

Clock-Frequency Assignment for Multiple Clock Domain Systems-on-a-Chip

Scott Sirowy, Yonghui Wu, Stefano Lonardi, Frank Vahid*

Department of Computer Science and Engineering – University of California, Riverside
{ssirowy, yonghui, stelo, vahid}@cs.ucr.edu

*Also with the Center for Embedded Computer Systems, University of California, Irvine

Abstract

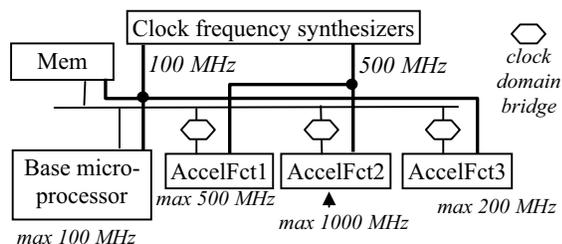
Modern systems-on-a-chip platforms support multiple clock domains, in which different sub-circuits are driven by different clock signals. Although the frequency of each domain can be customized, the number of unique clock frequencies on a platform is typically limited. We define the clock-frequency assignment problem to be the assignment of frequencies to processing modules, each with an ideal maximum frequency, such that the sum of module processing times is minimized, subject to a limit on the number of unique frequencies. We develop a novel polynomial-time optimal algorithm to solve the problem, based on dynamic programming. We apply the algorithm to the particular context of post-improvement of accelerator-based hardware/software partitioning, and demonstrate 1.5x-4x additional speedups using just three clock domains.

1. Introduction

Modern system-on-a-chip platforms support multiple clock domains on a single chip. A clock domain is a block of circuitry that operates at a single clock frequency that may differ from the frequency of other blocks on the same chip. In addition to reducing clock skew related problems, a second advantage of multiple clock domains is that a module having a shorter critical path than other modules can be clocked at its maximum frequency, rather than all modules being clocked at the maximum frequency of the slowest module. Figure 1 shows a system with four modules having maximum frequencies of 100, 500, 1000, and 200 MHz. Communication across clock domains is a challenge, but has been aggressively researched recently, with established solutions (e.g., [4][15]) and with pre-designed bridge blocks present in many system libraries.

Because circuitry to generate a unique clock frequency is not free, platforms impose a limit on the number of unique clock domains. For example, the Xilinx Virtex-II Pro FPGA (field-programmable gate array) has eight clock frequency synthesizers, able to generate frequencies between 24 MHz and 420 MHz via different clock multiplication and division factors [22]. If the number of modules having distinct maximum frequencies exceeds the limit on unique clock frequencies, then we define the *clock-frequency assignment problem* as assigning a frequency to each module such that a performance metric is optimized, subject to a limit on the number of unique frequencies. The example in Figure 1 shows a frequency assignment for four modules driven by only two available frequencies, chosen to be 100 and 500 MHz.

Figure 1: Four modules driven by two clock frequencies.



978-3-9810801-2-4/DATE07 © 2007 EDAA

The clock-frequency assignment problem has not been addressed in the design automation literature, to the best of our knowledge. The contributions of this paper are the identification and definition of the clock-frequency assignment problem, and the development of a novel optimal yet efficient dynamic programming algorithm to solve the problem. Section 2 of the paper introduces and defines the problem. Section 3 describes our dynamic programming algorithm. Section 4 gives results on a commercial H.264 benchmark and on numerous synthetic benchmarks. Section 5 summarizes related work. Section 6 concludes.

2. Problem background and definition

2.1 Background

A system-on-a-chip may consist of tens or hundreds of communicating processor-level modules. Some modules, such as those interfacing directly with external circuitry, may have hard real-time constraints that dictate a specific operating frequency. Other modules may have softer performance requirements, for which the clock-frequency assignment problem seeks to optimize a performance metric.

A common performance metric for multiple modules is the (possibly weighted) summation of the execution time for all the modules. For example, Kumar [14] sought to create a set of heterogeneous general-purpose processor core microarchitectures with the goal to minimize the sum of the execution times of a set of benchmarks on each core.

A summation metric could be used to minimize the critical path of a task-level dataflow graph [12]. Such tasks may be mapped to two or more processing modules. When those processors are not system-level pipelined (mapping such tasks to multiple processors without pipelining is often done for hardware modularity purposes), the goal of minimizing the task graph's critical path becomes the goal of minimizing the sum of the critical tasks' execution times.

Another use of a summation metric is in accelerator-based hardware/software partitioning of sequential programs, illustrated in Figure 2. When microprocessor execution reaches a critical function, control switches from the microprocessor to a hardware accelerator (such as a hardware floating-point unit or graphics accelerator); after the accelerator completes, control switches back to the microprocessor. The goal of minimizing program execution time in this case is the same as minimizing the sum of the microprocessor and accelerator execution times for the program.

