Configurable Computing: Technology and Applications, November. 2-3, 1998
Proceedings of SPIE Vol. 3526

Comparing Computing Machines

André DeHon

University of California at Berkeley, Soda Hall #1776, Berkeley, CA

ABSTRACT

Reconfigurable computing devices are emerging as a viable alternative to fixed-function components and programmable processors. To expand our knowledge of the role and optimization of these devices, it is increasingly imperative for us to compare implementations of tasks and subroutines across this wide spectrum of implementation options. The fact that most processors, FPGAs, ASICs, and memories are fabricated in a uniform technology medium, CMOS VLSI, where area scaling is moderately well understood eases our comparison task. Nonetheless, the rapid pace of technology, limited device size selection, and economic artifacts complicate the picture. In this paper, we look at the task of comparing computing machines, reviewing normalization techniques and many important issues which arise during comparisons. This paper includes examples intended to underscore the methodology and comparison issues, but does not attempt to make definitive conclusions about the merits of the technology alternatives from the small sample set. The immediate intent of this work is to help designers faced with tradeoffs between technological alternatives. The longer term intent is to help the community collect and analyze the broad-based data needed to better understand the range of available computing options.

Keywords: Configurable Computing, FPGA, Programmable Architectures, Performance Comparison, Computational Density

1. INTRODUCTION

There is a growing literature documenting cases where reconfigurable machines outperform select programmed processors and, occasionally, even custom ICs. These design points have fueled interest in this emerging field of configurable computation. This absolute performance quantification has been essential—justifying consideration of this computing style and showing us what is achievable.

However, absolute performance without costs leaves one wondering whether the superior performance arises from intrinsic features of the reconfigurable architecture, or whether it simply comes from the use of more silicon for the computation. Are the FPGAs using the silicon more efficiently such that they get greater performance with the same or less silicon? If the performance benefits are only in proportion to the greater amount of silicon used, do the FPGA solutions offer better scaling characteristics than alternatives? The earliest FPGA systems were built from boards containing tens of FPGAs, yet were often compared to microprocessors or single ASICs, so the answers to these questions are not immediately obvious without some careful normalization.

For a perfect comparison, we would prefer to compare systems solving the same problem, using the same fabrication technology, and using equal amounts of silicon or achieving equivalent performance levels. Unfortunately, it is seldom feasible for a single research effort to develop all such implementations, and even if it were, the rapid advance of silicon technology makes it impractical to fully standardize on a single level of technology. As an alternative, we must often pick best-in-class implementations from the literature for comparisons. This, inevitably, leads to differences in technology, area, and performance.

In this paper, we explore several issues associated with cost normalization in cross technology and implementation style comparisons. We do not provide a single figure of merit to address all comparisons, but rather suggest a number of viewpoints and metrics which are useful in characterizing the cost/performance of an implementation. We draw heavily from the published literature in order to illustrate these points. The data points included here were selected based on data availability and do not necessarily make up a representative set of application requirements. We also make recommendations on the data which should be collected and reported to make design points most useful for broad analysis and comparison.

author E-mail: andre@acm.org

The next section emphasizes the role and importance of normalized comparisons for today's system designers. We then review CMOS technology scaling (Sections 3 and 4) to motivate proper comparisons across technology generations, and we take a look at what resources a task actually consumes (Section 5). We focus on a specific figure of merit which is useful for high throughput tasks in Section 6, and then review a number of others which may be useful in various situations in Section 7. Section 8 comments briefly on how area and cost may diverge downstream from fabrication. We make our suggestions for reporting performance in Section 9.

2. VIEWPOINT AND IMPORTANCE

Why should we bother quantifying implementations and comparing them?

Today's designer at the system, board, and IC level is faced with the question of what to implement in custom silicon (ASICs, embedded macros), on processors (stand-alone or embedded cores), or in reconfigurable arrays (FPGAs, embedded arrays). An important goal of comparisons between implementation options is to help the system designer make these tradeoffs.

If the designer knows a particular class of subtasks generally achieve the same level of performance with one tenth area in a custom implementation compared to an FPGA implementation, he can then decide when it is better to include multiple, specialized macros which may not be used simultaneously, versus when to include a flexible array. For example, if the designer knows he only needs one of four different functions he might be better off including custom silicon for all four function even if only one is used at a time. Alternately, if he needs one of 40 different functions, the flexible array may provide a superior alternative.

