An Accurate Energy Estimation Framework for VLIW Processor Cores

Sourav Roy, Rajat Bhatia, Ashish Mathur
India Design Center, Freescale Semiconductors
sourav.roy@freescale.com; rajat.bhatia@freescale.com; ashish.mathur@freescale.com

Abstract— **In this paper, we present a comprehensive energy estimation framework for software executing on Very Long Instruction Word (VLIW) processor cores. The proposed energy model is used as an average energy estimator coupled to the instruction set simulator (ISS) of the processor. The base energy of an execution set is computed as the NOP energy added with incremental energies of each instruction in the execution set. The inter execution-set energy is accurately modeled with a new approach as a linear equation of three factors - functional to functional instruction switching; functional to NOP or prefix instruction swithing; and variability in the length of the execution set. This reduces the characterization complexity of the model to $O(N)$, where N is the total number of instructions in the instruction set of the processor. We have introduced the concept of "functional separability" in the energy model, wherein the energy of each high-level function of the processor core is distinctly mapped to only one component in the model. The model is also capable of handling control codes with branches and predicated execution. The average error magnitude of the framework when applied on the StarCore processor with a large suite of DSP and control benchmarks is 2.5%, whereas the maximum error is less than 6.0%.**

I. INTRODUCTION

Low power is an inherent requirement for hand-held devices like cell phones, PDAs and a host of other mobile apparatus. The digital signal processor (DSP) has inceasingly become an integral component of these mobile devices. Most of the state-of-the-art DSPs are VLIW in nature, e.g. StarCore from Freescale Semiconductors, TigerSHARC from Analog Devices, TMS320C64X from Texas Instruments, etc. Hence energy estimation of the DSP software on these VLIW machines is key to designing low power systems. Different implementations of the same software can consume significantly different amounts of energy in the active mode. This variation in energy can be due to the computational complexity of the algorithm employed, usage of different compilers and compiler options, and structuring and hand-optimization of the C and assembly codes. Energy or power variation is more prominent in a VLIW processor than a simple single-issue processor due to the availability of high degree of instruction-level-parallelism. Once the energy consumed by the software is estimated, the software programmer can use several software techniques to reduce the power consumption. Also energy estimation permits software design-space-exploration based on cycles, code-size and energy. Moreover accurate average energy prediction can help us to perform battery life estimation of the portable system. While energy is important for portable devices, power is important from a thermal stability perspective.

There are several approaches to estimate the energy of a processor viz., gate-level [1], microarchitectural level [2], [3] and instruction level. The instruction level model is the most useful for the software programmer, since he can clearly identify the energy consumption with each individual instruction. It is also easy to integrate with an ISS. The pioneering work in this field is from Tiwari et.al [4]. In this work the authors identified a base energy cost with each assembly level instruction of the processor. Apart from this, the energy of an instruction is also influenced by the instructions preceding it. This is termed as inter-instruction effect. In [5], the inter-instruction energy for RISCs and DSPs is modelled as a table look-up between each pair of instructions. To reduce the complexity, the instructions are clustered to form a few groups and the table is calculated between pairs of these groups. Based on this work, several models for single-issue processors have been proposed. However inspite of the fact that multiple-issue machines like the VLIW core is more prone to higher degree of power variation, energy models for multiple-issue machines have not been investigated thoroughly. Sami et.al. [6], [7] have proposed a basic model for VLIW processors. However this model has several limitations when applied on practical VLIW processors, as we discuss later.