Figure 2: Accelerator hardware/software partitioning model.

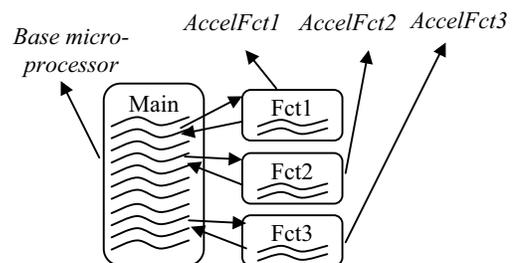


Figure 3: Excerpt of H.264 application’s critical functions’ information.

Function	sw time	# hw cycles	hw max clk
<MotionComp_00>:	0.040733	1	281
<InvTransform4x4>:	0.034787	8	194
<FindHorizontalBS>:	0.025026	1	140
<GetBits>:	0.024681	1	200
<FindVerticalBS>:	0.02366	1	140
<MotionCompChromaFullXFully>	0.023577	1	285
<FilterHorizontalLuma>:	0.023559	4	134
<FilterVerticalLuma>:	0.020008	4	138
<FilterHorizontalChroma>:	0.018803	4	134
<CombineCoefsZerosInvQuantSca	0.018438	1	120
<MotionCompensate>:	0.016822	10	40
..<FilterVerticalChroma>:	0.016035	4	138
..<MotionCompChromaFracXFrac	0.016023	32	78
..<ReadLeadingZerosAndOne>:	0.015665	1	106

This problem, using FPGAs to implement accelerators, served as the original motivation of our work. Several platforms today integrate FPGAs closely with a microprocessor [1][6][22] and commercial tools are beginning to appear to automatically partition a sequential application’s critical functions to accelerators [3][7][16].

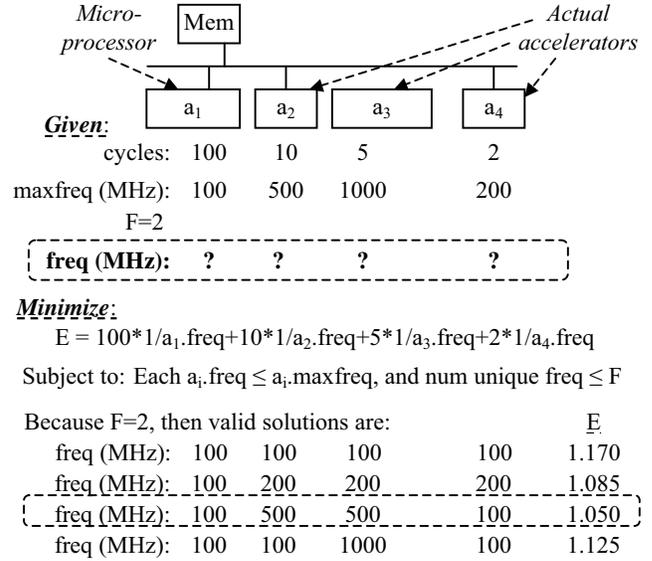
We note that the accelerator model of partitioning, which uses coprocessors as microprocessor surrogates rather than parallel processors, represents a different model than the widely researched model of partitioning a task graph among concurrently-executing programmable and coprocessors (e.g., [8][12]). Nevertheless, the accelerator model has received research attention [2][10][17][20] and is the model for several commercial partitioners [7][16]. The model frees the architecture from the complexity of synchronizing concurrent processors that share common data, and is justified due to well-known limits on available parallelism in sequential programs [23], as well as to the relatively short execution time of typical accelerators.

Figure 3 shows data for the twelve most critical functions of a commercial H.264 video decoder by Freescale, obtained from Stitt [21]. Unlike common benchmarks taken from publicly available reference implementations, that decoder’s code was highly optimized, and thus did not consist of just two or three critical functions, but rather of 42 critical functions that together accounted for about 90% of execution time. Stitt’s partitioning into accelerators was straightforward, involving implementing an accelerator for each critical function. Figure 3 lists, for each H.264 function, the function’s execution time in software on a given microprocessor, the number of cycles for the function’s corresponding accelerator (including communication cycles), and the maximum clock frequency (in MHz) at which that accelerator could execute as determined by Xilinx synthesis tools for Virtex FPGAs. Notice the variation in maximum frequencies, ranging from 40 MHz to 285 MHz.