Similarly, if the designer knows a class of applications runs in one tenth of the time on an FPGA versus a processor of comparable area, he can reason about the merits of implementing the function on either. If the throughput required from the task is sufficiently high to keep the FPGA continuously busy, the FPGA may be the better option. However, if the throughput required from the task is less than one-tenth of that offered by the FPGA and there are other tasks which need to be run in the system, the processor may be able to implement the collection of tasks in less area than the FPGA(s).

Documented FPGA-based implementations potentially provide value to the system designer by:

- showing the area, time, and power capacity required for an important computational subtask
- highlighting the techniques employed to achieve the design point(s)

Capacity comparisons allow the designer to make up-front architectural judgments and first-pass partitioning estimates before committing significant resources to developing the details of a particular application(s).

3. CHIP $\neq$ CHIP

How do we characterize implementation resource costs?

All chips are not created equal. Even limiting ourselves to a common technology media, CMOS VLSI, chips vary in die size and feature size. This is fairly obvious when we realize an XC2064 and an XC4025[1] require very different amounts of raw silicon resources. Similarly, a Pentium requires substantially more resources than an 8086, and a 256K$\times$4 SRAM requires much more than a 2K$\times$8 SRAM. Even in the same technology, there may be considerable variation in die sizes from part to part. The XC4005, for instance, is certainly smaller than an XC4013 when both are implemented in the same 1.2μm process.

When we talk about an IC, we need to qualify that with its size – *i.e.* the silicon area which it represents. However, a square millimeter in one technology is not equal to a square millimeter in a technology with a different feature size. Smaller geometries allow us to pack more functions into each square millimeter of silicon area. To capture this effect, it is worthwhile to measure silicon area in feature-size units rather than in absolute units.

Lambda (λ) is the typical unit used to characterize a MOS VLSI process. One lambda is defined as half the minimum drawn feature size on a process. Typically processes are named by the minimum transistor channel width and lambda is half this value. So a 1.0μm CMOS process would have $\lambda = 0.5\mu$m. As long as all features shrink uniformly, a die or macro will occupy the same λ^2 area as feature geometry shrinks. For example, in Ref. 2, Intel

describes 0.8μm and 0.6μm implementations of the Pentium. The 0.8μm die is 284mm^2 or $\frac{284\times10^6\mu\text{m}^2}{(0.4\mu\text{m}/\lambda)^2} = 1.78\text{G}\lambda^2$, while the 0.6$\mu$m die is 163mm^2 or $\frac{163\times10^6\mu\text{m}^2}{(0.3\mu\text{m}/\lambda)^2} = 1.81\text{G}\lambda^2$.

Processes, of course, have many different parameters and not all processes with the same feature size are equivalent. Different numbers of metal layers at the same feature size will, for example, yield different feature densities. Nonetheless, processes are generally optimized in order to make densities track feature size (*e.g.* Ref. 3, Ref. 4). As such, lambda normalized area generally gives a good estimate of the area required to implement a die or macro, usually within 20%.

As a result, this uniform media normalization allows us to compare the area real-estate required for microprocessors, ASICs, memories, and FPGAs broadly across the CMOS family of technologies. Further, the common area metric allows us to calculate an aggregate area estimate for multi-IC systems.

4. TIME

Just as IC dies vary widely and are not a good normalizing metric for area, clock cycles also vary widely and are not a good normalizer for time. As we saw in the CISC versus RISC debates over a decade ago, it is often possible to execute more, simple cycles in less time than fewer, more computationally loaded cycles. What matters is not how fast each cycle is or how much gets done between clock ticks, but the product of these two factors – *i.e.* the amount of time it takes to complete a task or the data throughput rate. When comparing two implementations using the same basic technology level, absolute time (*e.g.* seconds) or throughput (results/second) is the best time unit for comparison.

As features sizes shrink, intrinsic delays will also shrink. If voltage is scaled along with λ, gate delays will also scale with λ. However, wire transit times, which are becoming an increasingly significant fraction of cycle time, are not scaling down at this rate. Consequently, the cycle time reduction associated with feature size scaling is not as clean as the area scaling. This effect is exacerbated by the fact that device voltages remained at 5V for many technology generations then made a discrete jump to 3.3V, rather than scaling along with feature size.

