Low Power through Software Optimisation

SOCRATES'04
Joan Oliver
ETSE-UAB

Power analysis of embedded software

- Until now
  - Power analysis techniques based on circuit-level or architectural-level
- Instruction-level power analysis model
  - For off the shelf microprocessors/microcontrollers
  - For embedded cores or IPs
- Problem:
  - Not available power consumption information
- So: development of a methodology and application to
  - 486DX2-S with 4MB computer board for mobile applications – Tiwari et alt.
  - To the actual Motorola 68HC908GP32 microcontroller
- Minimisation techniques in DSP processor
- Low power in Intel® 855GM Chipset

Experimental method (I)

- Microprocessor power consumption traditional method based on power consumption analysis of the unitary microprocessor modules → Difficult to establish general model because power consumption varies from program to program.
- Hypothesis: *By measuring the current drawn by the processor as it repeatedly executes certain instructions or certain short instructions sequences, it is possible to obtain most of the information that is needed to evaluate the power cost of a program for that processor.*
- Based on:
  - Complexity hidden behind a simple interface: the instruction set.
  - Individual instruction power analysis: specific circuit activity per instruction.
  - To take into account: inter-instruction effects.
  - Valid for standard microprocessor and embedded cores.

Experimental method (II)

- Power versus energy
  - Average power consumed by a microprocessor: $P = I \cdot V_{CC}$.
  - Energy consumed by a program: $E = P \cdot T$
  - Execution time: $T = \tau \cdot N$.
- Current measurement
  - *Though results are specific for every processor and board, the methodology of the model is widely applicable.*
    - The current is measured through a standard off the shelf, dual slope, integrating ammeter.
    - Execution time of the program is measured through a specific state detection by a logic analyser.
    - A program writen with several instances of the instruction sequence executing in a loop, has a periodic current waveform which yields a steady reading in the ammeter.
    - In the setup: Vcc=3.3V, τ = 25ns ($f_{internal}$=40MHz) →
      - Energy cost of a program that takes N cycles with an average current of I amps is $E = I \cdot VCC \cdot N \cdot \tau = 8.25 \cdot 10^{-8} \cdot I \cdot N$ J

Instruction level modeling (I)

- Development of a instruction level model
  - Each instruction has assigned a *base energy cost*.
  - Base energy cost variation due to different operand and address values can be quantified.
  - Energy cost of a program based on the sum of base energy costs.
  - To be considered: circuit state effects and resource constraints (can lead to stalls and cache misses).
- Base energy cost per instruction
  - It is determined by constructing a loop with several instances of the same construction and measuring the average current being drawn.
  - The total energy is is this average current multiplied by the number of cycles taken by each instance.
  - Besides 486DX2 executes more than one instruction at a given time (including pipelining) the concept of base energy cost per instruction remains unchanged.
  - When instructions take multiple cycles (in a given pipelining) also the base energy cost is just the average current measured multiplied by the number of cycles taken by the instruction in that stage.

Instruction level modeling (II)

- CPU base costs for some 486DX2 instructions.
  - Overall base energy cost per instruction: E_B=Column3·Column4·V_{CC}·τ.
  - Variations in repeated run experiments: ±1mA over average currents.

Number	Instruction	Base Cost (mA)	Cycles
1	NOP	275.7	1
2	MOV DX,BX	302.4	1
3	MOV DX,[BX]	428.3	1
4	MOV DX,[BX][DI]	409.0	2
5	MOV [BX],DX	521.7	1
6	MOV [BX][DI],DX	451.7	2
7	ADD DX,BX	313.6	1
8	ADD DX,[BX]	400.1	2
9	ADD [BX],DX	415.7	3
10	SAL BX,1	300.8	3
11	SAL BX,CL	306.5	3
12	LEA DX,[BX]	364.4	1
13	LEA DX,[BX][DI]	345.2	2
14	JMP label	373.0	3
15	JZ label	375.7	3
16	JZ label	355.9	1
17	CMP BX,DX	298.2	1
18	CMP [BX],DX	388.0	2

Instruction level modeling (III)