2.2 Definition

We define the clock-frequency assignment problem in the context of accelerators, but the problem definition directly applies to any modules whose execution sum must be minimized. The problem definition, illustrated by example in Figure 4, begins with an application represented as a set of accelerators $A = \{a_1, a_2, \dots, a_M\}$, where M is an integer ≥ 1 . Each a_i is a circuit to accelerate one or more functions of the application, where a function may be a subroutine, loop, or even a large block of code. Note that a single accelerator a_i may accelerate multiple functions, much as a single

Figure 4: Clock-frequency assignment example.



floating-point unit accelerates floating-point addition, multiplication, and other functions. For formulation simplicity, we can treat the microprocessor itself as just another accelerator, which will implement all the remaining functions not implemented on an actual accelerator. We assume the accelerators were determined by an earlier hardware/software partitioning step. While the eventual clock frequency of each accelerator could influence partitioning choices, previous partitioning work has assumed a single frequency. Thus, using multiple clock domains can be viewed as a post-processing step to partitioning to improve performance further. Future work may include integrating clock-frequency assignment with hardware/software partitioning for even better results.

Each accelerator a_i has several weights. The weight $a_i \cdot \text{cycles}$ corresponds to the number of clock cycles that the accelerator contributes to the total clock cycles for the application, not including cycles required for accessing memory. The number may be obtained through profiling, code analysis, or user annotation, and may represent average or worst-case values, depending on designer goals – those issues are orthogonal to our approach.

The weight $a_i \cdot \text{maxfreq}$ represents the fastest clock frequency at which this accelerator may execute. That frequency would typically be determined by synthesizing the accelerator and then taking the inverse of the critical path.

The weight $a_i \cdot \text{freq}$ represents the frequency at which accelerator a_i is being clocked in an implementation. This number is not given, but rather must be determined. The determined number must be less than or equal to $a_i \cdot \text{maxfreq}$.

The application’s execution time E is the sum of the application’s computation time and communication time. The computation time equals the cycles multiplied by $1/\text{freq}$ values for every accelerator. The communication time equals the total number of memory accesses multiplied by the memory access time. We originally included communication time in our problem formulation, but found that component of time unnecessary to include during clock-frequency assignment. The reason is that communication time equals the number of memory accesses by each accelerator times the time associated with each access. The time associated with each access consisted of two parts, one part dependent on the accelerator’s frequency and hence foldable into the accelerator’s compute time, and the other part independent of the accelerator frequencies, instead dependent on the frequency of

the microprocessor and memory, which do not impact the *relative* total execution time of a given partitioning. Note that this non-overlapping computation/communication model of execution time, while different from the model uses in multi-processor based hardware/software partitioning, holds for accelerator-based hardware/software partitioning.

We are also given a maximum number of unique clock frequencies F available to the accelerators.

Thus, the *clock-frequency assignment problem* is to:

Find a positive integer value for every $a_i \cdot freq$, such that each $a_i \cdot freq$ is less than $a_i \cdot maxfreq$ for every i , the number of distinct $a_i \cdot freq$ values is less than or equal to F , and the execution time E is minimized.

Figure 4 provides an example showing a microprocessor (a special “accelerator”) and three actual accelerators. Each of these four accelerators is weighed with the number of cycles executed on the accelerator (100, 10, 5, and 2, respectively), and the maximum frequency for that accelerator (100, 500, 1000, and 200 MHz, respectively). The figure shows all possible clock-frequency assignments with the execution time resulting from each; the optimal solution, having the best execution time, is circled. Note that this example is trivially-small; actual applications may have many tens of accelerators, and platforms may have tens of available frequencies, with both numbers growing yearly.

A real clock frequency synthesizer may not generate all possible frequencies in a range. In that case, we replace each accelerator’s ideal maximum frequency by the highest synthesizable frequency less than or equal to that ideal. If an accelerator’s ideal frequency exceeds the range, we reduce the ideal frequency to be the highest synthesizable frequency.