E.g. consider the aforementioned Pentium shrink. The 0.8μm version ran at 66MHz on a 5V supply, while the 0.6μm version ran at 100MHz on a 3.3V supply. The 25% reduction in feature size accompanied by a 33% reduction in supply voltage gave a 33% reduction in cycle time.

When comparing technologies with close features sizes (10-20%), the speed difference is usually in the noise for this level of comparison and absolute times can be used. When comparing large difference in technology (*e.g.* factor of two or more in feature size), it is worthwhile to be aware of potential differences in device speeds.

5. WHAT TO INCLUDE?

How much of the IC or system area do we "charge" to a task.

The trivial answer is to sum up all the IC areas in the entire system and take that as a number. For tasks which truly consume all of a system's resources this is certainly a reasonable metric. However, it is often the case that we are looking at subtasks which may co-exist and operate along with other tasks. For example, a data encoder may take up 300 CLBs, while leaving room for other functions to be implemented simultaneously on other parts of the system or even in the same FPGA. Similarly, a decoding function running on a microprocessor may require 1000 processor cycles for each input symbol, but if each input symbol only arrives once every 10,000 processor cycles, 90% of the capacity of the processor is available to perform other functions.

Taking a macro or subtask view, it is worthwhile to separate out the area actually consumed by each task. This selection can requires some judgment about resource consumption and bottlenecks. We define the amount of a resource which is consumed as that fraction which cannot be used for other purposes while the task is running. *E.g.*

- Basic logic blocks – the blocks assigned to a subtask are consumed by that task and should be charged to it. Where possible this should also include any additional basic logic block area which is required to route the task; *e.g.* if a 70 CLB macro can only be routed in a 9×10 grid, 90 CLBs would be a better estimate of the area consumed than 70.

- Processor cycles – conventional microprocessors do not allow independent, concurrent execution, so each processor cycle consumes the entire processor area. Since processors can change their operation from cycle to cycle, they can effectively be time-sliced making it worthwhile to charge a limited throughput task a prorated area in proportion to the fraction of the processor's cycles required to achieve that throughput.

  Note that a processor may also be fully utilized when it still has spare compute cycles if its instruction storage area is fully consumed.

- Memory – memory may be consumed in one of many ways and it is worthwhile to figure out which demand is limiting and is the best estimate of memory consumption:

  - attached unit – if a memory is rigidly attached to some device, it may be consumed when the attached device is consumed whether or not the memory i/o bandwidth or storage capacity is fully utilized. In this case, the memory is consumed because it cannot be used for any purpose other than the use extracted by the attached device.
  - i/o bandwidth – memories typically have a single i/o port. When the i/o bandwidth is fully utilized, the entire memory is consumed regardless of the amount of unused memory inside. If i/o bandwidth is limited and the memory can be shared between tasks, memory area should be charged in a prorated fashion like processor area.
  - storage capacity – memories have a fixed size. When memories are free to be used for many purposes and the i/o bandwidth is not in danger of being saturated, they are consumed in proportion to their storage capacity allocated to a task.

- Fixed function device – an ASIC or fixed-function macro is generally fully consumed by a task.

  - If it needs to maintain significant state between operations, it cannot be time-sliced as the processor.
  - If it has minimal state and spare i/o and controller bandwidth, its use may be pro-rated in proportion to usage as per the processor, if it is likely that its function will be useful for other, simultaneously operating tasks.

6. THROUGHPUT DENSITY

How do we roll this area and time information together to compare among implementations?

Normalizing throughput to implementation area provides a throughput density metric which is easy to calculate and appropriate when the throughput requirement is very high as is the task concurrency. This provides a good measure of the area cost required to achieve various throughput levels and a good basis for comparing implementations.

To be specific, when we say the throughput requirement is high, we mean the desired throughput for the task is either unbounded or bounded at some fixed rate faster than a single hardware instance can achieve. For example, in applications like data searching, sorting, or numerical simulation, we often want to process data as fast as possible. In some signal processing and video coding applications, even a single, custom processing element cannot achieve reasonable throughput requirements; it is instead necessary to exploit problem concurrency to achieve the requisite throughput.

In these cases, knowing the normalized area which provides a given throughput gives us a good estimate of the efficiency of the implementation. If we want to implement a particular throughput level, then the implementation with the highest throughput density metric will admit the smallest implementation. Or, if we are willing to dedicate a particular amount of area to a task, the implementation with the highest throughput/area metric will provide the most throughput in the fixed area. Of course, this assumes (1) there is adequate parallelism in the problem to admit replication and (2) composition overhead is negligible such that replication of the basic unit will achieve essentially linear scaling above the measured point.

