Software Power Estimation and Optimization for High Performance, 32-bit Embedded Processors[*]

Jeffry T. Russell Margarida F. Jacome
Department of Electrical and Computer Engineering, University of Texas at Austin
jeffry@equator.ece.utexas.edu jacome@ece.utexas.edu

Abstract

A software energy estimation model is presented for a family of high performance, integrated, 32-bit embedded RISC processors. This model is significantly less complex than previous models, and yet is demonstrated to accurately predict energy consumption to within 8% with 99% confidence based on physical measurements. Factors such as operating frequency, source/destination registers, and operand values are explored. In view of this model, previously proposed optimizations are evaluated for potential energy savings. We conclude that, for the class of processors under discussion, a good optimizing compiler that minimizes execution time will simultaneously minimize energy consumption.

1. Introduction

Power consumption is becoming increasingly important in the design of embedded applications. It places requirements on heat dissipation, which directly affect unit cost. Furthermore, in portable systems, the minimization of energy consumption, the time integral of power, is critical to extending battery lifetime.

In this paper we develop a software energy estimation model for two processors of a family of high performance, 32-bit, RISC embedded processors and then study the effectiveness of previously proposed power saving optimization techniques. Examples of embedded applications based on a single processor of this type include laser printers, network interfaces, X-terminals, digital storage oscilloscopes, Global Positioning Receivers, industrial controllers, and medical instrumentation.

A typical family is the i960 Embedded Processor architecture, consisting of about twenty implementations that vary widely in cost (10s to 100s of dollars) and performance (up to 100s of MIPs). The processors are differentiated by speed, packaging, and level of integration, which may include such system support features as caches, pipelined bus interfaces, interrupt controllers, DRAM memory controllers, or virtual memory managers. We considered two implementations of the architecture, the 80960JF and 80960HD [16][18]. The HD enhances performance for 3-4 times the cost with a clock doubled core and parallel execution of math and memory operations [22].

A mature processor family such as this includes a number of software libraries for operating systems, device drivers, communication protocols, and high-level routines to support math, graphics, and user applications. Development tools such as optimizing compilers, assemblers, debuggers, emulators, and simulators enable timely development of complex applications using any processor in the family.

In order to study the effectiveness of previously proposed software energy optimization techniques for this important class of embedded processors, we started by working on the derivation of an energy estimation model based on the characterization of average power for individual instructions. Accordingly, our initial experiments examined the variation of power consumption across the assembly instructions of the two target processors. Statistical analysis techniques were used to infer the significance of factors such as operating frequency, source and destination registers, and operand values.

The interesting conclusion, supported by our extensive physical measurements, was that for both the JF and HD, the variations of power consumption across the assembly instructions proved to be of no *statistical significance*. In short, there is actually no need to consider the individual assembly instructions to accurately predict power and energy consumption. Specifically, it will be shown that power consumption can be predicted to within 8% accuracy with 99% confidence, simply by using the processor average power consumption per cycle multiplied by the software execution time (in cycles). The important implication of this result, for this class of processors, is that minimizing execution time means also minimizing energy consumption. The optimizing compiler that supports this processor family, while minimizing software execution time, is also simultaneously generating minimum energy code.

The remainder of this paper is as follows. In Section 2 we discuss related work. In Section 3 we outline the experimental method, the proposed energy model, and the statistical test to support the model. Some conclusions are given regarding applicability of these results in Section 4.

[*] Sponsored by NSF, under CAREER Grant No. MIP-9624321.

2. Background

A good survey of low power design and estimation techniques for hardware is given in [1] and [2]. Many power analysis techniques use software as a stimulus for a hardware simulation [10] [11] [12] [13] [14] [15]. Tiwari, et al. [4] used physical measurements for software power analysis. Several software optimizations for power and/or energy were proposed in [5][6][7][8][9] .

The idea of developing an instruction level power model for individual processors was first proposed in [4]. Power is modeled as a base cost for each instruction plus a circuit state overhead that depends on neighboring instructions. A method is proposed to empirically determine these costs in which small program segments with repeated instructions were run in an infinite loop so that all accesses occurred from the cache. An ammeter was used to measure the average current over a 100ms time window, which is proportional to the power consumed. We have used this experimental method as the starting point for our investigation.

