32.22ns to 40.27ns (Figure 5.7(a)) were obtained by first synthesizing the 24-by-24 multiplier with the fastest timing option, and then based on the fastest timing, gradually relaxing the timing constraints from 105% to 125%. For each multiplier, we performed experiments on four different numbers of configurations (from one to four configurations).

The results show that the multipliers with two configurations achieved the highest power reduction. For example, at the timing constraint of 32.22ns, the two-configurations achieved 17.92% power reduction with only 7.46% area overhead. Since more configurable control circuits are needed when increasing the number of configurations, the result also shows that the total area is proportional to the number of configurations.

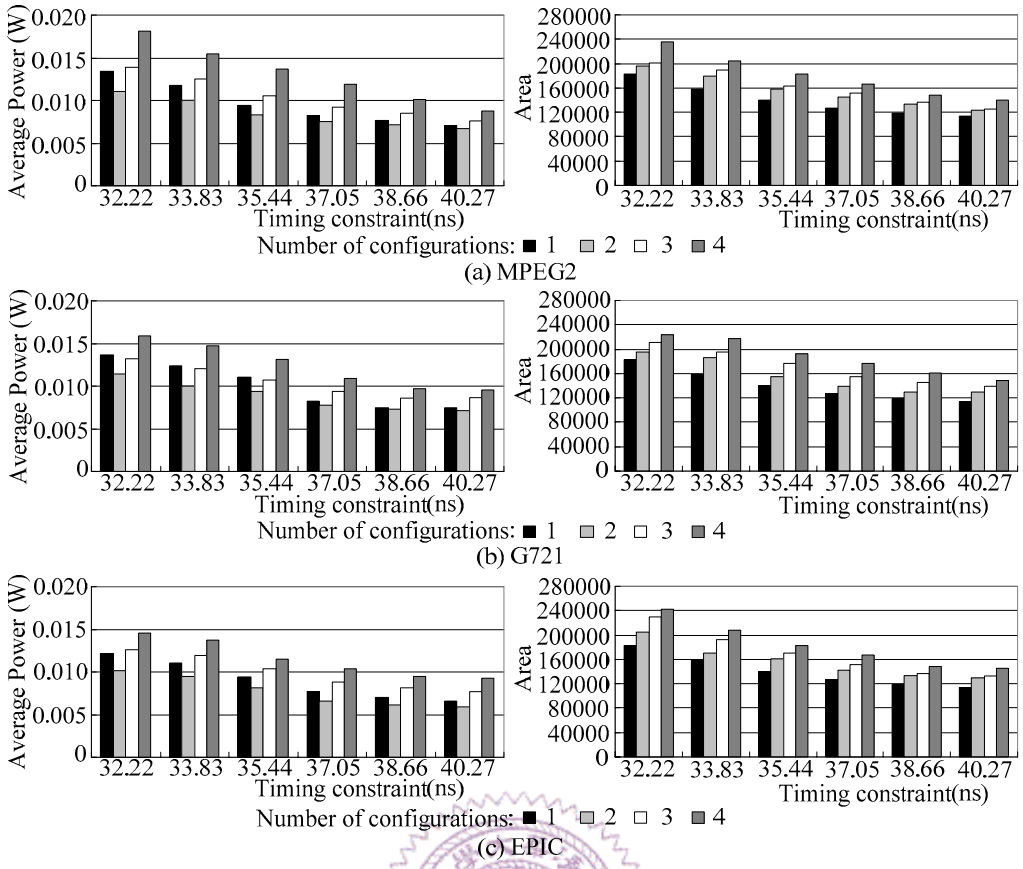

Figure 5.7: Power consumption and area of a configurable multiplier with different configurations.

Table 5.2: The multiplication instruction-set generated by the configurable-multiplier formation algorithm.

Bench	Multiplication Instruction-Set
MPEG2	{24, 24} and {13, 13}
G721	{24, 24} and {16, 16}
EPIC	{24, 24} and {14, 14}

Table 5.3: Power comparisons of the low power configuration with the bit-width being the half of the larger multiplier and that instruction generated by our algorithm.

Bench	Half of Large Multiplier(W)	Our Algorithm(W)	Power Improvement
MPEG2	1.22E-2	1.11E-2	9.02%
G721	1.27E-2	1.14E-2	10.23%
EPIC	1.19E-2	1.07E-2	10.08%

Table 5.2 shows the generated configurations for the three benchmarks when the configuration number is set to two. For these three benchmarks, the best bit-width of the smaller configuration is not half of the bit-width of the large multiplier. Take *MPEG2* as an example, we can find that the bit width of a smaller configuration is 13-by-13 rather than 12-by-12. Table 5.3 shows the comparison results of the low power configuration with the bit-width being the half of the larger multiplier (i.e., a multiplier has two configurations, 24-by-24 and 12-by-12) and that instruction generated by our algorithm (shown in Table 5.2). The results also show that our method achieves 10% improvement.

