

An Approach to Switching Activity Consideration During High-Level, Low Power Design Space Exploration ¹

Russell Henning and Chaitali Chakrabarti

Department of Electrical Engineering
Arizona State University
Tempe, AZ 85287-5706
Ph: (480) 965-9516
Fax: (480) 965-8325
rhenning@asu.edu, chaitali@asu.edu

Abstract

A novel approach is introduced that exploits characteristics of fixed-point, two's complement data in order to reduce power consumption related to switching activity. This approach is based on an intuitive switching activity model that captures the most essential data characteristics with statistical parameters. The approach is embodied in a heuristic that uses the model to systematically reduce switching activity of interconnect between datapath units. The perspective provided by the model and heuristic allows efficient and intuitive high-level design space exploration. This approach is demonstrated through an example of high-level design space exploration for a low power processor dedicated to implementing the IS-54 vector-sum excited linear predictive (VSELP) speech codec. Application of the heuristic results in up to 56% activity reduction at high energy locations in the datapath and estimated processor power reduction of about 15% on average during encoding compared to an obvious implementation.

¹This work was supported in part by a Motorola SABA grant and in part by the NSF/SIUCRC Center for Low Power Electronics.

1 Introduction

In early development of an application specific VLSI system, power reduction opportunities abound. These include opportunities at the algorithm level, architecture level, circuit level, and process level [1], [2]. However, one opportunity that is seldom adequately considered is switching activity-related power reduction. Activity-related power reduction potential must be exploited during high-level design space exploration, or else it will likely be lost as the design space is narrowed down. The lack of an intuitive method for accomplishing this task is a substantial barrier to this power reduction technique's acceptance.

Existing methods that take switching activity into account, generate an entire low power data path (functional units, registers, interconnect, buses, etc.) for a given CDFG [3], [4], [5]. However, a more global approach is needed that first identifies where data characteristics have the most potential for exploitation in an entire application and then focuses design exploration on this potential.

To better aid designers in exploiting activity-related power reduction opportunities, this paper first introduces a novel activity model based on four statistical parameters that are related to essential characteristics of fixed-point, two's complement data. Then, a heuristic is presented for intuitively applying this model to achieve significant power reduction during high-level design space exploration. Even though the methods used for reducing power consumption are not new, this new view allows opportunities to be taken advantage of in a more efficient and understandable manner while reducing the chance that important design alternatives will be missed.

The framework under which the model and heuristic are demonstrated is high-level design space exploration for an IS-54 vector-sum excited linear predictive (VSELP) speech codec [6]. The design space exploration process illustrated here considers constraints such as real-time encoding capability, acceptable speech quality, reasonable design and implementation complexity, low area, and, more importantly, low power. The first step is algorithm design exploration where power is saved by choosing the way in which data will be represented. The next step is architecture design exploration where a datapath consisting of two pipelined multipliers and two adders is chosen to speed execution of the algorithm, allowing power consumption to be reduced through supply voltage reduction. The third step is scheduling and allocation exploration where activity

related power reduction is achieved by exploiting data characteristics. The heuristic and switching activity model are used to guide this step. Due to application of the heuristic alone, it is estimated that about 15% average power reduction can be achieved during encoding compared to an obvious implementation of the VSELP codec. Activity is reduced by up to 56% at high energy locations in the datapath.

Thus, there are four main contributions of this paper:

- An intuitive new model that relates data characteristics to switching activity of datapath interconnect is presented.
- A heuristic that uses the model to significantly reduce activity-related power consumption is introduced and demonstrated.
- This new method is integrated with other power saving techniques into a high-level design space exploration example for a VSELP speech codec.
- Novel, low power implementations for the most computationally intensive modules in the encoder are found and described.

The rest of the paper is organized as follows. Section 2 describes the new activity model and how it is applied through the heuristic to accomplish high-level, low power design space exploration in an efficient and intuitive manner. Section 3 demonstrates application of the heuristic with a VSELP speech codec design exploration example. Section 4 concludes the paper.

2 Power Reduction Approach

In a typical static CMOS datapath, the nodes where transition activity can most affect power consumption are inputs to functional units, inputs to memory units, and the pathways that carry data between these units (datapath interconnect). Datapath interconnect can significantly affect overall power consumption when it has large capacitance. Similarly, inputs to functional or memory units can have large effective capacitance due to the dependence of node activity inside a unit on activity at its inputs.

