Instruction Subsetting: Trading Power for Programmability

William E. Dougherty, David J. Pursley, Donald E. Thomas
[wed,pursley,thomas]@ece.cmu.edu
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213

Abstract

Power consumption is an increasingly important consideration in the design of mixed hardware/software systems. This work defines the notion of instruction subsetting and explores its use as a means of reducing power consumption from the system level of design. Instruction subsetting is defined as creating an application specific instruction set processor from a more general processor, such as a DSP. Although not as effective as an ASIC solution, instruction subsetting provides much of the power savings while maintaining some level of programmability. Instruction set choice strongly affects the savings. We synthesized 5 ASIPs through place and route and found that a poorly chosen instruction set may consume more than 4 times the energy of an ASIP with a proper instruction set choice. This finding will allow designers to consider another set of trade-offs in their hardware/software design space exploration.

Introduction

The performance of a fully customized ASIC design can be optimized in terms of a variety of parameters such as critical path, execution speed, area, or power dissipation. Programmable implementations may not be optimized for any of the performance parameters but can be reprogrammed for a variety of applications. This work defines the notion of *instruction subsetting* and uses it as a technique to trade-off performance and programmability.

Consider the range of implementation styles shown in Figure 1. If we have a behavioral description, or a software program, that is to be implemented, the system can be designed as software running on a general purpose processor (GP), as software on a DSP processor, as an application specific instruction set processor (ASIP), or as an application specific integrated circuit (ASIC). These alternate implementations represent a trade-off of the level of customization and programmability of the implementation. Clearly, an ASIC implementation has fully customized datapaths and control logic, but it is not programmable. At the other end of the scale, a GP has little or no customized logic for the application at hand. For signal processing applications, a DSP is a more customized architecture than the general processor. An ASIP can be far more customized to an application or small set of applications but yet be programmable.

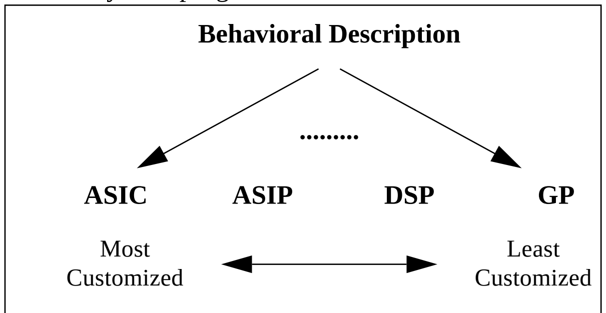


Figure 1. The Range of Implementation Styles

This paper explores the design space available between the general purpose processor and ASIC design styles mainly with the goal of power reduction. A behavioral description of a processor includes descriptions of all of its instructions. We could use an off-the-shelf version of the processor, or synthesize it using behavioral synthesis techniques. Alternately, we could determine a suitable subset of the instructions for the application at hand, and synthesize an ASIP that only implements that subset. This process, called *instruction subsetting*, typically reduces the area, critical path, and power dissipation of the implementation, while partially retaining its programmability.

Instruction subsetting is an aspect of hardware software codesign in that it provides for performance trade-offs between the software application to be executed and the underlying hardware architecture specifically designed to execute it.

Key to this approach is the use of behavioral intellectual property (IP), specifically that of a general or DSP processor. Given such behavioral IP, new techniques for creating of a range of ASIPs for custom low power system design can be developed. Not only will the power dissipation be reduced, but the ASIP will still be programmable (although with a more limited instruction set). Thus we can trade off the reduced power of an ASIC with the programmability of an ASIP. Unlike [Ing94], we will begin with a defined instruction set and simply remove unneeded instructions rather than synthesizing a new instruction set based specifically on the needs of the target application.

This paper describes the results of an experiment to characterize the design space and trade-offs available when using the instruction subsetting technique.