In this paper, we propose a novel energy model for VLIW processor cores which is accurate, comprehensive and easier to implement in practice. This model is generic in nature as it can be applied to most of the VLIW processor cores. The proposed methodology is used in building an accurate energy estimator from circuit-level simulation of post-layout netlist of the DSP core with Nanosim [8]. An accurate pre-silicon energy model is very useful for the embedded software developers as they generally start application development well before the silicon arrives. It is also useful for software developers who do not have access to the silicon or the instruments needed to accurately measure power. This energy model is primarily meant to be used at run-time with the ISS of the DSP core. With small modifications it can also be applied in non-runtime environment, but with lower accuracy. In our methodology, the energy for the memory sub-system as well as other peripherals is separately calculated. The reason for separately reporting the core and memory power is to provide the software programmer better insight into the energy consumption of the embedded DSP platform. In many cases, the user can trade-off between memory and core energy consumptions. Further a separate energy model for the DSP core will also enable reuse as the same core is used in several platforms with different memory organizations. In this paper we particularly concentrate on estimating energy of the DSP core. The paper is organized as follows - in section II, a generic VLIW DSP is described in brief. In the next section III we describe the energy model for the VLIW DSP. Later we describe our characterisation methodology to obtain the energy tables. Finally we discuss the results and compare it with previous works.

II. The VLIW DSP

In a VLIW DSP, several instructions that execute simultaneously are grouped into an "execution set" (which is the very long instruction word). The execution set can have fixed or variable number of instructions. The former is a fixed length VLIW processor, whereas the latter is referred as a variable length VLIW processor. Fixed length processors use no-operation (NOP) word padding to equalize the length of all execution sets. In a variable length processor, the grouping of individual instructions into an execution set is generally done either with separate prefix words or with a few bits in each instruction to form a serial chain. VLIW DSPs have multiple execution units for parallel execution of instructions. For example the StarCore SC140e DSP [9] packs upto four arithmetic instructions and two move like instructions in a single execution set. The core typically interacts with a memory sub-system comprising of the instruction cache, the data cache and an optional SRAM memory.

III. The Energy Model

Let us consider a VLIW processor core **V**. A group of instructions, which are executed in parallel on **V** constitute an execution set. We consider an assembly program **P** of m execution sets. The energy of an individual execution set i is denoted by E^i. Further let us assume that **V** has q different types of pipeline stalls due to the memory sub-system - e.g., cache misses, memory contention etc. Let the energy per cycle of stall type j be given by E_s^j. Let the number of cycles due to stall type j while executing **P** be c_j. If the total number of stall cycles due to the memory-subsystem while executing **P** is k, then $\sum_{j=1}^{q} c_j = k$. The total energy while executing **P** on **V** is then given by:

$$E_{tot} = \sum_{i=1}^{m} E^i + \sum_{j=1}^{q} c_j * E_s^j \tag{1}$$

The execution set energy E^i can be further decomposed into a base energy component E_b^i and an inter-execution set energy component E_{ies}^i . The base energy of an execution set is the energy consumed by the execution set itself, while the inter-execution set energy is due to its change from/to neighboring execution sets. Hence equation 1 can be refined as:

$$E_{tot} = \sum_{i=1}^{m} (E_b^i + E_{ies}^i) + \sum_{j=1}^{q} c_j * E_s^j \tag{2}$$

Next we model the base energy of an execution set. Let the ith execution set consist of N_i individual instructions denoted by the set $\{I_1, I_2, \ldots, I_{N_i}\}$, which are executed in parallel. The background energy of the execution set is defined as the energy consumed by an execution set consisting of only NOP instructions executing on **V**. This NOP background energy of the execution set is primarily dependent on two factors - the position of the execution set (s_i) and the length of the execution set (N_i). It is denoted as $E_{NOP}(s_i, N_i)$. Examples of position s are straight-line, normal hardware loops, special hardware loops without instruction fetches operating from L0 buffer etc. Over and above the background energy, each of the N_i individual instructions in the execution set contributes an incremental energy per cycle denoted by ΔE_k for $k \in \{I_1, I_2, \ldots, I_{N_i}\}$. The incremental energy ΔE_k is the average energy of that instruction. For implementation ease it is computed in such a way as to average out the energy variation due to the data values of the instruction operands. For a more accurate energy estimate, ΔE_k is a function of instruction operand values and other parameters like instruction modes. Let the execution time of the kth instruction be n_k cycles. Then the total execution time of the execution set is given by $\max(n_{I_1}, n_{I_2}, \ldots, n_{I_{N_i}}) = p$ cycles (say). The base energy is then modeled as the summation of the background NOP energy of the execution set and the incremental energies of the execution set.