3. Energy Model

The energy, E, consumed by a processor running a program is

$$E = \int_{t_o}^{t_o+T} P(t)dt\,,$$

where T is the software execution time and P(t) is the instantaneous power. Average power, P_{ave}, is defined as

$$P_{ave} = \frac{1}{T}\int_{t_o}^{t_o+T} P(t)dt$$

and thus we can write

$$E = T \cdot P_{ave}$$

Accordingly, to measure energy consumption, a measurement for average power and total execution time is needed.

However, there is a more convenient form in which to pose the software energy estimation problem. Total time, T, is related to N, the total number of execution cycles, and τ, the clock period, by T=$N\tau$. The average power, P_{ave}, is thus the summation of the average power during each clock period, $P_{\tau,ave}(i)$, where i=1…N, as given by

$$P_{ave} = \frac{1}{N}\sum_{i=1}^{N} P_{\tau,ave}(i)$$

The value $P_{\tau,ave}(i)$ is the average power consumed during one cycle of the instruction and motivates our study of the variation in power consumption across the instruction set. (Note that some instructions take multiple cycles to execute.) We observe that each $P_{\tau,ave}(i)$ is dependent on the present and past states of the processor,

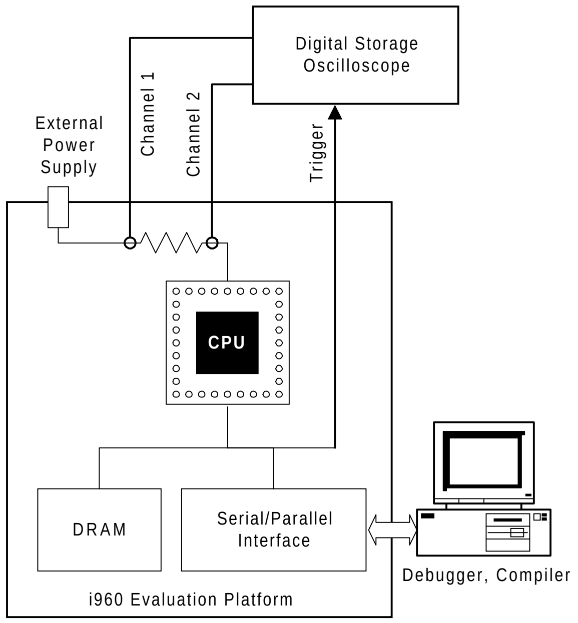


Figure 1. Experimental platform.

and thus measuring an instruction in isolation is a biased estimate of the power consumed by an instruction in a typical application.

Execution time depends significantly on runtime conditions such as cache misses, data dependent pipeline stalls, operand dependent instruction execution times, and external DRAM refresh cycles. There are several accurate ways to measure execution time: cycle accurate simulation, instrumentation of the actual system, or software based timers inside the processor. For the i960 processor family, Intel actually offers a remote, web-based evaluation facility where one can run code on a variety of embedded processors and obtain execution time estimates. For our experiments, we directly measured execution time with an oscilloscope.

In the following sections, we describe the experiments used in our investigation, propose a model for power estimation, and present results that demonstrate the accuracy of the model. We conclude with a discussion on the effectiveness of potential software optimizations.

3.1 Experimental Method

We initially expected to develop a model similar to Tiwari, et al [4] that assigned a power cost to each instruction. As in [4], we measured power consumption of several instances of individual instructions executing in an infinite loop. However, our experimental measurement technique had a number of important differences with respect to [4]. We also felt it was important to perform a careful statistical analysis of the measured results.

To measure the characteristics of each instruction, the following assembly pseudo code was used.

```
main(random seed)
{     init. system: cache, system regs
loop: randomize input values
      load (start address)
      100 x ( assembly instruction )
      load (end address)
      branch loop
}
```

The processor was configured so that the program executed from cache, yet external memory cycles could be probed. The `start address` and `end address` were used to set up a time window so as to ensure that power was measured only during the instruction under examination. Prior to each loop iteration, the operand values were randomized.

Figure 1 shows our measurement configuration, using the i960 Cyclone Evaluation Platform. It provides a common system with interchangeable processor modules. The tested processors are binary compatible, and every effort was made to control environment conditions during measurements. Hence, the results for the two processors are directly comparable.

To measure power, a precision resistor was placed in series with the power connection to the processor. This introduced stray inductance in the power delivery circuit, which in turn induces a noise voltage during switching surge currents. To counteract this effect, decoupling capacitors were added to the wires. We experimented with different capacitor values to reduce the high frequency component of the noise.