Approach

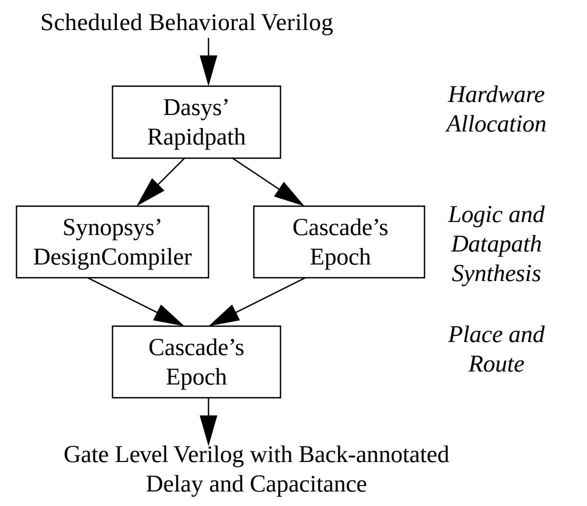
Research in computer-aided design techniques for mixed hardware/software systems has begun to focus on reducing the power consumption of the systems being produced. Attention has been focused on reducing transition counts in the logic hardware using behavioral synthesis techniques [Rag94][Meh96] reducing memory accesses [Cat97], code generation [Tiw94][Sri96], and shutting down unused parts of a system [Mon96]. Our goal is to examine power trade-offs that can be made in the exploration of the design space.

Consider the design of a 4-tap finite impulse response (FIR) filter. The most customized implementation of it is an ASIC that only executes this FIR filter. The least customized, as illustrated on the right of Figure 1, is with a general processor. For our experiments, the GP design is a partial implementation of the Motorola M68HC11 8-bit microcontroller [Mot91] that will execute 77 different instructions. Between those extremes, we have implemented three instruction subsets of the Motorola DSP56000 [Mot90] 24-bit digital signal processor (executing 36, 19, and 11 instructions). These designs can be thought of as a DSP and two increasingly customized ASIPs. A second version of the M68HC11 was also created (executing only 24 instructions). This corresponds to an ASIP version of the M68HC11. Each design and the environment used to synthesize each design is described below.

The smaller M68HC11 design implemented only the 24 instructions used in a hand-designed FIR program, while the larger design (77 instructions) implemented the instructions needed for an FFT program [Wil94]. Note that care must be taken in comparing the raw results of these designs with the other designs, since the datapath for these is 8-bits wide while the others have 24-bit datapaths.

The smallest DSP56000 design (11 instructions) corresponded to the instructions used in a hand-designed FIR program. The second design (19 instructions) implemented the instructions necessary for an FFT algorithm[Mot92]. The third design (36 instructions) implemented the instructions needed to implement a portion of the SPHINX-III speech recognition front-end [Wei97] as determined by compiling the original C code.

All six designs were implemented with the design flow shown in Figure 2. Scheduled behavioral level Verilog is input to DASYS' RapidPath [Das97], which allocates hardware for the design. The controller of the register-transfer level Verilog is passed through Synopsys' Design Compiler [Syn97] for logic synthesis, while the datapath portions are synthesized with Cascade Design Automation's Epoch [Cas97]. The designs were mapped to 0.5 micron CMOS technology. Epoch is also used to place and route the entire design. Power estimates were obtained from the gate level Verilog with timing and capacitance data back-annotated from the placed and routed design. An in-house simulation-based power estimation tool [Pur96] was used.



Scheduled Behavioral Verilog
Dasys' Rapidpath
Hardware Allocation
Synopsys' DesignCompiler
Cascade's Epoch
Logic and Datapath Synthesis
Cascade's Epoch
Place and Route
Gate Level Verilog with Back-annotated Delay and Capacitance

Figure 2. High-level design flow

Our FIR filter is a baseline design; the ASIC and the smallest DSP and GP ASIPs were designed specifically to execute this filter algorithm. Fuller implementations of the DSP and GP architectures, where we have implemented larger subsets of their instructions, allow us to measure the incremental changes in power dissipation when using larger ASIPs. Although several other applications were programmed on the processors, the FIR filter remains the only application comparable with the ASIC.

Results and Analysis

Table 1 presents the physical characteristics for each design used in our experiments. The design names appear in the first column of the table. The second column shows the size of the design, placed and routed in a 0.5 micron static CMOS process. The transistor counts presented in the third column are divided between logic transistors and memory. Like the 56K, all the DSP designs contain 9KB of memory, divided equally among an instruction memory and two data memories. The ASIC design contains only 6KB of memory, as it does not require an instruction memory.

The critical path of the design, shown in the fourth column, represents the minimum clock period for the design as determined by Cascade's TACTIC static timing analyzer. The fifth column, time per datum, shows the amount of time needed to complete one FIR iteration. The HC designs do not directly support signed multiplication, and therefore have significantly longer time per datum. The sixth and seventh columns show the number of instructions supported by each design and the percentage of the total instruction set this comprises, respectively.