$$E_b^i = p * E_{NOP}(s_i, N_i) + \sum_k n_k * \Delta E_k \tag{3}$$

If the VLIW processor **V** supports predicated execution, then equation 3 needs to be further modified. For predicated instructions, if the condition is true (i.e. the instruction is executed) then the incremental energy is the same as given by ΔE_k. But if the condition is false (i.e. the instruction is not executed) then the incremental energy reduces to ΔE_{k^r}, where ΔE_{k^r} = reduced incremental energy of the instruction. Further the execution time of the predicated instruction with false condition should be considered as 1 cycle instead of n_k cycles.

$$\begin{aligned} \text{Let } \delta_k = 1, \quad & \text{if } k\text{th instruction is a predicated} \\ & \text{instruction with false condition} \\ = 0, \quad & \text{otherwise.} \end{aligned}$$

Then the modified execution time and incremental energy of the kth instruction can be represented as:

$$n'_k = \delta_k * 1 + (1 - \delta_k) * n_k \tag{4}$$

$$\Delta E'_k = \delta_k * \Delta E_{k^r} + (1 - \delta_k) * \Delta E_k \tag{5}$$

Hence the base energy of the ith execution set is given by:

$$E_b^i = p' * E_{NOP}(s_i, N_i) + \sum_k n'_k * \Delta E'_k \tag{6}$$

where $p' = \max(n'_{I_1}, n'_{I_2}, \ldots, n'_{I_{N_i}})$.

The inter-execution set energy E_{ies}^i is due to the change of the execution sets in the controller part of the processor like fetch, decode, and dispatch units. Actually there is also an inter-execution set energy due to change of operand values at the functional unit inputs. But we consume this data part of the inter-execution set energy in the base energy calculation of instructions. This is done by either averaging out the change in operand values while computing the incremental energy numbers or using a mathematical function based on the operands to represent the incremental energy. Hence we just concentrate on the control part of the inter-execution set energy E^i_{ies}. The instructions in an execution set are executed in parallel along several "lanes" or "ways" in the VLIW processor. We divide the functional instructions into a few clusters based on functional units like say multiply, ALU, load/store, shifter and control. The NOP and prefix instructions form a separate special cluster of non-functional instructions. Let the energy consumed when an instruction of functional cluster c changes to another instruction of functional cluster d in a lane of **V**, be denoted by $k_1(c,d)$. Also when an instruction of functional cluster c changes to a NOP or prefix instruction in a lane of **V**, let it consume an energy given by $k_2(c)$. For execution set i let, $y^i_1(c,d)$ be the number of changes from one instruction in functional cluster c to another instruction in functional cluster d in all lanes of **V**; and $y^i_2(c)$ be the number of changes from one instruction in functional cluster c to a NOP or prefix instruction in all lanes of **V**. Then the inter-execution set energy of the execution set i can be modeled as:

$$E^i_{ies} = k_0 + \sum_{c,d} k_1(c,d) * y^i_1(c,d) + \sum_{c} k_2(c) * y^i_2(c) \quad (7)$$