In the remainder of this section, we will illustrate this throughput density metric by revisiting several familiar tasks from the literature.

Architecture	Design	Feature Size (λ)	Area and Time	16×16 mpy $\frac{}{\lambda^2 s}$	16×16 scale $\frac{}{\lambda^2 s}$	8×8 mpy $\frac{}{\lambda^2 s}$	8×8 scale $\frac{}{\lambda^2 s}$
Custom 16×16	5	0.63μm	2.6Mλ^2, 40 ns	9.6	9.6	9.6	9.6
Custom 8×8	6	0.80μm	3.3Mλ^2, 4.3 ns			70	70
Gate-Array 16×16	7	0.75μm	26Mλ^2, 30ns	1.3	1.3	1.3	1.3
FPGA	XC4K[1]	0.60μm	1.25Mλ^2/CLB	0.097	0.24	0.30	1.5
	8		16×16 | 316 CLBs, 26 ns				
	9		16×16c | 84 CLBs, 40 ns				
	10		8×8 | 220 CLBs, 12.1 ns				
	9		8×8c | 22 CLBs, 25 ns				
16b DSP	11	0.65μm	350Mλ^2, 50 ns	0.057	0.057	0.057	0.057
RISC	12	0.75μm	125Mλ^2, 66 ns/cycle	0.0028	0.017	0.0051	0.030
(no multiplier)	13		16×16 | 44 cycles				
			16×16c | 7 cycles				
			8×8 | 24 cycles				
			8×8c | 4 cycles				

Table 1. Multiplier Throughput Comparison

Architecture	Reference	Feature Size (λ)	Area and Time	TAPs $\frac{}{\lambda^2 s}$
32b RISC	12	0.75μm	125Mλ^2, 66 ns/cycle×6+cycles/TAP	0.020
16b DSP	11	0.65μm	350Mλ^2, 50 ns/TAP	0.057
32b RISC/DSP	14	0.25μm	1.2Gλ^2, 40 ns/TAP	0.021
64b RISC	15	0.18μm	6.8Gλ^2, 2.3 ns/TAP	0.064
FPGA	XC4K[16]	0.60μm	240 CLBs, 14.3 ns/8-TAPs	1.9
	Altera 8K[17]	0.30μm	30 LEs×0.92Mλ^2/LE, 10 ns/TAP	3.6
Full Custom	18	0.75μm	400Mλ^2, 45 ns/64 TAPs	3.6
	19	0.60μm	140Mλ^2, 33 ns/16 TAPs	3.5
	20	0.75μm	82Mλ^2, 50 ns/10 TAPs	2.4
(fixed coefficient)	21	0.60μm	114Mλ^2, 6.7 ns/43 TAPs ($n.b.$ 16b samples)	56

Table 2. FIR Throughput Comparison – 8b sample, 8b coefficient

Multiply Table 1 shows throughput comparisons across fixed, programmable, and reconfigurable architectures for four multiplication tasks. The "scale" problems are for the case where one operand is a constant, while both operands are variables in the "mpy" cases. Since multiplies have been highly studied in the literature, it was possible to select a set of custom, DSP, and processor implementations for comparison which have feature sizes moderately close to that of the FPGA.

Not surprisingly, the full-custom implementations achieve the greatest performance density. The FPGA implementations achieve higher raw multiply throughput than the DSP despite the fact the DSP includes a custom multiplier; this, too, is unsurprising when you realize the multiplier makes up only 5-10% of the area on the DSP die, with the rest going largely to instruction control, and instruction and data storage. The RISC processor without a custom multiplier achieves the lowest density since it must synthesize the multiply operation out of many primitive ALU operations.