For datapath interconnect, it is usually sufficient to reduce average activity for each bit of a multi-bit path to reduce its power consumption. While the relationship between switching activity

at inputs of a functional unit or memory unit and switching activity of nodes inside the unit is not as easily understood, there are methods for estimating power inside units having specific structures based on activity of input/output nodes [7], [8]. Power can be significantly reduced inside a functional or memory unit by reducing activity at particular inputs according to this type of characterization.

However, it is not enough to be able to estimate the effect activity of a particular design choice has on power consumption. This is because, for a typical sized design, it is very computationally expensive to test all design possibilities to find the one with lowest power consumption. Therefore, a good strategy is required to quickly and accurately narrow down design choices. The foundation of our strategy is a model that relates data characteristics to activity such that the causes of activity can be well understood with little effort.

Existing high-level design techniques that consciously try to reduce transition activity do not consider data characteristics. The most basic ones simulate all possible scheduling and allocation choices [3], [9]. Thus, these methods are too computationally expensive for narrowing down a typical large design space. Others attempt to reduce the complexity by looking at a few choices, one control step at a time, thereby potentially excluding good designs [10], [11].

Existing work in data characterization, for the most part, has been geared towards illustrating and addressing special cases [12]-[16]. There are only a couple models that practically relate data characteristics to transition activity in general. Others, discussed in [17]-[19], are limited by difficulty of use or constraining assumptions.

One of the more practical models is the Dual Bit Type (DBT) model [7],[14] which uses probabilistic parameters to characterize the data-related activity. This model was developed to determine ranges for two types of activity that can be used to estimate power consumption inside certain functional units. The main benefit of the DBT model is its ability to parameterize power estimates using fewer parameters than activity estimates for each bit of a multi-bit node. However, while the utility of this model has been successfully demonstrated for high-level power estimation, it does not appear to be well suited for use in making high-level design decisions for a number of reasons [20]. Most importantly, characterizing data in terms of probabilistic parameters (mean, variance, and correlation coefficient) clouds intuitive relationships between data and activity, making design

decisions difficult. The DBT model also has trouble capturing relationships for certain types of data like slowly varying, nonstationary, or truncated data.

A new model based on statistical parameters related to practical methods for reducing activity-related power consumption addresses these problems and appears to be more suitable for high-level design space exploration than the DBT model. This method was originally proposed by the authors in [21]. At the word-level, scaled, truncated, and/or slowly varying data are easily related to the model's parameters, which are in turn easily related to average activity at the bit-level through the model. As a result, this new model allows high-level design decisions to be made based on the intuitive relationship between its parameters and activity estimates, rather than more accurate but more difficult to find power estimates provided by the DBT model. The new parameters can in many cases be estimated from knowledge of the algorithm. Additionally, slowly varying, nonstationary, or truncated data are characterized better with this model than the DBT model. The reader is referred to [20] for more detailed comparison of the models. In the rest of this section, this new model is described, and a heuristic for applying it to efficiently reduce significant power consumption is introduced.

2.1 Switching Activity Model

In [21] three main types of fixed-point, two's complement data are identified that can significantly affect $0 \rightarrow 1$ transition activity of interconnect between datapath functional units. The first type is characterized by a significant amount of sign extension evident in consecutive words appearing at a multi-bit node. The second type tends to result when consecutive words differ by a very small magnitude. The third type has consecutive words that are significantly truncated.

To understand the effect these types of data can have on power consumption, [21] proposes the use of four basic parameters. These parameters are very intuitive, because they are derived from the simple case of a single word transition at a multi-bit node. The simplicity of these parameters make relationships between input and output parameters for functional units easier to understand. The single-transition versions of these parameters will be developed first, followed by multiple transition parameters. A model that relates these parameters to activity will then be given.

To derive the first single-transition parameter, sign bits must be defined. Sign bits are consecutive bits following the most significant bit of a fixed-point, two's complement value that are the

	Binary value	Decimal Value
Previous value	0000011010010000	0.05126953125
Current value	0000011011001000	0.052978515625
	<div style="display: flex; justify-content: space-around; width: 100%;"> </div> <div style="display: flex; justify-content: space-around; width: 100%; font-size: small;"> IS CMD IT </div>	

(a)

	Binary value	Decimal Value
Previous value	0000011001010000	0.04931640625
Current value	1111001100110000	-0.10009765625
	<div style="display: flex; justify-content: space-around; width: 100%;"> </div> <div style="display: flex; justify-content: space-around; width: 100%; font-size: small;"> IS IT </div>	

(b)

Figure 1: Transition between (a) two positive 16-bit values and (b) a positive and a negative 16-bit value.

same as the most significant bit. For example, the five most significant bits (MSBs) of the positive 16-bit number 0000011010010000 are sign bits, as are the four MSBs of the negative 16-bit number 1111001100110000.