Here the constant k_0 primarily models the variability in length of execution sets. In majority of VLIW processors the inter-execution set energy is much smaller compared to the base energy of an execution set. In such cases, with limited loss of accuracy the inter-execution set energy model of equation 7 can be further reduced as follows: For all combinations of functional clusters c and d, $k_1(c,d) = constant = k_1$. Similarly for all functional clusters $c, k_2(c) = constant = k_2$. Then equation 7 reduces to:

$$\begin{aligned} E^i_{ies} &= k_0 + k_1 * \sum_{c,d} y^i_1(c,d) + k_2 * \sum_{c} y^i_2(c) \\ &= k_0 + k_1 * x^i_1 + k_2 * x^i_2 \end{aligned} \quad (8)$$

where for execution set i, x^i_1 = total number of changes from one instruction in a functional cluster to an instruction in another functional cluster in all lanes of **V**; and x^i_2 = total number of changes from one instruction in a functional cluster to a NOP/prefix instruction in all lanes of **V**. Depending on the implementation on the processor controller, in some processors clustering based on functional units do not work well. In such a case equation 8 can still be applied with x^i_1 = total number of changes from one functional instruction to any another functional instruction in all lanes of **V**; and x^i_2 = total number of changes from one functional instruction to a NOP/prefix instruction in all lanes of **V**.

So finally the energy consumed by the assembly program **P** on the VLIW processor **V** is given by using equations 6 and 7 [or 8] in equation 2.

IV. Model Characterization

The model characterization step involves creating the NOP energy table, the incremental and reduced incremental energy tables, the stall energies and the inter-execution set energy coefficients. Our methodology involves circuit-level simulation of post-layout netlist of the DSP core with Nanosim [8]. Specific assembly routines are used and the vectors corresponding to these routines are used to excite the Spice netlist in Nanosim. This gives very accurate energy numbers. In this section we describe the methodology of creating the energy values and coefficients.

A. *The NOP energy*

The NOP energy signifies the background energy consumed while executing a series of NOP instructions on the processor. It includes clock energy, leakage energy and instruction fetch and decode energy. But since it fetches and decodes only NOP instructions, it is only a part of the fetch and decode energy. The NOP energy significantly varies depending on its position in the program. For a short hardware loop operating from a L0 buffer, the DSP does not fetch the instructions after the first time. The long hardware loop has much higher energy as it involves fetch of execution sets. The NOP energy in a straight line sequence is similar to that of the long hardware loop, but is slightly less as there is no control overhead and change-of-flow of the program. The NOP energy also increases with increasing length of the execution set. This is primarily due to increased rate of program address generation, as the processor has to fetch more number of program words in the same number of cycles. Since the program words are all virtually NOPs there is no switching happening on the program data bus. In table I, we plot the NOP energy of the SC140e processor as a function of position and length of the execution set. For short hardware loops, we have ignored the length effect as it is very small. We compute the NOP energy table of the processor by running a sequence of NOPs (packed in specific numbers corresponding to the length of the execution set) placed in a specified position like hardware loops or straight line.

B. *The Incremental Energy*

Computing the incremental energy of each instruction is the most important aspect of energy estimation. The most common approach is to execute the same instruction in a hardware loop for a large number of cycles, ensuring that there are no pipeline stalls due to the memory sub-system. The average current during the loop execution is measured and converted to an energy value. If available, special short hardware loops should be used for better accuracy as they do not require a program fetch after the first time. For some instructions that permanently change the destination register at the first time (e.g., CLR R0), we still use a loop to keep the

TABLE I
NOP ENERGY(PJ) PER CYCLE ON SC140E

NOP Position	VLES length							
	1	2	3	4	5	6	7	8
short loop1	98.3	98.3	98.3	98.3	98.3	98.3	98.3	98.3
short loop2	109.0	109.0	109.0	109.0	109.0	109.0	109.0	109.0
long loop	109.4	112.8	114.8	116.1	118.5	119.2	118.8	119.0
striaght line	105.1	107.1	112.1	112.0	115.8	116.6	117.6	118.0

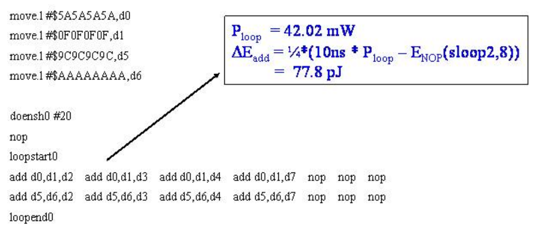
processor pipeline filled up with the same instruction. However we measure the current during the first execution cycle only.