FIR Finite Impulse Response (FIRs) have achieved considerable attention in the FPGA literature since they easily admit to systolic implementation, and they have shown good performance on FPGAs. Table 2 summarizes the performance of processor, DSP, FPGA, and custom FIRs for low-precision FIR computations. The custom

Architecture	Reference	Feature Size (λ)	Area and Time	16b $\frac{\text{TAPs}}{\lambda^2 s}$	10b $\frac{\text{TAPs}}{\lambda^2 s}$
16b DSP	22	0.60μm	200Mλ^2, 500 ns/biquad	0.010	0.010
FPGA	XC4K[16]	0.60μm	16b — 60 CLBs, 320 ns/biquad 10b — 43 CLBs, 200 ns/biquad	0.044	0.093
Full Custom	23	0.90μm	68Mλ^2, 11.8 ns/4 biquads		5.0

Table 3. IIR Throughput Comparison

Architecture	Reference	Feature Size (λ)	Area	Keys/Second	$\frac{\text{Keys}}{\lambda^2 s}$
DES IC	24	1.5μm	11.1Mλ^2	310K	0.028
FPGA	Altera 8K[25]	0.30μm	81188 (930Mλ^2)	800K	0.00086
RISC	2, 25	0.30μm	1.8Gλ^2	41K	0.000023

Table 4. DES Key Search

implementations all have feature sizes in the 1.2-1.5μm range, comparable to the Xilinx XC4K FPGA, the 16b DSP, and the 32b RISC shown. The Altera FPGA and the larger RISC processor datapoints have feature sizes which are 2-4$\times$ smaller than this range so may merit some speed normalization.

The surprising observation from this comparison is that the FPGAs implementations have comparable throughput density to the custom implementations which allow programmable coefficients. However, compared to the custom implementation with fixed coefficients, the FPGA implementation is 10-30$\times$ less dense. We also see here that the FPGA FIRs are 30$\times$ more dense than the DSPs, underscoring the reason for the high interest in FPGAs for this application.

IIR Table 3 contains a similar comparison for biquads, the core computation for Infinite Impulse Response (IIR) computations. Here the multipliers cannot be specialized around the coefficient, so the FPGA suffers a full 50$\times$ density penalty versus a programmable, custom IC for IIR. While the FPGA is still denser than the DSP, the advantage is only 4$\times$ in density for same size task. As the precision drops, the FPGA advantage relative to the DSP does improve.

DES Search In a class project[25] Berkeley students compared the effectiveness of FPGAs for brute-force DES key search. The major FPGA and processor results along with a custom IC are shown in Table 4. The custom IC is on a much older process, so probably deserves a 2-4$\times$ speed normalization versus the FPGA and processor which are both fabricated in 0.6μm processes. While the FPGA implementation is actually faster than the given custom silicon implementation, it achieves this speed with 80$\times$ the normalized silicon area.

This is a good example of the kind of task for which the throughput density metric is ideal. The brute-force search problem can be trivially parallelized, and desirable throughput rates are certainly larger than even a single custom device will provide. An attacker might want to either spend as little money as possible to achieve a given search rate, or achieve as high a search rate possible given limited funds.

Sequence Matching DNA sequence matching on SPLASH was one of the earliest successes of FPGA-based computing machines. Table 5 normalizes the throughput data provided in Chapter 8 of Ref. 27.

As shown along with the table, the areas for the SPLASH implementations were approximated by their major components – FPGAs and memories. For SPLASH 2, the omission of the 9 crossbars may make the estimate a bit optimistic. Both workstations were estimated simply as their processor area; this, too, may be optimistic since the first SPARCs had no on-chip data caches and even the SuperSparc cache is too small to hold the data required in this computation. However, none of these omissions is going to change the total by more than a factor of two.

Architecture	Reference	Feature Size (λ)	Area	Cell Updates per Second	$\frac{cu}{\lambda^2 s}$
Custom	26	2.0μm	270Mλ^2	500M†	1.9
FPGA					
(SPLASH 2)	27	0.60μm	43Gλ^2	3,000M	0.070
(SPLASH)	28	0.60μm	33Gλ^2	370M	0.012
RISC					
(SparcStation I)	29	0.75μm	273Mλ^2	0.87M	0.0032
(SparcStation 10)	30	0.40μm	1.6Gλ^2	1.2M	0.00075

† – This number comes via direct calculation from Ref. 26 and not the test setup in Ref. 31.

For purposes of this comparison, silicon content is approximated as follows:

P-NAC	34 components (7.8Mλ^2 each)
SPLASH 2	17 XC4010s (500Mλ^2 each) + 17 256K×16 SRAMs (2Gλ^2 each)
SPLASH	32 XC3090s (420Mλ^2 each) + 32 128K×8 (600Mλ^2 each) SRAMs
Sparc I	SPARC CPU
Sparc 10	SuperSparc CPU

Table 5. DNA Sequence Matching

We see here that most of the 400× advantage which SPLASH exhibited over the SparcStation I came from its use of greater silicon area. However, as noted in Ref. 28, SPLASH was handicapped at one-tenth its potential throughput by the low-bandwidth i/o. Fixing this deficiency was one of design objectives for SPLASH 2, and we see that it does achieves a 6× throughput density advantage over the original SPLASH.