The first single-transition parameter is the number of intersecting sign (IS) bits between the two values involved in the transition. This parameter is the number of consecutive bits of a multi-bit node, beginning with the most significant bit, that make the same transition as the most significant bit. Two examples of single transitions are shown in Figure 1. Each transition shows consecutive binary values occurring at a 16-bit node driven by glitch-free static CMOS circuitry. The binary numbers are fixed-point, two's complement representations of decimal values between -1 and 1 (also shown in Figure 1). For the transition in Figure 1a, the number of IS bits is five, while in Figure 1b, it is four.

To characterize the effect of significant sign bit-related activity in a single transition, identification of the sign transition is the next step. As can be seen in Figure 1b, if the sign transition is $+ \rightarrow -$, all the IS bits will consume power in that transition. If the single transition is anything else, they will not consume power. So, the second single-transition parameter is just a flag that is 1 for a $+ \rightarrow -$ transition and 0 for any other transition.

The third single-transition parameter deals with truncation. For a single binary number, define truncation bits as the consecutive bits beginning with the least significant bit (LSB) that are zero. For instance, the number of truncation bits in Figure 1a is four in the previous value and is three in the current value. Similar to sign bits, the single-transition parameter associated with truncation bits is the number of intersecting truncation (IT) bits between consecutive node values. This parameter is defined as the minimum of the number of truncation bits in the two values involved in the transition. In Figure 1a, the number of IT bits is three. In Figure 1b, it is four. The property

of IT bits that relates them to significant power reduction is they all make $0 \rightarrow 0$ transitions. They never make power consuming $0 \rightarrow 1$ transitions.

The final single-transition parameter attempts to characterize data bits, which include all other bits of a value that are not sign or truncation bits. In Figure 1a, the previous value has seven data bits and the current value has eight data bits. The single-transition parameter associated with data bits is the number of consecutive matching data (CMD) bits between consecutive node values. This parameter is defined as the number of consecutive data bits that match in value and position between the two values involved in a transition beginning with the most significant data bit of both values. In Figure 1a, the number of CMD bits is four, whereas in Figure 1b, it is zero. The property of CMD bits that relates them to significant power reduction is they all make $0 \rightarrow 0$ or $1 \rightarrow 1$ transitions. They never make power consuming $0 \rightarrow 1$ transitions.

With all the single transition parameters defined, multiple transition parameters are found by simply averaging each of the single-transition parameters over multiple transitions. The resulting parameters are denoted α_{OS} , n_{IS} , n_{CMD} , and n_{IT} . α_{OS} is the fraction of transitions that are opposite sign ($+ \rightarrow -$ or $- \rightarrow +$) transitions. Thus α_{OS} is approximately two times the single transition parameter averaged over all transitions. n_{IS} is the average number of IS bits per transition. n_{CMD} is the average number of CMD bits per transition. n_{IT} is the average number of IT bits per transition.

Because it is known approximately what fraction of the time each classification of bits makes a $0 \rightarrow 1$ transition, these parameters can be used to estimate $\alpha_{0 \rightarrow 1}$, the average $0 \rightarrow 1$ transition activity per bit on an N -bit node. (In terms of Average Hamming Distance (AHD), $\alpha_{0 \rightarrow 1} \approx \frac{1}{2N}$ AHD.) IS bits make $0 \rightarrow 1$ transitions approximately $\frac{1}{2}\alpha_{OS}$ of the time. CMD and IT bits never make $0 \rightarrow 1$ transitions. It can be reasonably assumed that most of the remaining bits make $0 \rightarrow 1$ transitions $\frac{1}{4}$ of the time, because the bit values involved in the transitions are completely independent. Since the average number of these remaining bits is $N - n_{IS} - n_{CMD} - n_{IT}$, $\alpha_{0 \rightarrow 1}$ can be modeled as

$$\hat{\alpha}_{0 \rightarrow 1} = \frac{1}{N} \left[\frac{1}{4} (N - n_{IS} - n_{CMD} - n_{IT}) + \frac{1}{2} \alpha_{OS} n_{IS} \right] + \frac{error}{N}. \quad (1)$$

The error term in this equation is usually due to a breakdown of the assumption that the unclassified bits encounter equally likely transitions. This typically occurs because bits that should

be unclassified are classified as IS, CMD, or IT bits during transitions where they look like IS, CMD, or IT bits. The error term can be estimated depending on the particular data stream being characterized. However, a good general estimate has been found to be 0.29.