- Measurement conditions:
  - The loop size should be large enough to in order to obtain a converged value. It minimises the impact of the branch conditions at the end of the loop.
  - But it has not to be to much large in order to avoid caches misses.
  - System effects like multiple time-sharing and interrupts are indesirable.
- Variations in base costs:
  - Table shows that instructions with differing functionalities and different addressing modes can have very different costs. It is expected since different functional blocks are being affected in different ways by these instructions.
  - The same family of instructions shows different base costs depending on the value of the operands. For example, MOV instructions presents less base cost as number of 1's in data increases.

data	0	0F	0FF	0FFF	0FFFF
No. of 1's	0	4	8	12	16
Base Cost	309.5	305.2	300.1	294.2	288.5

Instruction level modeling (IV)

- So:
  - As seen by last table, variation in immediate operand values are significant.
  - Use of different registers does not result in significant base cost differences.
  - Range of variation shown by the ADD instruction is small: < 5%.
  - In instructions involving memory operands, the base costs variations depends upon the address of the operand, and depends upon the number of 1's in the operand address.
- Inter-instruction effects:
  - When sequence of instructions are considered, comes into play certain inter-instruction effects.
    - Circuit state effect. When a pair of different instructions is considered, the constext is one of greater change. A *circuit state overhead* is obtained with a cost always greater than the base cost of the pair. As an exemple, the measured cost of the sequence of the next table is 332.8mA (avg. current over 10 cycles).

      Using the base costs it should be shown a base cost of 326.8mA.

Instruction level modeling (V)

Number	Instruction	Base Cost	Cycles
1	MOV CX,1	309.6	1
2	ADD AX,BX	313.6	1
3	ADD DX,8[BX]	400.2	2
4	SAL AX,1	308.3	3
5	SAL BX,CL	306.5	3

- Experiments revealed that the circuit state overhead has a limited range between 5.0mA and 15.0mA.
- The overall impact of switching that occurs on address and data lines is small.
- Data reads/writes on cache has an impact less than 5%.

- Effect of resource constraints
  - Pipeline stalls and write buffer stalls could be considered as inter-instruction effect. Exemple: a sequence of 120 MOV DX, [BX] instruction takes 164 cycles due to prefetch buffer stalls.
- Effect of cache misses
  - Cache misses leads to extra cycles to be consumed. It has been found a cost penalty of 216mA per cache miss.

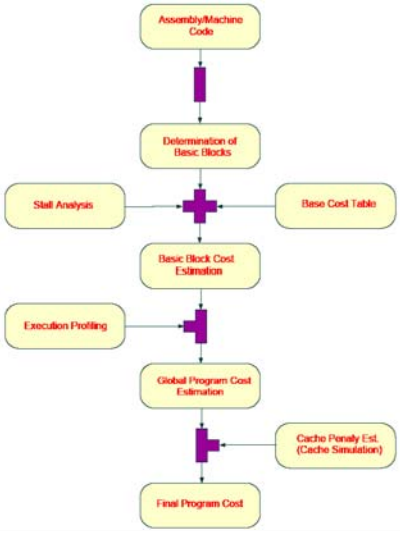
Estimation process

- Based on the computation of the current per instruction the cost consumption of a program can be obtained.
- Considerations have to be taken on:
  - To put the program in an infinite loop. The unconditional jump cost has to be added.
  - Computation of the times each block is executed.
  - The theoretical average current plus the state overhead costs gives an average current of 384mA, against the 385mA measured.

Program	Base Cost(*mA*)	Cycles
; Block B1		
main:		
mov bp,sp	285.0	1
sub sp,4	309.0	1
mov dx,0	309.8	1
mov word ptr -4[bp],0	404.8	2
;Block B2		
L2:		
mov si,word ptr -4[bp]	433.4	1
add si,si	309.0	1
add si,si	309.0	1
mov bx,dx	285.0	1
mov cx,word ptr _a[si]	433.4	1
add bx,cx	309.0	1
mov si,word ptr _b[si]	433.4	1
add bx,si	309.0	1
mov dx,bx	285.0	1
mov di,word ptr -4[bp]	433.4	1
inc di, 1	297.0	1
mov word ptr -4[bp],di	560.1	1
cmp di,4	313.1	1
jl L2	405.7(356.9)	3(1)
;Block B3		
L1:		
mov word ptr _sum,dx	521.7	1
mov sp,bp	285.0	1
jmp main	403.8	3

Estimation process flow

- Figure shows the overall flow
  - First, machine (or assembler) program code is split up into basic blocks.
  - Adding up the base instructions costs of the instructions the base cost of the block is determined.
  - Energy overheads due to pipeline, stalls, writes and other costs of the block are added up to the base block cost.
  - The number of times the block is executed has to be determined.
  - Circuit state overhead (cost of a pair of instructions, always greater than the sum plus the difference of bloth instructions) is added to the overall sum at this stage.
  - In the end, the estimated cache penalty is added to get the final estimate.