7. OTHER FIGURES OF MERIT

While throughput density tells the story for high throughput, parallelizable operations, this certainly does not cover all the cases we might care about. Other metrics of interest include:

- area required to achieve some low, fixed throughput rate
- performance achievable within a fixed area budget
- performance achievable given a fixed power budget

In the cases with adequate parallelism and high throughput requirements we were able assume a linear area-time relationship in the region of interest. Note that the area-time curve will not necessarily be ideal below certain points – *e.g.* a 2× increase in compute time will not necessarily allow a 50% reduction in area. While the regular, concurrent portions of a task running on an FPGA can be scaled down with decreasing throughput requirements, residual control and irregular operations are often left.

FIR To illustrate, Figure 1 shows the area-time curve for a 16-TAP FIR implemented in custom silicon, a Xilinx FPGA, and a DSP. It takes 800 ns to evaluate each filter output value on the DSP.[11] Below this point, the DSP area scales linearly assuming we have other tasks which can be scheduled between filter evaluations. The custom FIR component[19] can perform one evaluation of the filter every 33 ns. However, the hardwired silicon requires the same amount of area regardless of the how much slower it is run. Of course, a different custom FIR component could be built at a different area-time point, if one is able to select the FIR throughput before fabrication. Different styles can be used in the FPGA implementation[16] depending on the application throughput requirements. There is, however, a minimum area configuration for the FPGA below which further throughput reduction does not allow the area of the implementation to be compressed further.

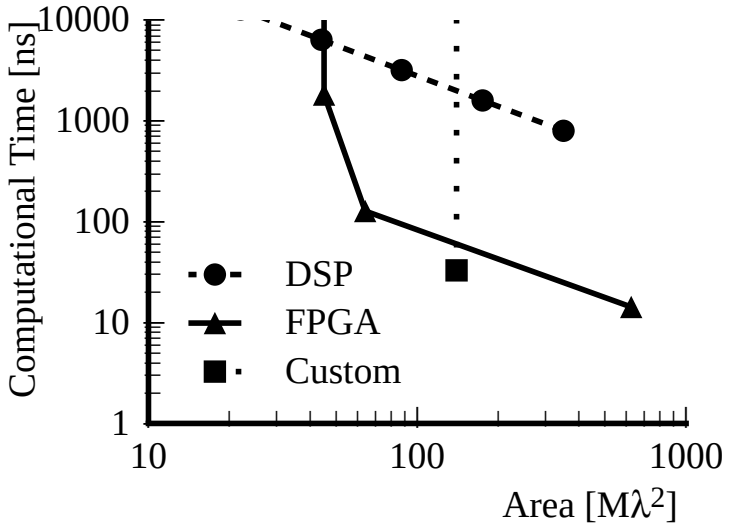

Figure 1. Area-Time Curve for 16 TAP, 8b FIR

This defines an interesting design space, worth understanding when allocating resources for an implementation. For limited throughput tasks, we need to look at the task throughput requirements to best rank the options. In the 0.55–8.0MHz region, the FPGA is the smallest implementation, with its advantage over the DSP diminishing towards the lower end of that region. Below 0.55MHz, the DSP implementation requires the least area.

8. COST VIEWPOINTS

Using area as a normalizer is a good cost metric from the viewpoint of the silicon designer. Silicon area is the limited resource constraining design at this level. If the designer is allocating space on the same die, area is exactly the commodity of trade—allocating $500M\lambda^2$ to FPGA resources must come at the expense of that $500M\lambda^2$ being used for custom logic or other programmable structures. Downstream from the silicon designer, however, costs may diverge from silicon area for several reasons. *E.g.*

- Actual fabrication costs in advanced fabs will be higher per λ^2 than more mature, less dense, processes – especially during the learning curve for the fab and the design when yields are low.
- One tends to pay a premium for the highest density and largest dies.
- Gate arrays will cost less per λ^2 than full-custom at the same feature size since they require less custom processing.
- Price to an end user reflects customer valuation and volume demand rather than raw material costs. A $G\lambda^2$ of silicon in a high-volume, commodity part will be less expensive to board designers buying off-the-shelf parts than a $G\lambda^2$ of silicon in a specialized, custom IC.
- At the board level, non-silicon costs accrue in packaging, inter-chip interconnect, and handling.