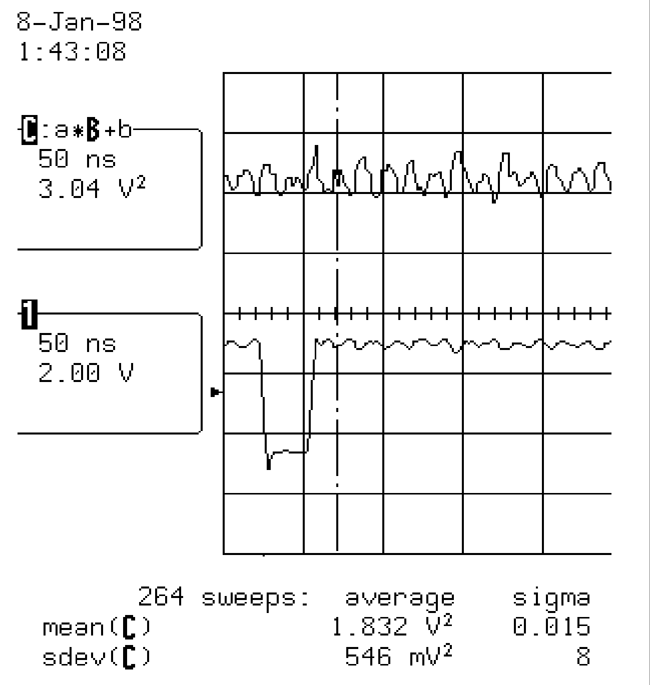
A digitizing oscilloscope, a LeCroy LC534, with a sample rate of 500 Mbps was used to measure the instantaneous power. The oscilloscope provides for math operations on input values and gathers statistics on its measurements. We measured instantaneous power at the bandwidth of the oscilloscope, and calculated average power for each iteration of the loop of instructions. We accurately measured the power across only the body of the loop, which consists of 100 instances of the instruction.

Referring to Figure 1, Channel 1 is used to probe the power supply voltage, V_1. To measure the instantaneous power, an absolute measurement of the voltage at the processor, V_2, was multiplied by the differential voltage across the resistor (V_1-V_2), and scaled according to the resistance, R, as given by

$$P(t) = I(t)V(t) = \frac{(V_1(t) - V_2(t))}{R} V_2(t) \qquad (*)$$

The average power, P_{ave}, was measured for several iterations of the program loop. Based on these independent trials we estimate its mean, $\overline{P}_{ave}$, and corresponding sample deviation, *s*.

The oscilloscope measures voltages to within ±3mV, which we use to model measurement uncertainty of one standard deviation for a single measurement. An analysis of error propagation [16] for power measurements based on (*) gives measurement uncertainties of ±42mW (≈2.5%) and ±70mW (≈2.5%) for the JF and HD processors, respectively. We have assumed that V_1, V_2 are independent and approximately equal, making the above estimates conservative.



```
      264 sweeps:  average    sigma
 mean(C)           1.832 V²   0.015
 sdev(C)            546 mV²       8
```

Figure 2. Screen plot of oscilloscope.

Power fluctuates significantly during the clock cycle, but has a stable average value. This is demonstrated on the JF processor for which the instantaneous power has typical deviations of 600 mW from P_{ave} and of *s*=20 mW from $\overline{P}_{ave}$. An example screen plot of the oscilloscope in Figure 2 shows the power on the top (Channel C) and the address strobe line on the bottom (Channel 1). Notice the high frequency power deviations; the vertical scale on Channel C is 3 w/div. The horizontal division is 50 ns, and the clock period is 30 ns.

3.2 Measured Power and Energy

We applied the above measurement procedures to all "normal" instructions that would be output by a compiler. This excludes special instructions for manipulating the cache, modifying processor controls, and debug/trace instructions. Both processors offer a variety of instructions to support high-level languages, such as compare and decrement (`cmpdeci`), conditional add (`addc`) or branch if greater (`bg`), which were included in the measurements.