The other important issue here is that for incremental energy calculation of most instructions, we need to consider the effect of the operands that are dispatched to the functional units. During a normal program flow, the inputs to a functional unit in the DSP will normally change every time its instruction is executed. This is what we term here as the inter-instruction data energy, as it involves the datapath of the processor. In most modern day processors, the inputs to the datapath only change when it is required. Hence the input switching on the adder can be due to two ADD instructions located very far apart in the program. Calculating inter-instruction data effects can be done only at run-time, with detailed energy models of each datapath unit of the DSP. This is not practically feasible. Also it can make the simulator very slow. Hence as an engineering trade-off for normal DSP programs we propose an average model for incremental energy where we consume the inter-instruction data effect in the base energy calculation. We can activate the instruction with random data for a large number of cycles and measure the incremental energy value. However, we have followed a different approach by using two or more precomputed sets of inputs vectors for each datapath instruction. The switching between these two input vectors is 50%. i.e. for 16 bit words 8 bits toggle. Also we account for the datapath characteristics by assuming 50% effect. For example, 50% of the bits during addition generate a carry. For 16 bit multiplication with radix-4 Booth recoding, all 8 combinations of multiplier bits during Booth recoding occur once. We further incorporate both positive and negative numbers equally to account for the extra switching due to sign changes. There are several advantages of using this strategy instead of random inputs.

1. Most modern DSPs are load-store processors. Hence if we use random values they have to be loaded from memory into internal registers so that they can be used as inputs to the datapath units. This somewhat complicates the incremental energy calculation due to extra load instructions used with the datapath instruction being characterized.

2. If we use pre-computed pair of input vectors, we need to run the loop only a few times (say 20) to characterize the energy. We just need to wait for the pipeline to fill up completely with the instruction to be characterized; then observe the average energy for a few cycles. However if we use random values we must execute the loop several thousand times. Hence our method speeds up the characterization time by a significant factor.

As an example we can use (0x5A5A + 0x0F0F) followed by (0x9C9C + 0xAAAA) for characterizing the ADD instruction. Similarly we can use (0x5A5A * 0x2E74) followed by (0x9696 * 0x74E8) to characterize MPY. It requires a one-time effort to come up with a suitable pair of input vectors which more-or-less satisfy the conditions of 50% switching and functionality, which is fairly straight-forward. In figure 1 we show how we use a short hardware loop of two execution sets on the SC140e DSP with pre-computed data inputs to characterize the ADD instruction. We have used the maximum number of ADD instructions permissible in an execution set i.e. 4, so that any energy difference across the several adders is averaged out. Using multiple ADD instructions is permissible due to the additivity property of their incremental energies. Also NOP word padding is used to ensure that the intended data values reach the corresponding functional unit (in this case they are all equal). NOP word padding also eliminates unnecessary toggles in the instruction register which can otherwise add some inter-instruction energy into the base cost.



```
move.l #$5A5A5A5A,d0
move.l #$0F0F0F0F,d1
move.l #$9C9C9C9C,d5
move.l #$AAAAAAAA,d6

doensh0 #20
nop
loopstart0
add d0,d1,d2   add d0,d1,d3   add d0,d1,d4   add d0,d1,d7   nop   nop   nop
add d5,d6,d2   add d5,d6,d3   add d5,d6,d4   add d5,d6,d7   nop   nop   nop
loopend0
```

$P_{loop} = 42.02\ \mathrm{mW}$

$\Delta E_{add} = ¼*(10\mathrm{ns} * P_{loop} - E_{NOP}(\mathrm{sloop2,8}))$
$= 77.8\ \mathrm{pJ}$

Fig. 1. Incremental energy calculation of ADD

However, it is to noted that this model is not accurate for advanced DSP programs. The user can use low power software techniques like MAC input operand swapping, introduction of a bias in the accumulator etc. In such cases this model will over-estimate the energy. Hence we built a linear run-time model of the MAC unit of the DSP, based on present and preceding input values and the characteristics of the MAC itself (like Booth recoding and number of zeroes or ones in the MAC inputs). This model is invoked by the user only for advanced low power DSP techniques. In practice for most DSP programs, the 50% input switching and functionality assumption is a safe approximation.