5.3.2 The multiplication instruction-set formation algorithm

In this experiment, we applied the multiplication instruction-set formation algorithm presented in Section 5.2. The number of configurations of the multiplier is set to two. Table 5.4 shows the multiplication instruction-sets generated by our algorithm. For these three benchmarks, we found that the best bit-width of the smaller multiplier in each configurable multiplier selected by our algorithm would not be half of maximum bit-width. Moreover, two combined dual multiplier do not form a square multiplier. Take *MPEG2* as an example, two low power configurations of two multipliers are 20-by-13(out of 24-by-13) and 13-by-13(out of 24-by-13). Two dual multipliers are 24-by-13 and the combined larger multiplier is 24-by-26 bits. Moreover, for the example of *EPIC*, the bit-widths of the two multipliers are not the same (24-by-13 and 24-by-16).

Table 5.4: The multiplication instruction-set generated by the multiplication instruction-set formation algorithm.

Bench	Multiplication Instruction-Set
MPEG2	*M1* performs {24, 13} and {20, 13}, *M2* performs {24, 13} and {13, 13}, *M3* performs dual-multiplication, and *M4* performs {24, 26}.
G721	*M1* performs {24, 16} and {14, 16}, *M2* performs {24, 16} and {16, 16}, *M3* performs dual-multiplication, and *M4* performs {24, 32}.
EPIC	*M1* performs {24, 13} and {13, 13}, *M2* performs {24, 16} and {14, 14}, *M3* performs dual-multiplication, and *M4* performs {24, 29}.

Tables 5.5, 5.6, and 5.7 show the comparison results of equal bit-width partitioning of square multiplier; that is, two 24-by-12 bits multipliers (each multiplier has two configurations, 24-by-12 and 12-by-12) and multiplication instructions generated by our algorithm (shown in Table 5.4). From Table 5.5 and 5.6, it can be seen that power and execution time are improved by using the instructions produced by our algorithm. Take *MPEG2* as an example. The power reduction achieves 17.91% improvement by replacing the equal-size partitioning multiplication instructions with the instructions shown in Table 5.4. Although the critical timing of the multipliers generated by our method is longer than that of traditional one (32.22ns), the number of execution cycles is reduced. As a result, the total execution time using our instruction set is shorter than that using the traditional multiplication instruction set. For the other two benchmarks, similar results were observed. Tables 5.6 and 5.7 show the improvement of execution time and the area overhead. For *MPEG2*, the execution time is 3.51% faster with only 8.02% area overhead. For G721, at the area overhead of 28.27%, the execution time is improved to 10.43%.

The results demonstrate that our method can improve the power consumption and execution time of multiplication instructions with some increase in area overhead. The results also show that the bit-width of the smaller multiplier in a configurable-multiplier is not necessary to be half of maximum bit-width. Moreover, the two multipliers in a dual-multiplier do not necessarily form a square multiplier.

Table 5.5: Power comparisons on an equal bit-width partitioning of square multiplier and our algorithm.

Bench	Square Multiplier(W)	Our Algorithm(W)	Power Improvement
MPEG2	2.01E-2	1.65E-2	17.91%
G721	2.32E-2	1.89E-2	18.53%
EPIC	1.84E-2	1.56E-2	15.22%

Table 5.6: Timing and execution time comparisons on an equal bit-width partitioning of square multiplier and our algorithm.

Bench	Critical Timing(ns)		Cycle Count (Average)		Execution Time(s)		
	Square Mul.	Our Method	Square Mul.	Our Method	Square Mul.	Our Method	Imp. (%)
MPEG2	32.22	33.51	8.62E6	7.99E6	277.74	267.99	3.51
G721	32.22	34.14	4.85E6	4.10E6	156.27	139.97	10.43
EPIC	32.22	33.72	2.38E6	2.11E6	76.68	71.15	7.21

Table 5.7: Area comparisons on an equal bit-width partitioning of square multiplier and our algorithm.

Bench	Square Multiplier	Our Algorithm	Area Overhead
MPEG2	188527	203646	8.02%
G721	188527	241829	28.27%
EPIC	188527	222800	18.18%