In what follows, the way in which these parameters can be used to reduce switching activity will be illustrated with examples. First, special attention will be paid to sign bits, because they often require the most creative scheduling and allocation to exploit. Then, after introducing a new design heuristic based on this model, examples will be used to show how all the exploitable types of data presented in this section can be taken advantage of in the VSELP codec design.

2.2 Power Reduction through Exploitation of Sign Bit Characteristics

For simplicity, assume n_{CMD} and n_{IT} are not significant. We can then come up with a model that relates the parameters that characterize sign bit activity, α_{OS} and n_{IS} , to the average $0 \rightarrow 1$ transition activity per bit of an N-bit node:

$$\hat{\alpha}_{0 \rightarrow 1} = \frac{1}{N} \left[\frac{1}{4}(N - n_{IS}) + \frac{1}{2} \alpha_{OS} n_{IS} \right] \quad (2)$$

Figure 2 gives a graphical interpretation of this model. It shows that activity is directly proportional to α_{OS} . However, the slope of this relationship is proportional to n_{IS} . If n_{IS} is very low, α_{OS} has almost no effect on activity. By the same respect, if $\alpha_{OS} \approx 0.5$, the value of n_{IS} has almost no effect on power consumption. In both of these cases, $\alpha_{0 \rightarrow 1} \approx 0.25$, which we call the nominal activity level.

When data is identified that could potentially have large n_{IS} , judicious scheduling and allocation can result in significant activity reductions. For example, take three vectors, L[], H[], and N[], of 16-bit fixed-point, two's complement data, each with $n_{IS} \approx \frac{N}{2}$. No other significant relationships exist between consecutive words in these arrays except that L[] has low α_{OS} , H[] has high α_{OS} , and N[] has $\alpha_{OS} \approx 0.5$ (which corresponds to nominal $\alpha_{0 \rightarrow 1}$). In addition, assume that the same bus has been allocated to all of the data and no significant relationship exists between the arrays L[], H[], and N[].

Examples of some interesting scheduling possibilities along with the resulting $\alpha_{0 \rightarrow 1}$ for the bus are shown in Table 1. Cases 1-3 show the effect of the different levels of α_{OS} on activity between consecutive array values. Cases 4 and 5 show that by breaking an array of high α_{OS} values into an