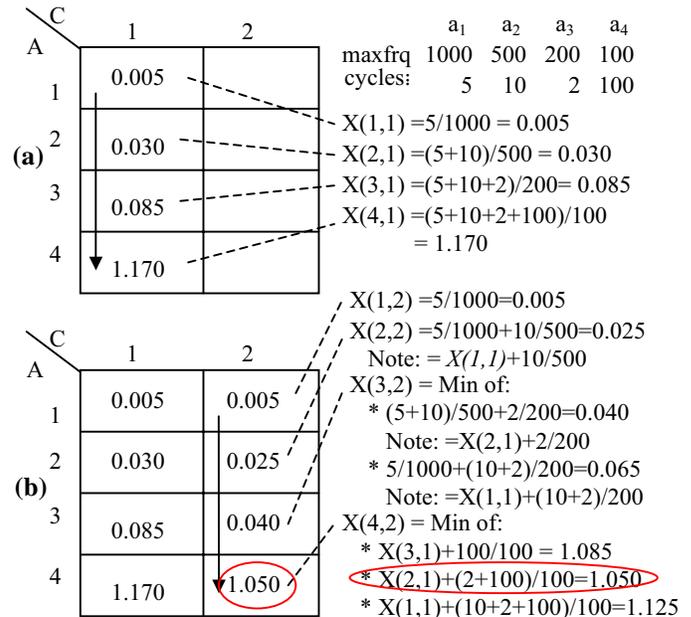
3. Dynamic Programming Solution

We considered several possible solutions to the clock-frequency assignment problem. Exhaustive search is feasible for small numbers of accelerators and possible frequencies, but grows at a quick rate and proved to be infeasible for practical problem sizes. In fact, we found that the number of solutions for n clock lines was equivalent to finding the n th Bell number of solutions, a well known combinatorial mathematics sequence whose complexity is factorial [18]. A heuristic could certainly be developed, but we suspected this problem contained enough structure that a polynomial-time optimal algorithm might be found.

3.1 Intuition

Examining the simple example of Figure 4 suggests the idea of first sorting the accelerators by their maximum frequency, resulting in $a_1, a_4, a_2,$ and a_3 , and frequencies of 100, 200, 500, and 1000. Consider the case of $F=1$. In that case, the problem has only one solution: 100, 100, 100, 100. Consider instead the case of $F=2$. In that case, a_1 would have to be 100. Solving for the remaining accelerators would represent a new sub-problem consisting of three accelerators, of $F=1$, and of an additional option of using a frequency of 100 for any of those three accelerators. Considering that sub-problem, and starting with the accelerator with the lowest maximum frequency, a_m , whose maximum frequency is 200, we can assign either 100 or 200 to a_m . If we assign 200, the sub-problem solution is done: 200, 200, 200, meaning the problem solution is: 100, 200, 200, 200. If we instead assign 100, then we again have a new sub-problem consisting of the remaining two accelerators, $F=1$, and the option of assigning 100. Noting the sub-problem structure in the problem, we investigated a dynamic programming solution.

Figure 5: Table forming basis of dynamic programming approach. Example has 4 accelerators ($M=4$) with maximum frequencies and cycles shown, and has 2 possible clock frequencies ($F=2$). Note that accelerators are sorted by maximum frequency and renumbered.



We developed the table structure in Figure 5 as the basis for a dynamic programming solution. The rows correspond to sub-problems with A accelerators. The columns correspond to sub-problems with C available clock frequencies. *Very importantly*, and without loss of generality, we assume that the accelerators have been *pre-sorted according to descending maximum frequency*, and that *each maximum frequency is distinct*. We shall justify these assumptions in Section 3.3.

The bottom right table cell represents the solution to the complete problem. Let $X(g,h)$ represent the cell with $A=g$ and $C=h$. Consider attempting to find a solution to the sub-problem represented by cell $X(1,1)$, illustrated in Figure 5(a). That sub-problem has one accelerator a_1 with maximum frequency of 1000, and one available frequency, so the only (reasonable) solution is obviously to assign a_1 a frequency of 1000. Since a_1 requires 5 cycles, the resulting execution time of that one accelerator sub-problem is $5/1000 = 0.005$ microseconds (assuming frequencies are in megahertz), which is entered into the table. Now consider moving down the column to cell $X(2,1)$. That sub-problem has two accelerators a_1 and a_2 with maximum frequencies of 1000 and 500, but has only one available frequency. Thus, the only solution is to assign both accelerators a frequency of 500. Since the accelerators require 5 and 10 cycles, respectively, the total execution time of those two accelerators is $(5+10)/500 = 0.030$. Continuing down the column, both remaining cells also have only one solution, with $X(3,1)$ requiring a frequency of 200, and $X(4,1)$ requiring a solution of 100, to be assigned to all accelerators. The computed execution times for those sub-problems are shown in the table.

Next, consider the top of the second column, cell $X(1,2)$. There is one accelerator a_1 , but two clock frequencies available. The only reasonable solution assigns the maximum frequency to the accelerator, yielding $5/1000 = 0.005$. Cell $X(2,2)$ has two accelerators and two clock frequencies available, so the only reasonable solution assigns the maximum frequency to each accelerator, yielding $5/1000 + 10/500 = 0.025$. The solution is the

same as $X(1,1)+10/500=0.025$; we had an available frequency, so we assigned the present accelerator a_2 its maximum frequency, and then used the best solution for $X(1,1)$ for the previous accelerator a_1 . This cell hints how we can reuse prior sub-problem solutions in computing the present sub-problems solution.