Software power optimisation

- Energy savings are possible through software optimisation.
  - Instruction reordering can imply power consumption reduction.
  - The instruction set chosen has influence on energy consumption. For exemple, instructions with memory operands have very high average current in front of instructions with register operands.
    - Instructions using only register operands cost about 300mA
    - Memory reads cost upwards 430mA.
    - Memory writes cost upwards 530mA.
  - Lesser (base clock) cycle instructions are energy saving instructions. For example, ADD DX, [BX] takes two cycles, while ADD DX, BX takes just one.
  - Potential pipeline stalls, misaligned accesses, and cache misses add to the running time.
  - Though reductions in number of memory operands can be achieved by adopting suitable code generation policies, the best way to save memory operands is through better use of registers.

Software power optimisation: example

- Results of energy optimisation of sort and circle algorithms

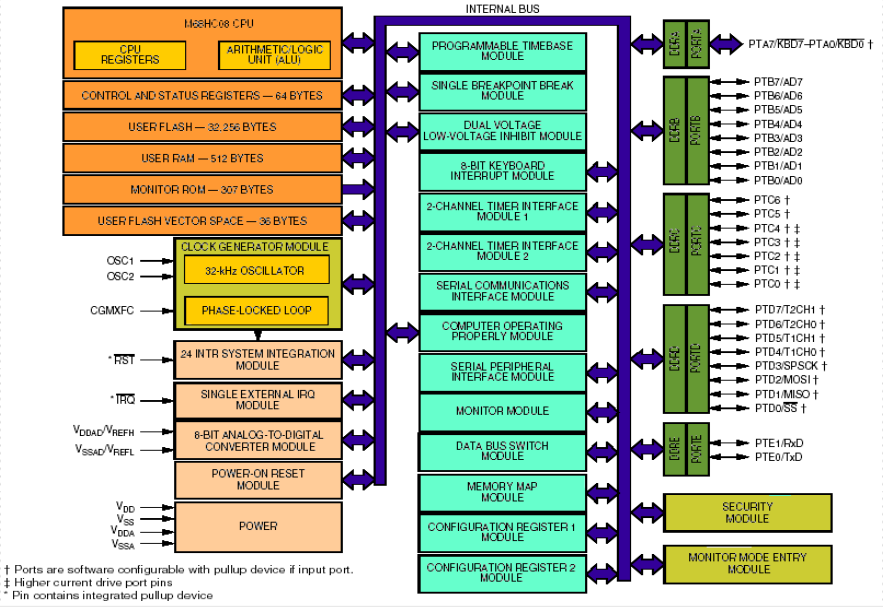
Program	hlcc.asm	hht1.asm	hht2.asm	hht3.asm
Avg. Current (mA)	525.7	534.2	507.6	486.6
Execution Time (μsec)	11.02	9.37	8.73	7.07
Energy ($10^{-6}J$)	19.12	16.52	14.62	11.35
Program	**clcc.asm**	**cht1.asm**	**cht2.asm**	**cht3.asm**
Avg. Current (mA)	530.2	527.9	516.3	514.8
Execution Time (μsec)	7.18	5.88	5.08	4.93
Energy ($10^{-6}J$)	12.56	10.24	8.65	8.37

  - hlcc.asm → assembly code generated by lcc, a general purpose C compiler that produces good code.
  - hht1 → Hand tuning for shorter (a 15% reduction) running time. Only temporarily variables allocated in registers.
  - htt2 → 3 local variables allocated in registers and the appropiate memory operands are replaced by register operands
  - htt3 → 2 more local variables allocated in registers and all redundant instructions are removed.
- As a result, the sort algorithm has suffered a 40.6% reduction in power consumption, and the reduction in the circle algorithm is about 33%.