One can normalize for these costs, but the normalized results tell us more about the current economic picture than the intrinsic resource costs. That is, such selling price normalized metrics may be accurate at one point in time for low-volume, board-level component consumers, but will not remain stable or accurate across shifts in component supply and demand.

9. REPORTING PERFORMANCE

When analyzing or reporting FPGA applications, it is generally worthwhile to report the area, time, and energy requirements for the task. When all of these basic quantities are included, it is possible to normalize the results as suitable for various application requirements as noted in the previous sections.

- **Area** – as noted in Section 5, the area includes the type and number of basic logic blocks (or fraction of a particular FPGA), memories, and fixed components consumed by the application. When using standard parts, the feature size is often implicit in the type of part used. When using custom parts, the die size and feature sizes of the custom parts should be included as well to allow normalization as per Sections 3 and 4.
- **Time** – use the absolute time taken to complete a task, or the absolute throughput rate (Section 4).
- **Energy** – where possible, include the energy required to perform a task or process each input or output; equivalently, the power required to achieve a given computational rate provides the same information.

Of course, where possible providing or describing the area-time curve for the FPGA implementation will help one understand how the FPGA solution adapts to various throughput requirements. In the future it may also be interesting to explore area-power curves as well.

10. SUMMARY

FPGAs are universal computing structures. Consequently, we know *a priori* that we can perform any computable function using an FPGA-based system. The real question to answer when assessing the quality of an implementation is: *How well we can perform a particular task on an FPGA-based machine relative to the alternatives?* Further, FPGAs are very scalable on highly concurrent tasks. Absolute performance figures are interesting, but by themselves they do not tell us if the FPGA solution is efficient or even superior to its alternatives. To understand the efficiency of FPGA solutions, we need to normalize the delivered performance to the cost of the solution in terms of critical resources such as area and power. This normalization tasks is eased by the fact that most of today's computing components are built in CMOS VLSI. Thus, we can use the area in the underlying silicon implementations as a normalizer to measure the area cost of an implementation. This metric is particularly relevant to silicon designers, but also provides a useful basis for general comparisons. Throughout this paper we have detailed issues in this cross-architectural comparison and illustrated the comparison using several common applications from the literature.

Acknowledgments:

This research is supported by the Defense Advanced Research Projects Agency under contract numbers F30602-94-C-0252 and DABT63-C-0048.

Thanks to Krste Asanovic, Timothy Callahan, John Hauser, Dzung Hoang, and John Wawrzynek for reviewing early drafts of this paper.

REFERENCES

1. Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124, *The Programmable Logic Data Book*, 1994.
2. J. Schultz, "A 3.3v 0.6μm bicmos superscalar microprocessor," in *1994 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 202–203, IEEE, February 1994.
3. M. Bohr, "Mos transistors: Scaling and performance trends," *Semiconductor International* , pp. 75–79, June 1995.
4. M. Bohr, "Interconnect scaling – the real limiter to high performance ulsi," in *International Electron Devices Meeting 1995 Technical Digest*, pp. 241–244, Electron Devices Society of IEEE, December 1995.
5. J. Fadavi-Ardekani, "$m \times n$ booth encoded multiplier generator using optimized wallace trees," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **1**, pp. 120–125, June 1993.
6. D. Somasekhar and V. Visvanathan, "A 230-mhz half-bit level pipelinined multiplier using true single-phase clocking," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **1**, pp. 415–422, December 1993.
7. G. Boudun, P. Mollier, J. Nuez, and F. Wallart, "A 30ns-32b programmable arithmetic operator," in *1987 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 54–55, IEEE, February 1987.