Cell $X(3,2)$ reveals such reuse more fully. That sub-problem has three accelerators a_1 , a_2 , and a_3 having maximum frequencies of 1000, 500, and 200, respectively, and has two available clock frequencies. Accelerator a_3 must be assigned a frequency of 200, because it has the lowest maximum frequency of the three accelerators (recall that the accelerators were initially presorted according to their maximum frequency). To complete the sub-problem solution, we have two choices. We can assume that a_3 is the only accelerator assigned 200, in which case the remainder of the sub-problem consists of the two accelerators a_1 and a_2 and one available frequency, i.e., $X(2,1)$. Alternatively, we can assume that frequency 200 is assigned to both a_3 and a_2 , in which case the remaining sub-problem would be $X(1,1)$. There is no need to consider assigning 200 to a_1 , because there is one remaining available frequency. Similar cell reuse exists with cell $X(4,2)$.

Thus, we see that a cell with $A=N$ can be computed by selecting the minimum of $N-1$ alternatives, where each alternative combines a simple new term with the results from a previous cell.

3.2 Dynamic Programming Formulation

We assume (without loss of generality) that accelerators a_1, a_2, \dots, a_M , have been pre-sorted in decreasing order of maximum frequency and that each maximum frequency is unique. Let $X(A,C)$ equal the total execution time of the first A accelerators using the first C clock frequencies. We define the following recurrence relation as a function:

If $(A=0)$ then $X(A,C)=0$
 Else If $(C=0)$ then $X(A,C)=\text{infinity}$
 Else

$$X(A,C) = \text{Min}_{i=1}^A \left(\frac{\sum_{j=i}^A a_j.\text{cycles}}{a_i.\text{mxfreq}} + X(i-1, C-1) \right)$$

If $A=0$, there are no accelerators, and thus the execution time is 0. If $C=0$, there are no clock frequencies available, so execution time is infinite. We intentionally define X to return 0 for $X(0,0)$.

The ‘‘Min’’ term compares the alternative solutions that assume the present accelerator’s (a_A) maximum frequency is assigned to the present accelerator only, to the present accelerator and the next accelerator, to the present accelerator and the next two accelerators, etc. The expression inside that term computes the total execution time for this cell as the sum of the execution times for the accelerators assigned to the present maximum frequency, added to the previously-computed best solution for the other accelerators with one less available clock frequencies.

3.3 Justification of Assumptions

Several observations must be established to justify assumptions we made in the dynamic programming formulation. For convenience of this discussion, we restate the problem definition in a form that partitions the accelerators into groups:

Partition the m accelerators into at most F groups, such that the total execution time as determined by the following equation is minimized:

$$E = \sum_{1 \leq i \leq N} \frac{\text{total cycles of group } i}{\text{min frequency of group } i} = \sum_{1 \leq i \leq N} \frac{\sum_j (a_{i,j}.\text{cycles})}{\text{MIN}(a_{i,j}.\text{freq})}$$

where $a_{i,j}$ denotes the j^{th} component in the i^{th} group of a solution.

We note that if the maximum number of available clock frequencies F is greater than or equal to the number of accelerators M , the solution is trivial – we just assign each accelerator a frequency equal to that accelerator’s maximum frequency. If F is less than M , some accelerators must be grouped to share a single clock frequency. We make three observations that allow us to formulate the dynamic programming algorithm.

Observation 1: *If $a_i.\text{freq} = a_j.\text{freq}$ (i.e., if two accelerators i and j have the same frequencies), then in the optimal solution, the two accelerators must be assigned to the same group.*

We prove this observation by contradiction. Assume $a_i.\text{freq}$ is equal to $a_j.\text{freq}$ but a_i and a_j are assigned to two different groups P_1 and P_2 with minimal frequencies min_freq_1 and min_freq_2 respectively. Without loss of generality, assume $\text{min_freq}_1 < \text{min_freq}_2$. If that is the case, we can move a_i from P_1 to P_2 , and it can be easily verified that the new solution as obtained will have a smaller total execution time (since a_i now has a faster clock), which contradicts our assumption of the optimality of the original solution. Therefore, according to the above observation, without loss of generality, we can assume that the frequencies of the given accelerators are all *distinct*. If not, we can simply combine those accelerators having identical frequencies into one single accelerator. The new accelerator will have the same frequency as that of the original accelerators, and its total cycles will equal the sum of the total cycles of the original accelerators.

Observation 2: *If $F < M$, then the optimal solution will always consist of F non-empty groups, i.e., all the available frequencies F will be used.*

Otherwise, we could split any group consisting of more than two accelerators into two groups and the new solution will have a smaller total execution time, which again contradicts our assumption of the optimality of the original solution.