Power optimisation at the MC68HC908GP32 Microcontroller

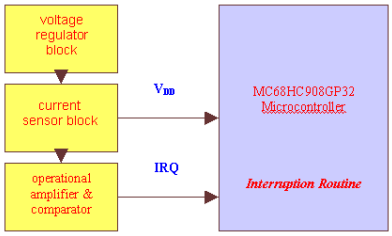
- The MC68HC908GP32 microcontroller
  - It is an actual microcontroller based on an 8-bit technology with low-cost and high-performance attributes characteristic for the M68HC08 family. The internal bus frequency is 8MHz.
  - Internal registers are: 8-bit accumulator, 16-bit index (acting also as a general purpose) register, the 16-bit program counter and an 8-bit condition code register. It includes also a stack pointer register of two bytes.
  - The whole address space comprises 64KB divided in different functional regions.
  - The stack addressing mode compensates the lack of internal registers.
  - It has 5 8-bit port sets (four are bifunctional). There are 20 different modules, each of them dedicated to a given task, not to mention the CPU and the execution module (computer operating properly).
  - The microcontroller has a low-voltage inhibit module which monitors the V_{DD} value and forces a reset if it falls below a critical voltage.
  - It can operate at 3V or 5V power supply. The maximum current in transitory state attains 100mA, although during the steady state the order of magnitude is at most of tens of milliamperes (at the port pins of maximal values of 10-15mA).
  - It has special suitability for low-power applications. It possesses two idle operating modes named wait and stop. Both are characterized by a reduction of the power consumed. In the wait state only the CPU clock is disabled, whereas during the stop state, the control of the almost entire module spectrum is relinquished, thus lowering even more the power consumption.

The MC68HC908GP32 Microcontroller



Measurement setup

- Current measuring circuitry composed of
  - Voltage regulator. A stable supply voltage is supplied to the whole system.
  - Current sensor block, It uses a high precision resistor.
  - Operational amplifier and comparator.
    - The output level of the comparator swings whenever the voltage at the one of the two inputs exceeds the reference value. The time interval this process lasts counts for the actual current flown.
    - As soon as the output of the comparator changes in value, an interruption is issued and the computed current is stored into a table.
  - The whole process is repeated several times under the same test conditions and an averaged value is thus obtained.



Instruction groups according to the number of cycles

Cycles	Instruction Type	Addressing Mode
1	1.1. *shifts, rotations, ± n, negations* 1.2. *resets, test for < or 0* 1.3. *transfers to/from accu, nop, wait*	inherent mainly with respect to accu
2	2.1. *arithm. ops.(other than 1.1.), load accu & ix* 2.2. *1.2.* 2.3. *transfers to/from ix* 2.4. *clear/set interr.flag* 2.5. *jumps* 2.6. *pushes, pulls*	immediate and indexed indexed inherent direct, indexed inherent
3	3.1. *2.1. & store accu & ix* 3.2. *1.2.* 3.3. *load & compare ix* 3.4. *decr-br-not-zero* 3.5. *2.5.* 3.6. *relative branches* 3.7. *1.1.*	direct, indexed-8 immediate inherent, with accu & ix external, indexed-8 relative indexed
4	4.1. *3.1.* 4.2. *1.2.* 4.3. *3.3.* 4.4. *set/clear bit in mem* 4.5. *2.5.* 4.6. *moves* 4.7. *1.1.* 4.8. *comp-br-if-equal* 4.9. *jump to subroutine* 4.10. *branch to subroutine* 4.11. *return from subr.*	external, indexed-16, stack-8 stack-8 direct indexed-16 dir.+idx., idx.+dir., imm.+dir. direct, indexed-8 imm. with accu&ix, indexed-8 direct, indexed relative inherent
5	5.1. *3.1.* 5.2. *multiplication* 5.3. *3.4.* 5.4. *br. if bit set/clear* 5.5. *4.6.* 5.6. *1.1.* 5.7. *4.8.* 5.8. *4.9.*	stack-16 inherent direct, indexed-8 direct direct + direct stack-8 direct, indexed-8 external, indexed-8
6	6.1. *4.8.* 6.2. *4.9.*	stack-8 indexed-16
7	7.1. *division* 7.2. *return from interrupt*	inherent
9	9.1. *software interrupt*	inherent