The reduced incremental energy numbers are computed in the same way as incremental energy numbers shown in figure 1, but making the instruction predicated with a false condition. The amount of energy reduction depends on the actual implementation. In the SC140e, all move instructions associated with the address-generate unit have a similar reduced incremental energy of approximately 22.2pJ. But for all instructions associated with arithmetic units like ADD, MAC, MPY etc., the incremental energy is actually reduced by a constant amount of around 10pJ per cycle on average. This is because these instructions actually go through the arithmetic units but only the destination register update is blocked. For generality, for each instruction we store an incremental and a reduced incremental energy number as shown in table II.

The energy for different stall types are calculated by creating a prolonged stall for several cycles and measuring the average power during that duration. If that is not possible, similar techniques as described above are used.

TABLE II
SAMPLE INCREMENTAL ENERGY TABLE

Instruction	Cycles	Incremental Energy (pJ)	Reduced Energy (pJ)
MAC	1	105.3	94.4
MPY	1	79.5	65.1
ADD	1	77.8	63.9
MOVE.W (post-inc load)	1	38.4	22.2
MOVE.W (pre-inc load)	2	40.9	22.2
MOVE.W (post-inc store)	1	42.3	22.1
BF (branch taken)	4	8.3	-
BF (branch not taken)	1	10.4	-

C. Inter-Execution-Set Energy

In most modern DSPs due to aggressive clock gating the variation and absolute value of controller energy for different pairs of functional instructions is limited. The energy value further reduces when a functional instruction changes to a NOP or prefix instruction or vice-versa. This is because the NOP or prefix instruction does not do anything after it is decoded. If a functional instruction like MAC changes to another functional instruction like MOVE in the same lane, n controller signals with its associated circuitry will deassert and a new set of m signals will assert. When MAC changes to a NOP, only n controller signals will deassert. This is assuming that the control signals are like one-hot with separate control lines for different functions. This assumption is true to some extent for most complex controllers. Hence the switching energy of a functional to a NOP or prefix instruction will be smaller. This argument differs from that proposed in the NOP model of inter-instruction energy [10]. Additionally if a NOP instruction changes to prefix or vice-versa, the inter-instruction energy is negligible.

The VLIW processor has an instruction register (IREG) with several lanes with corresponding instruction decoders on each lane. For computing the inter-execution set energy, we align the instructions of the execution sets in the IREG as they

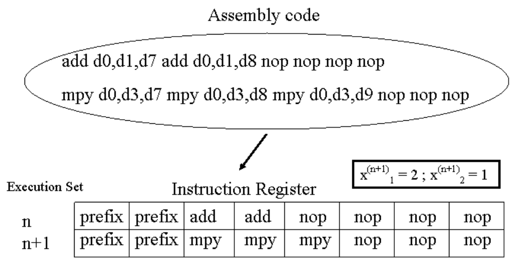
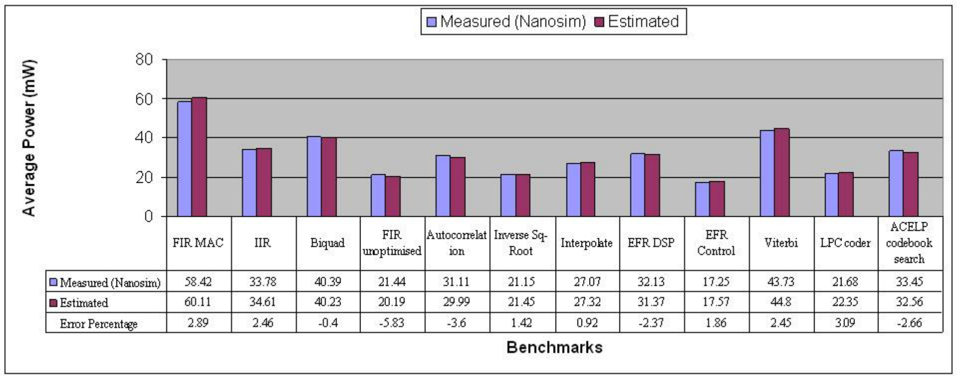