Observation 3: *Let P_1, P_2 be any two groups taken from an optimal solution. Let min_freq_1 be the minimal frequency of the components in P_1 , max_freq_1 be the maximal frequency of the components in P_1 , and min_freq_2 and max_freq_2 be similarly defined. Then either $\text{min_freq}_1 \leq \text{max_freq}_1 < \text{min_freq}_2 \leq \text{max_freq}_2$ or $\text{min_freq}_2 \leq \text{max_freq}_2 < \text{min_freq}_1 \leq \text{max_freq}_1$ holds, i.e., either the components in P_1 are all slower than those in P_2 or they are all faster than those in P_2 .*

We prove observation 3 by contradiction. Assume to the contrary, in the optimal solution we have that $\text{min_freq}_1 < \text{min_freq}_2 < \text{max_freq}_1$. Let a_x be in the accelerator in P_1 with frequency max_freq_1 . We can move a_x from P_1 to P_2 and obtain a new solution. It can be easily verified that the new solution will require less total execution time since the execution time for accelerator a_x is reduced. This is a contradiction. The case where $\text{min_freq}_2 < \text{min_freq}_1 < \text{max_freq}_2$ can be similarly proved. Due to observation 3, we can first sort the accelerators by their maximum frequencies. The optimal solution will be guaranteed to observe this order, namely each group of the optimal solution will contain only consecutive accelerators from the sorted list.

3.4 Complexity

The algorithm fills a table of size $O(m * F)$, where m is the number of accelerators and F is the number of available clock frequencies. The algorithm visits each cell once. Filling each cell requires $O(F)$ time. Thus, the complexity of the algorithm is $O(m * F^2)$.

Because the algorithm finds the optimal solution in polynomial time, the clock-frequency assignment problem is clearly not NP-complete. Although the problem bears resemblance to the well-known NP-complete problem of bin-packing, the key difference is that, while the bin-packing problem has a capacity constraint on each bin, the clock-frequency assignment problem does not have a limit on the number of accelerators that may be assigned to each clock frequency.

3.5 Limitations

The formulation and solution do not consider physical design factors that might influence assignment of frequencies to accelerators, such as placement issues (accelerators with different frequencies may be placed farther apart, impacting routing) or size constraints that may exist on each clock domain – integrating physical design issues with higher-level exploration might be a useful extension, as is the case with most high-level design automation. The formulation did not consider the case of pre-existing fixed frequencies being available (perhaps pre-assigned to a circuit with a hard real-time constraint) – we suspect that extending our algorithm to take a partial solution as input would be straightforward, but have not investigated that yet.

4. Results

We implemented the dynamic programming algorithm and ran experiments on a 2.66 GHz 1Gb RAM Pentium 4 PC. We applied the algorithm to the 42 functions of the earlier-introduced H.264 decoder. We targeted synthesis of the 42 functions to a Xilinx Virtex IV Pro and gathered information on cycles per function accelerator, and maximum clock frequency of each accelerator. Figure 6 illustrates assignments obtained by the algorithm given three available clock frequencies. Obtained speedup was over 2x versus having just one frequency (with that speedup being in addition to already-obtained speedup from partitioning functions to accelerators). The algorithm ran in about 0.2 seconds.

We exercised the algorithm using synthetically-generated examples that supported large ranges of computation cycles and clock frequencies. Figure 7 provides results of applying the clock assignment on 10 synthetic benchmarks with varying numbers of accelerators ranging from 5 to 50. For each, we report speedups relative to execution time of the set of accelerators using only one clock frequency, which must necessarily be the lowest maximum frequency of the set. We considered available numbers of frequencies F of 3, 6, and 9. Figure 7 shows that partitioning an application among the available clock improves overall execution time by 1.5x-3x. Figure 8 shows algorithm runtimes for Figure 7. For normal-sized examples having 5-15 accelerators, the runtime is .1-.2 seconds. Even for the large examples having 50 accelerators, the runtime is still a reasonable .4-.8 seconds.

These runtimes suggest applicability of the algorithm as a sub-algorithm of a higher-level exploration tool, or even as part of future on-chip CAD tools that may dynamically partition accelerators among available clock domains. We developed a higher-level exploration tool that repeatedly applied our algorithm to determine the point of diminishing returns for number of available clock frequencies – such determination would be important for a system-level tool that allocates available clock frequencies to different sub-groups of modules. Figure 9 shows

Figure 6: Clock-frequency assignment for the H.264 decoder with three available clock frequencies. All accelerators in a group are assigned the lowest (circled) frequency in that group.

Accelerator	Clock Freq (MHz)	Cycles
..<MotionCompChromaFullXFullY>:	285	285
..<MotionComp_00>:	281	281
..<UpdateValidMBs>:	204	1224
..<GetBits>	200	200
...		
..<GetSignedUVLC>	106	212
..<DeblockingFilterLumaRow>:	80	3520
..<MotionCompChromaFracXFullY>:	74	1184
...		
..<MotionComp_31>:	70	15680
..<DeblockingFilterChromaRow>:	60	2640
..<MotionCompensate>:	40	400

Speedup versus one frequency: 2.04x

results for H.264, where we applied our algorithm ten times, for one available frequency, two available, three available,... or ten available. The data shows that H.264 performance improves up until about 4 or 5 available frequencies; more frequencies yield little additional performance improvements. We also consider four synthetic examples of different sizes, and again found tapering off of benefits at different points. That data could be used by a system-level exploration tool. For each example, the total tool runtime was under 4 seconds..