	JF Processor			HD Processor		
Instr.	$\overline{P}_{ave}$	s	Cyc.	$\overline{P}_{ave}$	s	Cyc.
add	1.77	0.03	1	2.87	0.03	0.5
sub	1.74	0.02	1	2.85	0.03	0.5
mul	1.72	0.03	5	2.80	0.02	2.5
div	1.62	0.02	35	2.43	0.02	17.5
mod	1.62	0.02	35	2.49	0.02	17.5
rotate	1.79	0.03	1	2.85	0.03	0.5
and	1.75	0.03	1	2.86	0.04	0.5
xor	1.76	0.02	1	2.87	0.03	0.5
setbit	1.75	0.03	1	2.87	0.03	0.5
bswap	1.62	0.02	10	2.81	0.04	0.5
mov	1.67	0.02	1	2.83	0.03	0.5
Extern. ld	1.71	0.01	9	3.18	0.02	6.0
Intern. ld	1.75	0.02	2	3.40	0.03	0.5
Extern. st	1.97	0.05	5	3.23	0.02	5.0
cmp	1.63	0.02	3	2.76	0.03	0.5
cmpdec	1.62	0.02	3	2.78	0.03	0.5
b	1.40	0.01	2	2.57	0.02	1.0
call/ret	1.56	0.01	14	2.95	0.02	5.0

Table 1. Subset of power measurements.

Prior to measuring power and execution time for the instruction set, we examined some variations in the assembly instructions that might influence power consumption, specifically:

- We considered the effect of different source and destination registers on the dispersion of measured power for the move (`mov`) instruction. There are 16 local and 16 global registers, which is a total of 992 permutations of source and destination. We tested a random sample of size 32. The coefficient of variation (CV) for the JF and HD processor was 3.0% and 1.6%, respectively. These were judged to be insignificant.
- We examined the effect of operand values using the multiply (`mul`) instruction. We ran 22 trials using worst case operand values based on the number of 0s and 1s. The CV for the JF and HD processor was 5.4% and 2.2%, respectively. The CV for the JF processor was judged to be significant, which led us to include the randomizing element in the test program (for both processors).
- We examined the effect of condition codes and conditionally executed instructions. We tested all variations of the conditional add instruction with a random mix of true and false conditions. Each trial was repeated with five different operand values, as the randomization for each loop iteration was not implemented for this experiment. The CV for the JF and HD processor was 1.1% and 0.7%, respectively. These were judged to be insignificant.

t-test parameters							
Proc.	Ave. Error	Std Dev.	Est. P	Error Limit	t value	α.	Crit. pt
JF	-0.12	0.03	1.69	8.0%	-14.44	0.01	5.84
HD	-0.15	0.06	2.83	8.0%	-10.20	0.01	5.84

Table 2.

After determining which features of an assembly instruction were significant, we measured power and execution time for each of the 105 normal instructions. All these experiments were conducted at an operating frequency of 33 MHz. These experiments were conducted with randomized operand values. A small subset of the measurements is shown in Table 1. The average power and exact execution time is reported. Note that both an internal (cache hit) and external (cache miss) load instruction (`ld`) are listed. A store instruction (`st`) can only write through the cache. As would be expected, several instructions executed faster on the higher performance HD.

The mean of $\overline{P}_{ave}$ across all instructions is 1.68 watts for the JF processor and 2.85 watts for the HD processor. However, it is inappropriate to use this *descriptive* statistic to infer an average value for power consumption. This is primarily due to the fact that the tested instructions do not represent a random sample of all occurences of instructions. A more appropriate representation of a constant parameter for power is the median average. For the JF processor, the median average is 1.69 watts with a 0.07 watt semi-interquartile and for the HD processor it is 2.83 watts with a 0.06 watt semi-interquartile. The semi-interquartile range is a measure of dispersion that reports the approximate range that contains 50% of the values. Using the median average, we propose a hypothesis concerning the use of a constant value for power estimation, which we can test with statistical inference.

3.3 Proposed Model

We propose a simple model in which power is modeled with a constant parameter. We use the method of statistical inference to support the constant parameter model. The experiment hypothesis is that the power consumption can be modeled with a constant parameter to accurately estimate power with less than 8% error. We define the error estimate as $\partial_{est} = P_{ave} - P_{est}$, where P_{est} is the power estimator, our constant parameter.

		JF Processor				HD Processor			
		Measured		Derived		Measured		Derived	
Program	Description	Time (μS)	P_{ave}	Energy (j)	Rel. Error	Time (μS)	P_{ave}	Energy (j)	Rel. Error
`psdes`	Psuedo DES encryption of 20 32-bit integers.	53.3±8.34	1.848±.011	98	-8.5%	23.8±.19	3.03±.02	72	-6.5%
`heap`	Heap sort of 20 random, 32-bit integers.	62.6±.57	1.791±.010	112	-5.6%	26.4±.23	3.04±.02	80	-6.7%
`fft`	FFT of 32 floating point, positive values < 1000.	1403±10	1.779±.007	2495	-5.0%	492.5±2.1	2.90±.10	1429	-2.4%
`moment`	Moment statistics of 32 floating point values.	879±.1	1.805±.055	1587	-6.4%	273.9±.02	2.96±.01	810	-4.2%

Table 3. Sample programs used in the hypothesis testing.