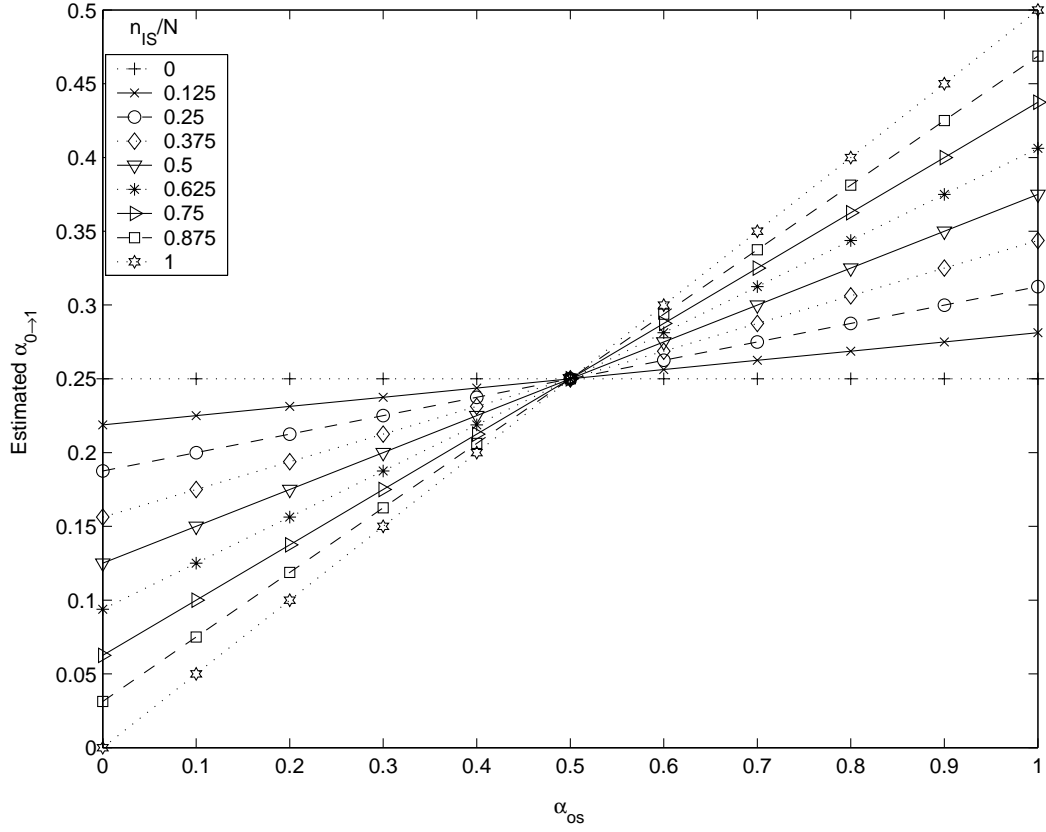


Figure 2: Estimated activity vs. opposite sign transition activity as the average number of intersecting sign bits varies.

Case	Scheduled Order for a Single 16-bit Bus	$\alpha_{0 \rightarrow 1}$
1	N[0],N[1],N[2], ...	Nominal
2	L[0],L[1],L[2], ...	Low
3	H[0],H[1],H[2], ...	High
4	H[0],H[2],H[4], ...	Low
5	H[1],H[3],H[5], ...	Low
6	H[0],N[0],H[1],N[1],H[2],N[2], ...	Nominal
7	L[0],H[0],L[1],H[1],L[2],H[2], ...	Nominal
8	L[0],L[1],H[0],H[1],L[2],L[3],H[2],H[3],L[4],L[5],H[4],H[5], ...	Nominal
9	L[0],H[0],L[1],H[2],L[2],H[4], ...	High or low over extended periods
10	L[0],L[1],H[0],H[2],L[2],L[3],H[4],H[6],L[4],L[5],H[8],H[10], ...	Low or nominal over extended periods
11	L[0],L[1],H[0],H[2],L[2],L[3],H[1],H[3],L[4],L[5],H[4],H[6], ...	Low but greater than Case 2

Table 1: Interesting scheduling options for three data streams allocated to the same bus

array of even elements and an array of odd elements, the new arrays will actually have low α_{OS} , which can result in low $\alpha_{0 \rightarrow 1}$.

Case 6 in Table 1 shows that if nominal α_{OS} data is scheduled between high α_{OS} data, $\alpha_{0 \rightarrow 1}$ will become nominal, or approximately 0.25. This schedule can be an effective way of reducing activity associated with a high α_{OS} array, if the array cannot be split into its even and odd elements. Cases 7 and 8, on the other hand, show less desirable ways of reducing high α_{OS} , because the potential for lowering $\alpha_{0 \rightarrow 1}$ even more by exploiting the low α_{OS} of the L[] array is lost.

The last three cases are interesting because they deal with two unrelated sets containing low α_{OS} data. If the designer has a good idea of when low α_{OS} is due to $+\rightarrow+$ transitions in both streams being considered and when it is due to $-\rightarrow-$ transitions, finding a low activity schedule is straightforward. If not, interleaving values from the two different sets can lead to high or low α_{OS} over extended periods of time.

Obviously, Table 1 does not represent all possible scheduling possibilities. However, by characterizing data in terms of the activity model parameters, it is possible to find a low power design without directly considering all possibilities.

2.3 Heuristic

Up to now, general information about how data affects activity in the data path has been given. Since this information is in a form that is very intuitive, activity at datapath interconnect can be predicted and understood with great ease in many cases. As a result, this new type of information can be used as a tool to help guide high-level design space exploration. A general heuristic is given below for using the model to reduce significant power consumption through switching activity reduction.

1. Determine approximately how much and where in the architecture and algorithm activity-related power reduction would be significant to the design
2. Do transformations where prudent
 - (a) Consider independently executable blocks of code as one
 - (b) Perform loop unrolling
3. Identify significant amounts of three key types of data in the algorithm being implemented
 - (a) Data with significant numbers of truncation bits
 - (b) Data with significant numbers of sign bits
 - (c) Data with slowly varying values
4. Identify significance of following sources of switching reduction and interactions between them (i.e. how certain options result in or take away other opportunities), but beware of control overhead and complexity of design/implementation associated with complicated scheduling and allocation
 - (a) Scheduling and allocation options that result in the highest value for n_{IT} at the highest capacitance nodes
 - (b) Scheduling and allocation options that result in the highest value for n_{IS} when α_{OS} is low at the highest capacitance nodes