Analysis from instruction set cycling

- Some notes from the instruction ordering can be stated:
  - The *inherent* addressing mode usually requires the smallest number of cycles. Nevertheless, there are some exceptions, such as: division, multiplication, instructions related to interrupts and subroutines.
  - From addressing modes can be stated:
    - *direct* precedes always *external*, if for the same subgroup both addressing modes are implemented.
    - for *jumps* (2.5; 4.9) it is the unique case when *indexed* and *direct* share the same number of cycles, otherwise, *indexed* precedes *direct*.
  - Within the *jump* subgroup, *jumps* require less cycles than *subroutine calls* and, in turn, are performed more quickly than a *software interruption*. The *relative* addressing mode refers exclusively to this subcategory.
- The first conclusion to arrive is that next following step is clock saving:
  - Address the direct page than the external memory.
  - Use 8-byte indexation than 16 (locality of data).
  - Prefer memory instead of stack addressing.
  - Use indexed than direct addressing .

Instruction and sequence alterations (I)

- The series of improvements in terms of energy savings can be complemented with one-off steps, gathered in the present subsection:
  1. If it resulted better to address the (page zero) memory instead of accessing the stack (subsection *B*), the same can be affirmed, or seen as a mere consequence, in the case of the instructions TAX / TXA (transfers between accumulator and index register) with respect to the homologous ones implying the stack (TSX, TXS).
  2. Whenever possible, avoid using *division* (DIV) and *multiplication* (MUL), if, for example, the multiplicand or dividend is a power of 2 (care must be taken to the magnitude of the exponent when deciding which of the two alternatives gives better results). Or, if the same operation can be performed as a succession of shifts and sums, vid. subtractions.
  3. The interruption process consumes more time, hence energy, for it is necessary to push on the stack not only the program counter content as it happens when a routine is called, but also those of the index, condition code register, and accumulator and furthermore reset the interruption flag. That is why it is better to avoid SWI instructions in favour of JSR (routine calls). The same discussion applies to the corresponding return commands. The savings attain 4 and 3 cycles, respectively.
  4. A similar justification can be adduced for a simple jump (branch) compared to a jump to subroutine. The difference between the number of cycles that reflects the duration of an ordinary jump in relation to a routine call with the same addressing mode is of two units. One should add to it the accompanying return instruction, 4 cycles against an average of 3 for a back jump.

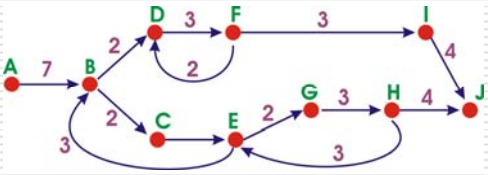
Instruction and sequence alterations (I)

5. A single instruction is sometimes more beneficial to our purpose than a couple which affords the same result (it shortens the program length, too). As an example, when moving data from one memory location to another, it is advisable to have it done by means of a unique move (MOV) instead of considering a sequence of *loading* and *storing* the data in and from the index register (LDX, STX).
6. If the semantic of program is known, as for example the data range with which it oftener runs, it is possible to change the branch conditions such that the taken jumps should result less frequent than the untaken.
7. Any independent operation should be taken out of forks and cycles, since on the contrary they will count for both branches or every iteration, respectively.
8. When the number of iterations a cycle performs is known in advance and its value is sufficiently small, it is better to replace it with its body replicated the same number of times.
9. Other improvements imply disentangling nested calls and merging calls with the same iteration number.

Assembly code optimisation compiler

```
A:   LDA displ,X
B:   BEQ C
D:   ADD #val1
F:   DBNZA
I:   BEQ J
C:   EOR #val2
E:   BPL B
G:   AND #val3
H:   BMI E
J:   STA displ,X
```

- Postprocess can help code optimisation.
- Cost digraph is used to find the low-energy path.
- An important software level algorithm optimisation ways can be applied for each case.