Table 1: Physical Design Characteristics

Design	Chip Area (mm2)	# of Trans-istors	Critical Path (ns)	Time/ Datum (ns)	# of Instr.	% of Total Instr.
ASIC	17.76	174,692 + 6KB SRAM	38	152	N/A	N/A
DSP11	30.8	179,099 + 9KB SRAM	66	464.1	11	17.74%
DSP19	80.91	355,821 + 9KB SRAM	90	630	19	30.65%
DSP36	85	411,865 + 9KB SRAM	102	714	36	58.06%
HC24	1.6	16,905	26	6,732	24	7.74%
HC77	4.25	33,483	28	7,219	77	24.84%

While we would expect that the critical paths and transistor counts would increase at a steady rate as more instructions are added, we see a jump in the critical path for the DSP designs at the DSP19 ASIP. This is accounted for by the significantly larger amount of control logic introduced between the DSP11 and DSP19 than is introduced between the DSP19 and DSP36 designs as a result of the two additional addressing modes and additional move instructions required by the fft application. The majority of the changes made between the DSP19 and DSP36 designs are additional datapath items. This is also reflected in the transistor count. As expected, implementing more instructions in a design increases physical characteristics like design and critical path.

Table 2: Energy Consumption per Datum for fir4

Design	Energy/Datum (nJ) (fir4)	Voltage Scaled Energy/Datum (nJ) (fir4)
ASIC	6.11	1.33 (@ 1.4V)
DSP11	20.07	11.80 (@2.3V)
DSP19	34.30	29.88 (@ 2.8V)
DSP36	44.64	44.64 (@ 3V)
HC-Fir subset	62.36	58.27 (@2.9V)
HC-FFT	153.16	153.16 (@ 3V)

Table 2 shows the energy consumption for all our designs running the 4 tap FIR filter (fir4). Both normal and voltage scaled energy results are presented. From column two, we can see that the DSP36 design consumes 44.6 nJ per iteration, while the DSP19 design consumes only 34.3 nJ for the same amount of work. If the DSP19 design is voltage scaled to operate at the same speed as the DSP36 design, energy consumption drops to 29.8 nJ. Clearly, there is a trade-off to be made here. If we only want to run 4-tap FIR filter, there is a significant benefit to using the smallest implementation available.

Table 3: Energy Consumption for DSP Designs

Design	Energy per Datum (nJ) (fir4)	Scaled Energy per Datum (nJ) (fir4)	Energy per Datum (nJ) (fir4)	Scaled Energy per Datum (nJ) (fir4)	Energy per Datum (nJ) (fir4)	Scaled Energy per Datum (nJ) (fir4)
DSP11	79.24	46.58	N/A	N/A	N/A	N/A
DSP19	142.61	124.23	640.17	557.66	449.63	391.65
DSP36	205.23	205.23	618.54	618.54	482.98	482.98

Table 3 shows similar data for other applications running on the DSP ASIPs. The other applications tested were: a 64 tap FIR filter (fir64), a 64 point FFT (fft), and a 64 tap adaptive least mean square filter (lms). The N/As in the DSP11 row under the fft and lms columns are present because the ASIP did not contain all of the instructions needed to run the fft and lms programs as implemented. The program could of course be rewritten to run on the smaller ASIP, but would take longer to run and therefore consume more energy overall.

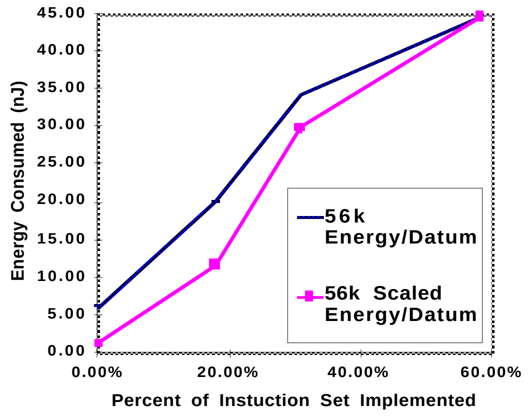


Figure 3. Energy and Voltage Scaled Energy of 56k Based ASIPs running fir4.

The cost of programmability, as determined by energy consumption, can be measured by looking at the energy consumed over the entire design space. The trade-off in energy consumption as compared to the

Conference submission: do not copy.

percentage of the instruction set implemented for the DSP designs running fir4 is shown in Figure 3. Both voltage scaled and unscaled measurements are presented. The point at 0% of the instruction represents the ASIC. Moving from the ASIC to the minimum FIR implementation more than triples the amount of energy consumed in this architecture. The slope then increases moving from the DSP11 to DSP19 designs, meaning the cost of adding this additional programmability is higher than that experienced by moving away from the customized design. This can be attributed to the additional datapath and control logic needed to support the additional addressing modes required by the fft, and accounts for the knee in the curves. Beyond this point, the curve increases at a much lower rate, as much of the complicated control logic and datapath elements are present in the design. The shape of the curves will of course be highly dependent on the particular instructions present in the design. By creating our ASIPs moving from simple to more complex DSP programs, we attempt to add instructions in the order of general application usefulness.