We define the statistical hypothesis as

$$H_0 : \partial_{est} \geq 8\%$$

$$H_1 : \partial_{est} < 8\%$$

A t-statistic is used since the sample size is small and the variance is unknown. We chose a sample size of programs, shown in Table 2, based on resource limits. The specified level of significance is α=0.01 for a two-tailed test. The results of the experiments are shown in Table 3.

Based on this statistical test, the null hypothesis is rejected. We therefore conclude, with 99% certainty, that our constant parameter model predicts power consumption with less than 8% error.

3.4 Power Optimizations

Lee et al. [7] demonstrated the relevance of a number of fine grain optimizations for a custom DSP processor, based on the Tiwari model [4]. This and other studies [5][8] report power saving optimization techniques based on instruction reordering, instruction packing or recoding, operand reordering, register allocation, and memory assignment. These techniques also correspond to performance improvements in most cases.

A good optimizing compiler, which is available for the family of embedded processors under consideration, addresses these performance issues so as to minimize execution time. Thus energy consumption is also minimized.

3.5 Power versus Frequency

The behavior of power consumption versus frequency for a few representative instructions of the processors is shown in Figure 3. The expected linear relationship between power and frequency [3] is apparent.

We conducted experiments on a random sample of 10 different instructions at four different frequencies. A least squares regression analysis was conducted to fit a line to the power-frequency relationship for each instruction. We estimate the power-frequency (P-F) ratio as the mean of the slope of the fitted lines. The JF processor P-F ratio is 0.915 with a standard deviation of 0.306. The HD processor P-F ratio is 0.935 with a standard deviation of 0.343. The CV for the JF and HD, respectively, is 37% and 33%.

The large dispersion of the P-F parameter, which is due to the mix of sampled instructions, precludes its usefulness as an accurate estimator for small changes in frequency. However, for a given program, the P-F ratio will have a small deviation because the instruction mix is constant. The power-frequency relationship indicates that the designer can achieve power/energy savings by increasing processor utilization in order to be able to reduce the clock frequency.

4. Conclusion

We have presented a model for estimating power and energy using a constant parameter for power consumption. This model is accurate to within 8% with 99% confidence. The model features *exceedingly simple* calculations for estimating energy: only the execution time estimate is needed, multiplied by the constant parameter for average power. The accuracy of the model was demonstrated for two implementations of the i960 Embedded Processor family. The JF and HD processors represent quite distinct points in the wide range of cost/performance offered by this important family. Based on the common architecture and development tools, we expect this power model to be widely applicable to this and similar families.

We further observe that power consumption is linearly dependent on frequency.

For the general class of embedded systems previously discussed, we conclude that the designer should work towards minimizing software execution time and, as a consequence, power/energy consumption will be minimized.

For external memory and I/O access instructions, it was found that both power consumption and execution

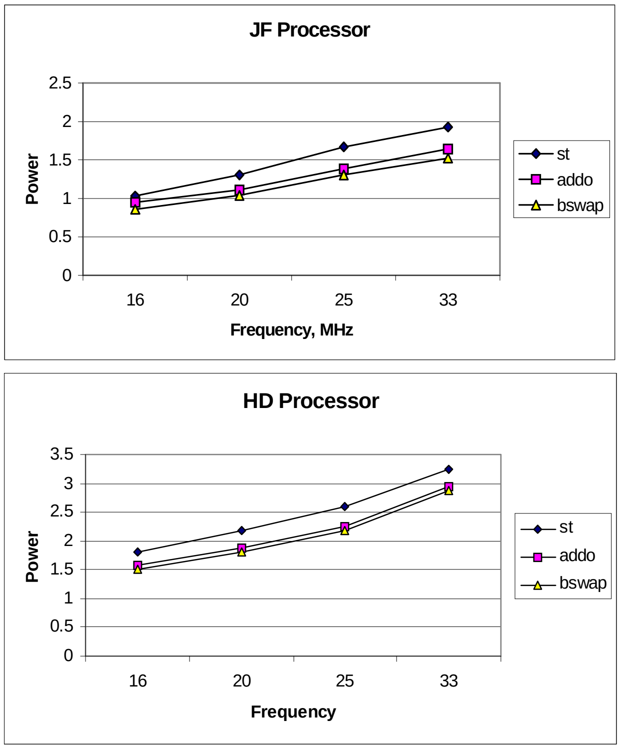


Figure 3. Power versus frequency.