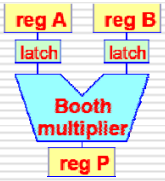
Minimisation techniques in DSP processor -I

- Lee, Tiwari, … analyse power model at instruction level in embedded DSP. Some significant points are:
  - Greater power consumption in DSP due to circuit state changes. That means that, with appropriate scheduling of instructions can lead to a reduction in the power-cost of the programs.
  - The study shows that faster programs consume less energy. Scheduling for power minimisation is explored.
  - Special architectures of the DSP processor provided to reduce the number of cycles for programs are also very effective for reducing the energy cost of programs:
    - Double data transfers from different memory banks to registers in one cycle.
    - Packing of two instructions into a single code-word.
  - Also, on-chip Booth multiplier is a major source of energy consumption for DSP programs.
    - Proposal of an effective technique for local code modification by operand swapping to power consumption reduction.
  - The energy minimisation methodology applied to a given piece of code on a Fujitsu (3.3V, 0.5μm, 40MHz, CMOS) embedded DSP processor shows energy reductions ranging from 26% tp 73%.

Minimisation techniques in DSP processor -II

- Effect of circuit state overhead
  - An inter-instruction effect (that affects the circuit state) due to the spetial on-chip multiplier design.
    - Current difference between adjacent instructions i and j in respect to their separate base current → It is considered as a measure of the change in circuit state.
  - Considering the following four instructions without overhead cost between instructions 1 and 3, the total current is 176.2
  - Measured current is 204.0. Why?

number	instruction	base	overhead	cycles
1	MUL:LAB (X0+1),(X1+1)	37.2	(1&2) 18.4	1
2	NOP	14.4	(2&3) 18.4	1
3	MUL:LAB (X0+1),(X1+1)	36.6	(3&4) 18.4	1
4	NOP	14.4	(4&1) 18.4	1
total:		102.6	+ 73.6	= 176.2



- Effects of a latch introduction in the multiplier
  - Retention of old multiplicand values until new execution of the next multiplier instruction.
  - State change at such imput latches accounted by the overhead instructions 1 and 3.
    - The average value was determined to be 12.5mA
    - So, global overhead is two times (1 to 3, and 3 to 1) → 27.6mA

Minimisation techniques in DSP processor-III

- Instruction level power model
  - For the target DSP processor, the instructions most commonly used were categorized into 6 unpacked classes.
    - Instructions in the same class have similar functionality and activate similar part of the CPU. Hence they have similar characteristics with regards to the current drawn.

class	addressing	functionality
LDI	immed → reg	load immediate data to a register
LAB	mem1 → reg A, mem2 → reg B	transfer memory data to registers A, B
MOV1	reg1 → reg2	move data from one register to another
MOV2	mem → reg, or reg → mem	move data from memory to a register, or from a register to memory
ASL	reg specified implicitly	add/sub, shift, logic operations in ALU
MAC	reg specified implicitly	multiply and accumulate in ALU

  - The range of base cost for different operands is calculated for each instruction unpacked. Table shows range and average base.
  - Also the overhead costs between instructions belonging to different classes are calculated.
    - Table shows a significant variation accros various entries. Data in the table suggests that choosing an appropriate order of instructions can lead to an energy reduction.

	LDI	LAB	MOV1	MOV2	ASL	MAC
range	15.8 - 22.9	34.6 - 38.5	18.8 - 20.7	17.6 - 19.2	15.8 - 17.2	17.0 - 17.4
average base	19.4	36.5	19.8	18.4	16.5	17.2

	LDI	LAB	MOV1	MOV2	ASL	MAC
LDI	3.6	13.7	15.5	6.3	10.8	6.0
LAB		2.5	1.9	12.2	20.9	15.0
MOV1			4.0	18.3	10.5	3.8
MOV2				25.6	26.7	22.2
ASL					3.6	8.0
MAC						12.5

Minimisation techniques in DSP processor -IV

- Base cost of packed instructions that have a data transfer instruction as a component, is very close to the base cost of the unpacked data transfer instruction alone.
  - An ALU-type instruction and a data transfer instruction can be packed into a single instruction codeword for simultaneous execution.
    - Except for instructions that have a packed MAC, most packed instructions have small range of variations.
      - MAC responds for Multiplier and Accumulator unit
    - Depending on the application, packed instructions MAC:LAB can show large overhead variations (from 1.4mA to 33.0mA).