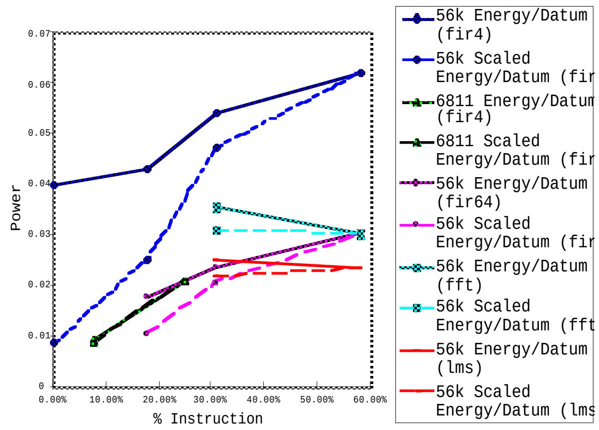


Figure 4. Power consumed per datum across all designs and applications

The rate of increase in power consumption will vary based on the types of instructions used in the programs. The energy results can be normalized by dividing by the amount of time needed to produce an output for the particular application, giving the amount of power consumed per datum. This information is presented in Figure 4. For the FIR implementations (fir4 and fir64) there is a substantial increase in power consumed per datum as the number of instructions implemented increases. This holds for both the 56K and 6811 based designs. The fft and lms applications show a different trend, with the power consumption per datum dropping when moving from the 19 to the 36 instruction ASIP. This trend appears to arise from the DSP36's better layout, as compared to the DSP19, which reduces the driven capacitance. Still, the trend is significantly different than that of the FIRs which are dominated by multiply-accumulate instructions, while the fft and lms programs contain large numbers of move and loop control instructions.

Beginning with the DSP36 ASIP, the power savings for the FIR applications increases an average 17% when scaling to the 19 instruction design. Moving to the DSP11 design yields an additional 22% gain in the non-voltage scaled case. For the 6811 based designs, a 58% improvement in energy consumption was seen moving to the smaller ASIP in the non-voltage scaled case. Voltage scaling increases the improvement in the fir4 program to 73% in the DSP designs and 60% in the HC designs. Again, we see a strong benefit from using the minimum instruction set as required by the application.

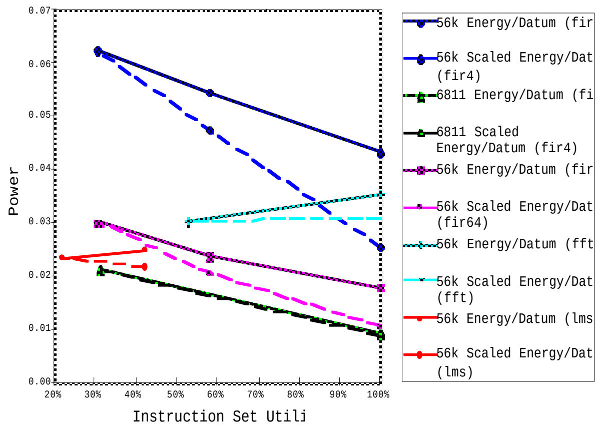


Figure 5. Power per datum compared to percentage of ASIP instructions utilized.

Power consumption seems to track linearly with the instruction set utilization of the ASIPs, higher utilizations tend to lower power consumption. Instruction set utilization is defined to be the number of instructions used in a program divided by the number of instructions in the ASIP. The fir4 application uses all 11 instructions in the DSP11 processor, and therefore has a utilization of 100%. Running the same program on the DSP36 design would have a utilization of 31%. This effect is plotted in Figure 5. For applications heavily dominated by calculations (fir4 and fir64), increasing utilization is accompanied by decreasing power consumption. As we move from simple applications to more complex ones, the slope of the power consumption line tends to increase, and reverses direction for the fft and lms applications in the unscaled measurements. This again appears to be due to the better layout achieved by the DSP36 design, and we would not expect this trend to continue if a larger ASIP was created. Computationally intense applications experience significant power savings if the ASIPs they are run on can be created in such a manner that the utilization of available instructions is high.