Assembly code
add d0,d1,d7 add d0,d1,d8 nop nop nop nop
mpy d0,d3,d7 mpy d0,d3,d8 mpy d0,d3,d9 nop nop nop
Execution Set
Instruction Register
$x^{(n+1)}_1 = 2$; $x^{(n+1)}_2 = 1$
n
n+1
prefix prefix add add nop nop nop nop
prefix prefix mpy mpy mpy nop nop nop

Fig. 2. E_{ies} calculation from assembly code

get decoded. Change in instructions in each lane is tracked to compute the variables x_1^i and x_2^i of equation 8. In figure 2, we show the computation of x_1^i and x_2^i corresponding to the execution set $(n+1)$. To calculate the coefficients, we create several code snippets with execution sets in hardware loops consisting of ADD, MPY, MAC, MOVE(different types), NOP, PREFIX, etc similar to that shown in figure 2. We use fixed maximal length execution sets to eliminate the constant factor k_0 from our initial analysis. We use exactly the same data values as used in incremental energy calculation. Then we calculate the difference between measured energy and that given by equation 3. This represents E_{ies}. We perform a regression analysis to model E_{ies} as a linear equation on the average values of x_1^i and x_2^i per execution set. The regression coefficient R^2 for the SC140e came out to be 0.92, justifying our assumptions. However a part of the controller is used to calculate the length and position of the current execution set. This effect is then introduced as a constant k_0. We argue that most VLIW processors will also have limited variation in energy due to the length variability effect. For fixed length processors we can omit this constant factor.

V. FUNCTIONAL SEPARABILITY

At the architectural level, the processor performs a set of well-defined functions. While building the model we make sure that we distinctly map the energy of each of these tasks into only one component of the model. We term this feature as "functional separability" of the core model that makes the whole energy model very accurate. This is the very reason that we built the model based on NOP and incremental energy values, used hardware loops with NOP word padding for incremental energy calculation, clearly separated data and control inter-instruction effects. For every high-level function we define two components of energy - fixed and variable. The fixed component is due to the function being performed by the same instruction. The variable component is due to the function being performed by different instructions. The table III points out the different high-level functions and their mapping to a single component in the energy equation. The incremental energy ΔE_k consumes the bulk of the energy in the dispatch and execution units, the NOP energy mainly accounts for idle state and the program address generation,



	FIR MAC	IIR	Biquad	FIR unoptimised	Autocorrelation	Inverse Sq-Root	Interpolate	EFR DSP	EFR Control	Viterbi	LPC coder	ACELP codebook search
Measured (Nanosim)	58.42	33.78	40.39	21.44	31.11	21.15	27.07	32.13	17.25	43.73	21.68	33.45
Estimated	60.11	34.61	40.23	20.19	29.99	21.45	27.32	31.37	17.57	44.8	22.35	32.56
Error Percentage	2.89	2.46	-0.4	-5.83	-3.6	1.42	0.92	-2.37	1.86	2.45	3.09	-2.66

Fig. 3. Power Estimation on Benchmarks @100MHz

whereas E_{ies} accounts for the switching part in program read, decode and dispatch.