- (c) Scheduling and allocation options that result in the lowest value for n_{IS} when α_{OS} is high at the highest capacitance nodes
 - (d) Scheduling and allocation options that result in the highest value for n_{CMD} at the highest capacitance nodes
5. Choose best design in terms of expected energy reduction, low control overhead, and low complexity of design/implementation
 6. Validate that the final design choice lowers activity as expected

The efficiency of this heuristic results, first of all, from determining what is meant by significant switching activity reduction. Activity reduction may only be significant at high capacitance nodes such as memory buses and inputs to certain functional units. In addition, it is important to know how much activity reduction at each of these nodes is significant. This threshold can vary depending on the constraints of the design.

Another key to the heuristic's efficiency is that the three types of data that typically result in significant activity reduction are identified in significant amounts (Step 3). This step quickly narrows down the design space to regions where significant opportunities have the best chance to exist. Both the quantity of data flowing through a high capacitance node and the ability to affect the activity of that data are important. Even though power reduction through activity reduction is an additive process, it is usually worth attempting to exploit the largest groups of data first to reach power reduction goals quickly.

Finally, scheduling and allocation is efficient in this heuristic because only four parameters must be considered that are directly related to the data characteristics identified in Step 3. It is recommended that scheduling and allocation options be identified (if they exist) in the order shown in step 4, because techniques higher in this list tend to have greater potential to propagate reduced switching activity opportunities to other parts of the design. Opportunities to propagate activity reduction should be identified and exploited wherever feasible as part of this process. However, control overhead and complexity of design/implementation must be considered before choosing any complicated scheduling and allocation option.

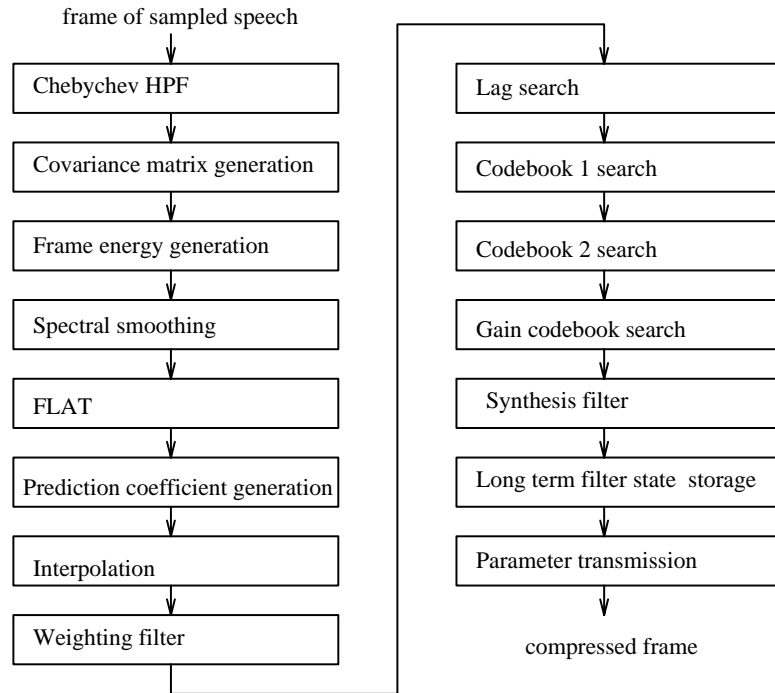


Figure 3: Block diagram of IS-54 VSELP Speech Encoder

3 VSELP Codec Example

Next, an example of high-level design space exploration will be presented for the IS-54 VSELP speech codec. All three steps of the exploration process are considered in order to reach the point where the new activity-related power reduction approach can be applied. First, the VSELP speech encoding algorithm is briefly introduced, and data representations for the algorithm are selected to reduce power consumption while sacrificing minimal speech quality. Then, design exploration for the processor architecture is described. Finally, the new heuristic is applied to accomplish low power scheduling and allocation in an efficient and intuitive manner. The importance of Step 2 of the heuristic, i.e. transformations, is specifically addressed during this final step of the exploration process.

3.1 Algorithm Design Exploration

Figure 3 shows a block diagram of the IS-54 VSELP speech encoding algorithm. It uses an analysis by synthesis process to do parametric, block coding of 8 kHz sampled, 8 bit speech samples at 8 kb/s, a compression ratio of 8:1. Blocks are frames of 160 samples, so frames must be compressed

at a rate of 20 ms/frame to keep up with real-time speech sample generation. For background on speech coding, refer to [22]. A description of the entire IS-54 VSELP speech codec is given in [23].