Table 4: Energy Consumption by Design Element

Design	Datapath Energy (nJ)	Control Energy (nJ)	Clock Energy (nJ)
ASIC	5.9	0.1	0.1
DSP11	16.1	1.7	2.3
DSP19	27.1	4.4	2.8
DSP36	37.1	4.4	3.1
HC24	37.5	15.1	9.8
HC77	89.3	51.4	12.4

Looking at the breakdown of energy consumption from datapath, control, and clock elements in each design shows why the cost of programmability increases as the designs become more flexible. This data, presented in Table 4 for the fir4 application, shows that the majority of the energy consumed in each design is from the datapath functional units. As the ASIPs become more general purpose in nature, the control and clock energy become increasingly more significant, growing at a rate of 70% from the HC24 design to the HC77 design when running the fir4 program. A 60% increase occurred moving from the DSP11 to DSP19 design, but stayed at about the same level moving to the DSP36 design for the same application. Still, control logic never makes up more than 13% of the DSP design space and 35% of the HC design space. The datapath energy consumption increased by up to 40% in the same design space, largely due to increased capacitance and spurious transitioning.

Summary

We found that instruction subsetting can be used as an effective means of power reduction. However, its effectiveness is dependent on the utilization of instructions available in the ASIP. Analyzing our five placed and routed designs, we determined the logic required to implement additional instructions is prohibitively wasteful if the instructions are not used. This effect is more prominent when executing computationally intense algorithms (such a FIRs), than algorithms dominated more by control and data movements (such as FFTs). For the FIR applications, larger instruction set designs (with correspondingly lower instruction set utilizations) saw an average of almost 60% increases in the power per datum and as much as a 400% increase in energy consumption over the entire design space. These results suggest that instruction set subsetting can be a valuable technique for architecture exploration during hardware/software codesign.

Acknowledgments

This work was supported by the Defense Advanced Research Projects Agency under Order No. A564, the National Science Foundation under Grant No. MIP9408457, and the Semiconductor Research Corporation under Task IDs 068-063 and 068-064. The United States Government has certain rights to this material.

References

[Cas97] Cascade Design Automation Corporation, Epoch User's Manual, February, 1997.

[Cat97] F. Catthoor, L. Nachtergaele and S. Wuytack, "Optimizing Data Transfers and Memory for Low Power," *ASIC & EDA Magazine*, Spring 1997.

[Das97] Dasys, Inc., *RapidPackage User's Manual*, October 1997.

[Ing94] H. Ing-Jer and A.M. Despain, "Generating Instruction Sets and Microarchitectures from Applications," *Proc. of ICCAD 94*, p391-6.

[Meh96] R. Mehra and J. Rabaey, "Exploiting Regularity for Low-Power Design," *Proc. of ICCAD '96*.

[Mon96] J. Monteiro, S. Devadas, P. Ashar and A. Mauskar, "Scheduling Techniques to Enable Power Management," *Proc. of DAC '96*, pp. 349-356, June 1996.

[Mot90] Motorola, Inc. *DSP56000/56001 Digital Signal Processor User's Manual*, 1990.

[Mot91] Motorola, Inc. *M68HC11 Reference Manual*, 1991.

[Mot92] Motorola, Inc. "Archive containing DSP56000-related files," *http://www.mot.com/pub/SPS/DSP/software/dr_bub/56000.zip*, January 1992.

[Pur96] D.J. Pursley, "A gate level simulator for power consumption analysis," M.S. Thesis, Carnegie Mellon University, May 1996.

[Rag94] A. Raghunathan and N.K. Jha, "Behavioral Synthesis for Low Power," *Proc. of ICCD '94*, pp. 318-322, October 1994.

[Sri96] M. Srivastava and M. Potkonjak, "Power Optimization in Programmable Processors and ASIC Implementations of Linear Systems: Transformation-based Approach," *Proc. of DAC '96*, pp. 434-348, June 1996.

[Syn97] Synopsys, Inc., *DesignWare User Guide*, August 1997.

[Tiw94] V. Tiwari, S. Malik and A. Wolfe, "Compilation Techniques for Low Energy: An Overview," *Proc. of 1994 Symposium on Low-Power Electronics*, October 1994.

[Wei97] N. Weinberg, private communication.

[Wil94] R. Williams, "fft.c11 - Fast Fourier Transform for the MC68HC11," *http://www.mcu.motsps. com/freeweb/pub/mcu11/ffthc11.asm*, March, 1994.

Conference submission: do not copy.