Another interesting possible use of the algorithm would be as part of a hardware/software partitioning approach. A hardware/software partitioning algorithm might apply clock-frequency assignment at certain points during exploration, to obtain accurate execution time feedback for a given partitioning of behavior among a microprocessor and accelerators.

5. Related Work

Several previous works used multiple clock frequencies to improve general-purpose processing. Semeraro [19] used multiple frequencies to reduce power of a single microprocessor. The clock domains included the front end (including L1 instruction cache), the integer units, the floating-point units, and the load-store units (including L1 data cache and L2 cache). Kumar [14] considered multiple frequencies for multiple heterogeneous microprocessors, where each microprocessor might be optimized for different application sets, each conceivably having its own clock frequency.

Some work has considered the similar topic of voltage islands. A voltage island is a sub-circuit operating at a different voltage, and typically therefore different clock frequency, compared to other islands [5]. Hu [11] considered mapping a set of cores, each with allowed voltage levels, into islands such that power was minimized. They used an iterative improvement heuristic, in particular simulated annealing, to group the cores into islands.

Numerous researchers (e.g., [24]) have considered the different concept of multiple voltage levels (and typically therefore multiple clock frequencies), namely the concept of voltage scaling of a single microprocessor. In such work, a microprocessor's voltage and clock may be dynamically adjusted to reduce power while still satisfying an application's performance constraints.

Figure 7: Execution time speedups for varying numbers of accelerators as a result of using multiple clock domains

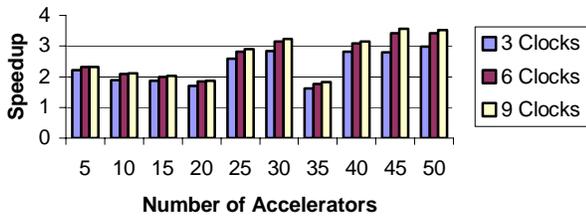
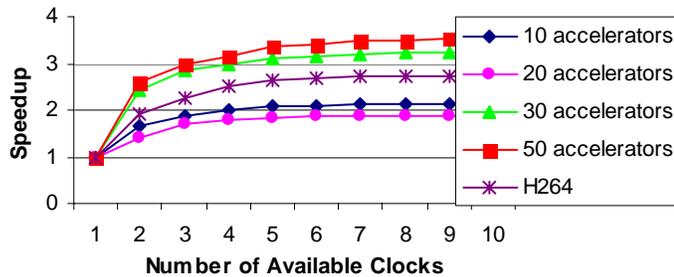


Figure 9 Applying the dynamic programming algorithm as part of a higher level exploration strategy.



We are not aware of prior work on clock domains, voltage islands, or voltage scaling, whose problem definition is equivalent to the clock-frequency assignment problem. To our knowledge, no hardware/software partitioning work (either multi-processor oriented or accelerator oriented) has considered multiple clock domains, due in part to such domains not having existed in cost-effective form until relatively recently.

Beyond such system-on-chip research, the problem of clustering items into a fixed number of groups is widely studied, such as the problem of quantizing a set of colors into a fixed number of colors, or of dividing a time series of data into a fixed number of straight-line segments[13]. The clock-frequency assignment problem differs from such clustering problems in the asymmetric constraint that an accelerator may belong to a group with a *lower* frequency than the accelerator's maximum frequency, but may *not* belong to a group with a *higher* frequency. Such asymmetry introduces more structure into the problem, allowing for an optimal algorithm with low runtime complexity.

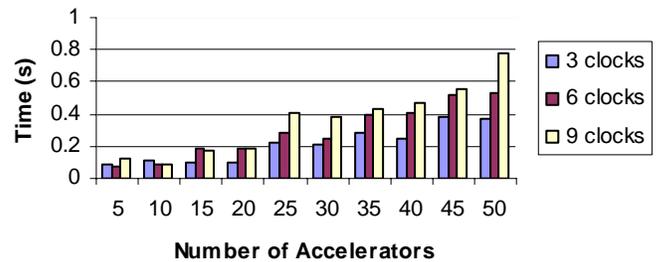
6. Conclusions

We showed that partitioning a microprocessor's accelerators among clock domains could yield application speedups of 1.5x-4x for applications with 5-50 accelerators, including a commercial H.264 decoder, and with 3-9 available clock frequencies. We defined the clock-frequency assignment problem for making use of multiple available clock frequencies, and developed a novel efficient dynamic programming algorithm that finds optimal results in polynomial time (thus showing that the problem is not NP-hard), and that runs in under a second for even very large examples. Future work will extend the problem to consider a wider range of clock-domain usage scenarios, and to integrate clock-frequency assignment with other exploration methods.