Algorithm design exploration is the first step in the high-level design space exploration process. Precision and range of data types in the VSELP algorithm can be reduced in this step. There are several benefits of doing this. First, computations are simplified, in general, so that less complex functional units that consume less power and execute faster can be used. Second, storage and interconnect bandwidth requirements are reduced for smaller data types, resulting in less area and power consumed by memory units and datapath interconnect. Finally, as discussed previously, certain data characteristics can be exploited to reduce switching during scheduling and allocation if lower precision data is represented by fixed-point, two's complement data types.

Based on speech fidelity measurements, the data types and operations chosen for this design of the VSELP encoder are summarized in Table 2. This algorithm design for the VSELP speech encoder calls mostly for 16 bit fixed point data types: 16x16 bit fixed point multiplications and between 16 and 32 bit fixed point accumulation. Accumulation must be done in 16, 24 and 32 bit precision to maintain fidelity in the processed speech signal. Floating point values are represented with a data type having a 16 bit mantissa and 16 bit exponent to allow implementation on a 16 bit data path. High precision, 31X31 bit multiplications are implemented with 16X16 bit multiplications and control.

The basic requirements of the data path and controller are defined by the algorithm design. While the design calls for mostly 16 bit fixed point multiplications and 16 to 32 bit fixed point additions, it must also support 31 bit fixed point multiplication, 16 bit fixed point square root and division, 32 bit floating point data types and operations, as well as scaling of fixed point values. Such a design can essentially be implemented with a 16 bit data path consisting of a 16X16 bit fixed point multiplier, a 32 bit fixed point adder, and a 32 bit shifter. Increased control complexity is required to execute operations of greater complexity (e.g. 32 bit floating point additions) on this data path. However, the low frequency with which such operations must be executed justifies microcoded implementation.

Data memory requirements are also provided by the algorithm design. All data can be stored in 16 bit memory locations with 32 bit fixed or floating point values stored in two 16 bit memory

Module	Operations Used
Chebyshev HPF	16X31 and 31X31 bit fixed point multiplication, 32 bit fixed point accumulation, fixed point value scaling
Covariance matrix generation	16X16 bit fixed point multiplication, 32 bit fixed point accumulation, fixed point value scaling
Frame energy generation	16 bit fixed point square root, fixed point value scaling
Spectral smoothing	16X16 bit fixed point multiplication, fixed point value scaling
FLAT	16X16 bit fixed point multiplication, 16 bit fixed point division, 32 bit fixed point accumulation
Prediction coefficient generation	16X16 bit fixed point multiplication, 32 bit fixed point accumulation, fixed point value scaling
Interpolation	16X16 bit fixed point multiplication, 16 bit fixed point division, 16 bit fixed point square root, 16 bit fixed point addition, 32 bit fixed point accumulation, fixed point value scaling
Weighting filter	16X16 bit fixed point multiplication, 32 bit fixed point accumulation
Lag search	16X16 bit fixed point multiplication, 16 bit fixed point accumulation, 32 bit fixed point accumulation
Codebook 1 search	16X16 bit fixed point multiplication, 16 bit fixed point division, 16 bit fixed point accumulation, 32 bit fixed point accumulation, fixed point value scaling
Codebook 2 search	16X16 bit fixed point multiplication, 16 bit fixed point division, 16 bit fixed point accumulation, 32 bit fixed point accumulation, fixed point value scaling
Gain codebook search	16X16 bit fixed point multiplication, 32X32 bit floating point multiplication, 16 bit floating point division, 16 bit floating point square root, 32 bit fixed point accumulation, 32 bit floating point accumulation
Synthesis filter	16X16 bit fixed point multiplication, 32 bit fixed point accumulation
Long term filter state storage	Memory accesses only
Parameter transmission	16X16 bit fixed point multiplication, 32X32 bit floating point multiplication, 32 bit fixed point accumulation, fixed point value scaling