8. T. Isshiki and W. W.-M. Dai, "High-level bit-serial datapath synthesis for multi-fpga systems," in *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 167–173, ACM, February 1995.
9. K. D. Chapman, "Fast integer multipliers fit in fpgas," *EDN* **39**, p. 80, May 12 1993.
10. C.-J. Chou, S. Mohanakrishnan, and J. B. Evans, "Fpga implementation of digital filters," in *International Conference on Signal Processing Applications and Technology*, 1993.
11. K. Kaneko, T. Nakagawa, A. Kiuchi, Y. Hagiwara, H. Ueda, and H. Matsushima, "A 50ns dsp with parallel processing architecture," in *1987 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 158–159, IEEE, February 1987.
12. J. Yetter, M. Forsyth, W. Jaffe, D. Tanksalvala, and J. Wheeler, "A 15 mips 32b microprocessor," in *1987 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 26–27, IEEE, February 1987.
13. D. J. Magenheimer, L. Peters, K. Pettis, and D. Zuras, "Integer multiplication and division on the hp precision architecture," in *Proceedings of the Second International Conference on the Architectural Support for Programming Languages and Operating Systems*, pp. 90–99, IEEE, 1987.
14. K. Nadehara, M. Hayashida, and I. Kuroda, *A Low-Power, 32-bit RISC Processor with Signal Processing Capability and its Multiply-Adder*, vol. VIII of *VLSI Signal Processing*, pp. 51–60. IEEE, 1995.
15. P. Gronowski *et al.*, "A 433mhz 64b quad-issue risc microprocessor," in *1996 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 222–223, IEEE, February 1996.
16. B. Newgard, "Signal processing with xilinx fpgas." `<http://www.xilinx.com/apps/appnotes/sd_xdsp.pdf>`, June 1996.
17. Altera Corporation, 2610 Orchard Parkway, San Jose, CA 95134-2020, *AN 73: Implementing FIR Filters in FLEX Devices*, January 1996. `<http://www.altera.com/document/an/an073_01.ps>`.
18. P. Ruetz, "The architectures and design of a 20-mhz real-time dsp chip set," *IEEE Journal of Solid-State Circuits* **24**, pp. 338–348, April 1989.
19. C. Golla, F. Nava, F. Cavallotti, A. Cremonesi, and G. Casagrande, "30-msamples/s programmable filter processor," *IEEE Journal of Solid-State Circuits* **25**, pp. 1502–1509, December 1990.
20. D. Reuver and H. Klar, "A configurable convolution chip with programmable coefficients," *IEEE Journal of Solid-State Circuits* **27**, pp. 1121–1123, July 1992.
21. J. Laskowski and H. Samueli, "A 150-mhz 43-tap half-band fir digital filter in 1.2-μm cmos generated by silicon compiler," in *Proceedings of the IEEE 1992 Custom Integrated Circuits Conference*, pp. 11.4.1–11.4.4, IEEE, May 1992.
22. A. Picco, J. C. Michalina, B. Laurier, D. Fuin, P. Menut, and J. Laborie, "The st18940/41: An advanced single-chip digital signal processors," in *Proceedings of the 1989 IEEE International Symposium on Circuits and Systems*, pp. 1559–1562, IEEE, May 1989.
23. M. Hatamian and K. K. Parhi, "An 85-mhz fourth-order programmable iir digital filter chip," *IEEE Journal of Solid-State Circuits* **27**, pp. 175–183, February 1992.
24. I. Verbauwhede, F. Hoornaert, J. Vandewalle, and H. J. De Man, "Security and performance optimization of a new des data encryption chip," *IEEE Journal of Solid-State Circuits* **23**, pp. 647–656, June 1988.
25. I. Goldberg and D. Wagner, "Architectural considerations for cryptanalytic hardware." CS252 Report `<http://www.cs.berkeley.edu/~iang/isaac/hardware/>`, May 1996.
26. R. Lipton and D. Lopresti, "A systolic array for rapid string comparison," in *1985 Chapel Hill Conference on VLSI*, H. Fuchs, ed., pp. 363–376, 1985.
27. D. Buell, J. Arnold, and W. Kleinfelder, *Splash 2: FPGAs in a Custom Computing Machine*, IEEE Computer Society Press, 10662 Los Vasqueros Circle, PO Box 3014, Los Alamitos, CA 90720-1264, 1996.
28. M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti, "Building and using a highly programmable logic array," *IEEE Computer* **24**, pp. 81–89, January 1991.
29. L. Quach and R. Chueh, "Cmos gate array implementation of sparc," in *Digest of Papers COMPCON'88*, pp. 14–17, IEEE, Februrary 1988.
30. F. Abu-Nofal *et al.*, "A three-million-transistor microprocessor," in *1992 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pp. 108–109, IEEE, February 1992.
31. D. Lopresti, "P-nac: A systolic array for computing nucleic acid sequences," *IEEE Computer* **20**, pp. 98–99, July 1987.