time increased significantly. These observations clearly indicate that system level architectural issues, including the partitioning and hierarchical organization of memory, coprocessor interfaces, and bandwidth balancing on busses, are of major importance in terms of overall system performance and power consumption. The current focus of our research is thus on the development of mechanisms to support design space exploration at the architectural level for embedded systems.

References

[1] Jan M. Rabaey, Massoud Pedram, editors, *Low Power Design Methodologies*, 1996, Kluwer Academic Publishing, Boston

[2] D. Srinivas, M. Sharad, Tutorial: A survey of optimization techniques targeting low power, *32nd Design Auto. Conf. Proc*, June 1995

[3] N. Weste, K. Eshraghian, *Principles of CMOS VLSI Design: A Systems Perspective*, 2nd Ed., Addison-Wesley, 1993

[4] V. Tiwari, S. Malik, A. Wolfe, Power analysis of embedded software: a first step towards software power minimization, *IEEE Trans. on VLSI Systems*, Dec. 1994

[5] V. Tiwari, S. Maik, A. Wolfe, Compilation techniques for low energy: an overview, Proc. Symp. Low Power Electronics, 1994

[6] V. Tiwari, S. Malik, A. Wolfe, M. Lee, Instruction level power analysis and optimization of software, *Journal of VLSI Signal Processing*, pp. 1-18, 1996, Kluwer Academic Publishing, Boston.

[7] M. Lee, V. Tiwari, S. Malik, M. Fujita, Power analysis and minimization techniques for embedded DSP software, *IEEE Trans on VLSI Systems*, Mar. 1997

[8] H. Mehta, R. Owens, M. Irwin, R. Chen, D. Ghosh, Techniques for low energy software, *Proc. Int. Symp. Low Power Electronics and Design*, Monterey, Aug. 1997

[9] C. Su, C. Tsui, A. Despain, Low power architecture design and compilation techniques for high-performance processors, Proc. Symp. Low Power Electronics, Mar. 1994

[10] T. Sato, M. Nagamatsu, H. Tago, Power and performance simulators: ESP and its applications for 100MIPS/W RISC design, *Proc. Symp. Low Power Electronics*, San Diego, Oct. 1994

[11] R. Ong, R. Yan, Power-conscious software design- A framework for modeling software on hardware, *Proc. Symp. Low Power Electronics*, San Diego, San Jose, Oct. 1995

[12] T. Sato, Y. Ootaguro, M. Nagamatsu, H. Tago, Evaluation of architecture-level power estimation for CMOS RISC processors", *Proc. Symp. Low Power Electronics*, San Diego, San Jose, Oct. 1995

[13] P. Landman, J. Rabaey, Block-box capacitance models for architectural power analysis, *Proc. Int. Workshop Low Power Design*, Napa, April 1994

[14] H. Mehta, R. Owns, M. Irwin, Instruction level power profiling, *Int. Conf. on Acoustics, Speech, and Signal Processing*, 1996

[15] J. Montereiro, S. Devadas, Techniques for the power estimation of sequential logic circuits under user-specified input sequences and programs, *Proc. Int. Workshop on Low Power Design*, April 1995

[16] P. Bevington, K. Robinson, *Data Reduction and Error Analysis for the Physical Sciences*, 2nd Ed., WCB/McGraw-Hill, 1992

[17] Intel Corporation, i960® Hx Microprocessor User's Manual, Order Number 272484-001, Nov. 1995

[18] Intel Corporation, i960® Jx Microprocessor User's Manual, Order Number 272483-001, Sep. 1994

[19] Intel Corporation, i960® Processor Compiler User's Guide, Order Number 651230-002, Jan. 1997

[20] W. Press, S. Teulolshy, W. Vetterling, B. Flannery, *Numerical Recipes in C*, 2nd Ed., Cambridge University Press, 1992

[21] Roger E. Kirk, *Statistics: an Introduction*, 3rd Ed., Holt, Rinehart, and Winston, Inc, 1990.

[22] Intel Corporation, i960 Processor Home Page, http://www.intel.com/design/i960