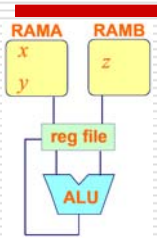
instruction	ASL:LAB	ASL:MOV1	ASL:MOV2
range	34.5 - 38.7	15.7 - 17.4	18.7 - 20.4
average base	36.6	16.6	19.6
instruction	MAC:LAB	MAC:MOV1	MAC:MOV2
range	33.9 - 39.9	15.9 - 18.9	19.0 - 21.2
average base	36.9	17.4	20.1

adjacent instructions	overhead
LDI, ASL:LAB	23.8
LDI, ASL:MOV1	19.6
LDI, ASL:MOV2	20.0
MOV1, ASL:LAB	14.7
MOV2, ASL:LAB	27.1
MOV2, ASL:MOV2	19.3
MAC, ASL:LAB	27.9
ASL:LAB, ASL:MOV1	27.1
ASL:LAB, MAC:MOV1	29.3
ASL:MOV2, ASL:MOV2	18.7

Variation of overhead cost for MAC:LAB

reg A * reg B	overhead
EFFFFF * FFFFFF	1.4
EFFFFF * EFFFFF	
7FFFFF * 555555	17.3
7FFF00 * 555666	
AAAAAA * 555555	33.0
555555 * AAAAAA	

Minimisation techniques in DSP processor -V



- Energy minimisation for the DSP
  - Memory bank assignment for low energy

The DSP has two on-chip data mamory banks RAMA and RAMB that supply data to the register file for an ALU operation, in the same cycle, or by double a transfer instruction–LAB. When both operands are stored in the same memory bank, two single transfer instructions MOV's are needed, and this takes two cycles.

The double transfer instruction takes about half the energy.

So, in order to reduce energy consumption, variables in an embedded program should be assigned to memory to allow maximal use of double-transfer instruction.

  - Instruction packing for low energy

unpacked	packed
LAB (X0+1),(X1+1)	LAB (X0+1),(X1+1)
MSPC	MSPC:LAB (X0+1),(X1+1)
LAB (X0+1),(X1+1)	MSPC:LAB (X0+1),(X1+1)
MSPC	...
...	MSPC:LAB (X0+1),(X1+1)
...	MSPC
LAB (X0+1),(X1+1)	
MSPC	

The single packed instruction represents the same functionality as the sequence of two unpacked instructions, but always leads to a reduction in energy.

For example, a multiply instruction (MSPC) and a LAB, can either execute as unpacked instructions for a total of two cycles, or as a single packed instruction that executes in just one cycle.

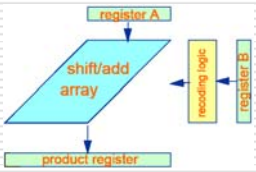
As seen from the results, the average current drawn by packed instructions is only marginally higher than for the unpacked instruction. However, the number of cycles halves.

reg A * reg B	packed (mA)	unpacked (mA)
AAAACC * FFFF00		
CCA0A0 * 898989	65.1	60.4
000000 * 000000		
0FFFFF * 0FFFFF	53.3	53.3

Minimisation techniques in DSP processor -VI

- Energy minimisation for the DSP
  - Instruction scheduling for low power
    - The circuit state overhead cost has significant variation across different instruction pairs. Thus, different instruction schedules for the same program can consume different power.
    - An auromated instruction scheduler looks up the overhead cost tables and chooses a good instruction schedule without violating data dependencies.
  - Operand swapping for the Booth multiplier
    - Multiplications in the MAC unit is usually (because of its complex design) a major source of power consumption.
    - It is usual to have to do filter operations such as $\Sigma c_i * X_i$ in DSP applications
    - The fundamental idea behind Booth algorithm it is to recode B data by the 'skipping over 1s' technique, in order to reduce the number of operations (additions and shits) to perform with A data.
    - For example, for a 7-digit B value 0011110 only four additions of shifted A are necessary.
    - So, the microarchitecture of the Booth multiplier does not treat A and B simetrically. So, if the weigth of A is smaller than that of B, the number of additions and substractions are reduced by just swapping the operands in registers A and B.