7. Acknowledgments

This work was supported in part by the National Science Foundation (CNS-0614957) and the Semiconductor Research Corporation (2005-HJ-1331), and by donations by Xilinx Corp. The authors would also like to thank Greg Stitt for gathering synthesis information relating to H.264.

Figure 8: Dynamic programming algorithm runtimes for varying numbers of accelerators and varying numbers of clock domains.



8. References

- [1] Atmel Corp. 2005. FPSLIC (AVR with FPGA), <http://www.atmel.com/products/FPSLIC/>.
- [2] Banerjee, S. and N. Dutt. Efficient search space exploration for HW-SW Partitioning. Int. Symp. on Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2004.
- [3] Celoxica. <http://www.celoxica.com>.
- [4] Chattopadhyay, A., and Z. Zilic. GALDS: A Complete Framework for Designing Multiclock ASICs and SoCs. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 6, June 2005.
- [5] Cohn, J.M., D.W. Stout, P.S. Zuchowski, S.W. Gould, T.R. Bednar, and D.E. Lackey. Managing power and performance for System-on-Chip designs using Voltage Islands. Int. Conf. on Computer-Aided Design (ICCAD), 2002, pp. 195-202.
- [6] Cray XDI. Cray Supercomputers. <http://www.cray.com/products/xd1/index.html>.
- [7] Critical Blue. 2005. <http://www.criticalblue.com>
- [8] Eles, P., Z. Peng, K. Kuchcinsky, and A. Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. Design Automation for Embedded Systems, vol2, no 1, 5-32 Jan. 1997
- [9] Excalibur. Altera Corp., <http://www.altera.com>
- [10] Gupta, R. and G. De Micheli. Hardware-Software Cosynthesis For Digital Systems. IEEE Design and Test of Computers. Pages 29-41, September 1993
- [11] Hu, J., Y. Shin, N. Dhanwada, and R. Marculescu. Architecting Voltage Islands in Core-Based System-on-a-Chip Designs. Int. Symp. on Low Power Electronics and Design (ISLPED), 2004, pp. 180-185.
- [12] Kalavade, A. and P.A. Subrahmanyam. Hardware/software partitioning for multi-function systems. IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 1997.
- [13] Keogh, E.J., S. Chu, D. Hart, and M.J. Passani. An Online Algorithm for Segmenting Time Series. IEEE Int. Conf. on Data Mining, pp. 289-296, 2001
- [14] Kumar, R., K.I. Farkas, N.P. Jouppi, P. Ranganathan, and D.M. Tullsen. Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction. Int. Symposium on Microarchitecture (MICRO), 2003.
- [15] Muttersbach, J., T. Villiger, H. Kaeslin, N. Felber, and W. Fichtner. Globally-Asynchronous Locally-Synchronous Architectures to Simplify the Design of On-Chip Systems. IEEE Int. ASIC/SOC Conference, 1999.
- [16] Mimosys. <http://www.mimosys.com/>.
- [17] Miyamori, T., and U. Olukotun. A Quantitative Analysis of Reconfigurable Coprocessors for Multimedia Applications. FPGAs for Custom Computing Machines (FCCM). 1998, pp. 2 - 11.
- [18] Rota, Gian Carlo. The Number of Partitions of a Set. American Mathematical Monthly., Vol 71. No 5 pp 498-504. 1964.
- [19] Semeraro, G., G. Magklis, R. Balasubramonian, D.H. Albonesi, S. Dwarkadas, and M.L. Scott. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. Int. Symp. on High-Performance Computer Architecture (HPCA), 2002.
- [20] Stitt, G., F. Vahid, and S. Nematbakshi. Energy Savings and Speedups From Partitioning Critical Software Loops to Hardware in Embedded Systems. IEEE Transactions on Embedded Computer Systems, January 2004.
- [21] Stitt, G., F. Vahid, G. McGregor, B. Einloth. Hardware/Software Partitioning of Software Binaries: A Case Study of H.264 Decode. Int. Conf. on Hardware/Software Codesign and System Synthesis (CODES/ISSS), Sep. 2005.
- [22] Virtex II and IV. Xilinx Corp., <http://www.xilinx.com>
- [23] Wall, D. Limits of instruction-level parallelism. ACM SIGARCH Computer Architecture News. Volume 19, Issue 2, pp. 176 - 188 April 1991.
- [24] Zhang, Y., XS Hu, and DZ Chen. Task scheduling and voltage selection for energy minimization. Design Automation Conference, 2002.