TABLE III
MAPPING OF PROCESSOR FUNCTIONS

Function	Fixed	Variable
Program Address generation	E_{NOP}	E_{NOP}
Read program data from memory	E_{NOP}	E_{ies}
Decode & Dispatch of instruction	ΔE_k	E_{ies}
Data Address Generation	ΔE_k	ΔE_k
Arithmetic Computation	ΔE_k	ΔE_k

* E_{NOP} also models idle energy of the processor

VI. RESULTS AND CONCLUSION

We tested this model on the SC140e DSP core with a large suite of real-life DSP C benchmarks (with different compiler options) and assembly codes. In figure 3, we show a sample set of benchmark programs. The benchmarks contained both small and large DSP functions, several codecs and control codes. The maximum error reported over the entire benchmark suite is less than 6.0%. Higher error is primarily due to the functionality effect in incremental energy calculation deviating from the assumed 50%. The absolute average error is just 2.5% over a large number of DSP and control codes.

Though there is lot of proven work in the area of single-issue processors, comprehensive energy models for multiple issue machines like VLIW have not been thorughly investigated. The previous works on energy estimation of VLIW processors have several deficiencies. In [6], the power of an execution set of (ADD MAC) is approximated as power of (ADD NOP) added to (NOP MAC). This adds up the NOP energy twice in an execution set. This is also evident in the estimation being over 15% higher than the measured power. The model in [7] maintains a table of additional energy contributions over the NOP energy for each pair of instructions. This has a characterization complexity of $O(N^2)$, where N is the total number of instructions of the processor. This is practically not feasible for commercial processors with over hundred instructions. Moreover clustering of instructions here will lead to higher inaccuracy because the base and inter-instruction components are merged together. Clustering of instructions works better for inter-instruction effects as it has relatively smaller absolute value and variation compared to base energy. When pipeline stalls occur, the absolute maximum error of this model can be 20%. In this paper, we presented a novel and comprehensive energy model for VLIW DSP cores. The model is equally effective in estimating both DSP and control kernels with predicated execution and pipeline stalls. Due to a simple yet accurate inter-execution set energy model, we are able to reduce the model complexity from $O(N^2)$ to $O(N)$. We integrated the energy model into the application software simulation environment. This simulator has several ISS corresponding to various VLIW processors of the StarCore family. The energy model is implemented as a function library that interacts with a particular ISS based on the processor selected. The energy tables are stored in separate data libraries corresponding to each processor and each technology node. In its default form, the power estimation tool named *JouleQuest* reports the energy and power (in the core and memory sub-system) of the application code with a function-wise breakup.

REFERENCES

[1] C. Tsui, M. Pedram, and A. Despain, "Efficient estimation of dynamic power consumption under a real delay model," in *Proceedings of International Conference on Computer Aided Design (ICCAD)*, 1993.
[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2000.
[3] P. E. Landman and J. M. Rabaey, "Activity-sensitive architectural power analysis," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 15(6), June 1996.
[4] V. Tiwari, S. Malik, and A. Wolfe, "Power analysis of embedded software: A first step towards software power minimization," *IEEE Transactions on VLSI Systems*, vol. 2(4), December 1994.
[5] M. Lee, V. Tiwari, S. Malik, and M. Fujita, "Power analysis and minimization techniques for embedded DSP software," *IEEE Transactions on VLSI Systems*, vol. 5(1), March 1997.
[6] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, "Instruction-level power estimation for embedded VLIW cores," in *Proceedings of CODES*, 2000.
[7] Sami, Sciuto, Silvano, and Zaccaria, "An instruction-level energy model for embedded VLIW architectures," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 21(9), September 2002.
[8] *Nanosim User Guide*, Synopsys Inc., December 2004.
[9] *SC140e DSP Reference Manual*, Freescale Semiconductors, Jan 2004.
[10] B. Klass, D. Thomas, H. Schmit, and D. Nagle, "Modeling inter-instruction energy effects in a digital signal processor," in *Proceedings of International Symposium on Computer Architecture (ISCA)*, 1998.