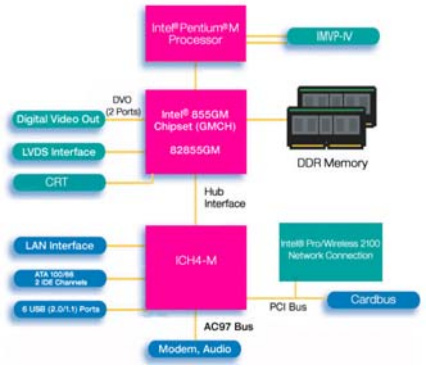
B		version of A selected
bit i	bit i-1	by bit i of B
0	0	0 * A
0	1	1 * A
1	0	-1 * A
1	1	0 * A



no.	operands		measured current		%saving
	op1	op2	op1 * op2	op2 * op1	
1	7FFFFF 000001	AAAAAA AAAAAA	58.9	46.9	20.4%
2	7FFFFF 000001	666666 AAAAAA	68.5	47.9	30.1%
3	7FFFFF 000001	AAAAAA 000001	65.7	49.1	25.3%

Low power in Intel® 855GM Chipset-I

- The Intel® 855GM chipset brings breakthrough integrated graphics performance and low power to Intel® Centrino™mobile technology platforms.
  - Intel Centrino mobile technology is based on the four vectors of mobility: performance, battery life, small form factor, and wireless connectivity
  - The mobile chipsets are optimized specifically for both performance and low power from the architectural definition through the validation effort. Low power consumptions enables a extended battery life of more than eight hours of usage with a single battery.



- For mobility, power/performance features were designed into the chipset microarchitecture.
  - Clock gating to reduce the average power of the integrated graphics engine
  - Delay Locked Loop (DLL) Power Down reduces power in theUnified Memory Architecture (UMA)
  - DRAM row power to reduce Dual In-line Memory Module (DIMM) system power.

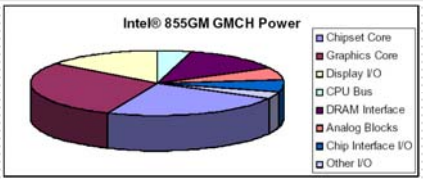
Low power in Intel® 855GM Chipset-II

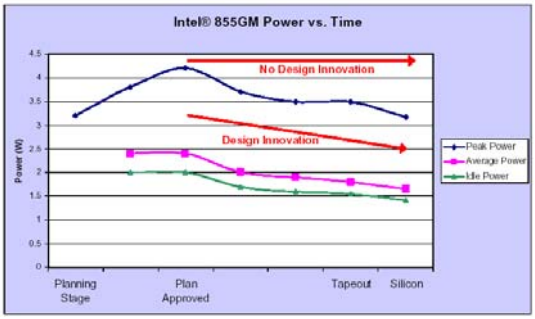
- Power management architectures
  - Main Memory Power Management.
    - Main memory power managed at normal and low-power advanced configuration.
    - Based on idle conditions in a given row of memory, that memory row may be powered down.
    - If the pages of a row have all been closed at the time of power down, the device wii enter in a active power down state. Otherwise if pages remain open, the device will enter in a precharge power down state.
  - Graphics Memory Controller Hub Dynamic IO/DLL Power Management.
    - Use of memory address tri-states when all memory powered down or self-refresh, memory clock tri-states for unpopulated DIMMs, disable control for control sense amps, data bus sense amps.
    - Use of DLL (Delay Locked Loops) for adjusting input signals to data strobe signals. DLLs are designed with master and slave parts. The master calibrates the delay elements to tuen the entire delay line. The slave is the actual delay line uses to delay a functional signal.
    - DLL's are disabled when possible.
  - Global design criteria.

    Global design criteria has been followed to adjust the parameters:
    - Validation methodology, from modeling hardware for logis simulation to BIOS validation in a pre-silicon environment.
    - Modeling components for logic simulations
    - Revision of circuit-level components .
    - Hardware emulation and BIOS validation.

Low power in Intel® 855GM Chipset-III

- Power tracking during design
  - During the product definition phase, targets for peak and average power were established based on past chipsets experience .
  - At pre and post-silicon phases, the power was tracked with a spreadsheet as a sum of individual components, with a closed loop process to unsure designers were aware of how far they exceeded the target with new design innovation.





Low power in Intel® 855GM Chipset-IV

- Results show significant power savings at idle, as well as savings under a sample intense graphics workload.
- Example shows three partitions (2D, 3D, and memory interface) with their individual clock gating disabled to show the impact to core power.
- Savings are produced even under intense workload by aggressively targetting mutually exclusive units in each partition.