Table 2: Operations used in algorithm design broken down module by module

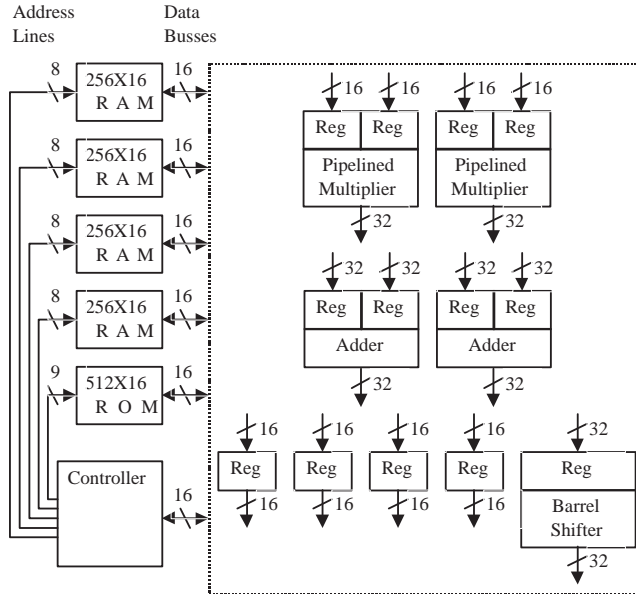


Figure 4: Block diagram of the processor

locations. Total RAM required is 1024 words based on memory access calculations in [24]. ROM is estimated to be 512 words based on fixed data such as quantization tables. With memory, datapath, and control specifications in place, the next step in the high-level design space exploration process, architecture design exploration, can be taken.

3.2 Architecture Design Exploration

The specifications from algorithm design exploration are used to select a low power data path and method of control that achieve the required throughput for real-time use. Supply voltage reduction is the focal point of power reduction in this step. (Much of the low voltage, data path design that follows is based on [24].) However, the architecture design also affects achievable activity-related power reduction.

3.2.1 Supply Voltage Reduction

Data path design for the VSELP speech encoder is most influenced by the fact that multiply-accumulate operations dominate the algorithm. A 16X16 bit Baugh-Wooley array multiplier and 32 bit carry look-ahead adder have been selected for the 16 bit data path specified by the algorithm design. It is assumed that the 16X16 bit Baugh-Wooley multiplier has a latency less than 60 ns and the 32 bit carry look-ahead adder has a latency less than 15 ns for 5 V supply voltage and

Module	16x16 bit multiplications per frame	Execution time per frame (ms)
Chebyshev HPF	4160	0.3588
Covariance matrix generation	1870	0.1613
Frame energy generation	0	0.0000
Spectral smoothing	66	0.0057
FLAT	2574	0.2220
Prediction coefficient generation	55	0.0047
Interpolation	471	0.0406
Weighting filter	4800	0.4140
Lag search	46400	4.0020
Codebook 1 search	27280	2.3529
Codebook 2 search	28560	2.4633
Gain codebook search	10172	0.8773
Synthesis filter	30400	2.6220
Long term filter state storage	0	0.0000
Parameter transmission	2092	0.1804
All modules	158900	13.7051

Table 3: Time estimate and number of 16x16 bit multiplications used per frame in IS-54 VSELP speech encoder design broken down by module

0.7 V threshold voltage. However, the targets for this design are a 1.1 V supply voltage and a 0.5 V threshold voltage. For such a reduction in supply voltage and threshold voltage, these latency limits scale to approximately 688 ns for the multiplier and 172 ns for the adder [1]. From the algorithm design, it is approximated that at most 158,900, 16X16 bit fixed point multiplications must be performed to encode one frame of speech. The number of multiplications broken down by module is shown in Table 2. Therefore, with $V_{dd} = 1.1V$, multiplier throughput must be increased by more than a factor of five for the algorithm to execute within the 20 ms per frame requirement of IS-54.

Because the structure of the Baugh-Wooley multiplier allows it to be pipelined to four levels of approximately equal latencies, it is possible to make its pipeline latency equivalent to the latency of an adder, 172 ns. While this feature enables the throughput of a multiplier to be increased by a factor of 4, including two multipliers pipelined to four levels increases throughput by a factor of 8. This speed-up should provide adequate slack for real-time operation, since algorithm dependencies prevent all 15,900 multiplications from being calculated with maximum efficiency.

Finally, additions must be executed quickly enough during multiply-accumulate computations to allow the multipliers to execute near full capacity. The majority of multiply-accumulate functions require 24 or 32 bit accumulation which is handled with two 32-bit adders in the data path, one for each multiplier. In addition to high throughput, multiple accumulation paths also allow activity to be better reduced at adder inputs, as will be seen in the scheduling and allocation examples.

3.2.2 Additional Features of the Data Path

A block diagram of the VSELP speech codec design is shown in Figure 2. Interconnect has not been completely specified yet, because it should be used conservatively based on the needs of scheduling and allocation which are identified later in the design process. This architecture design provides power savings in a number of ways in addition to increased throughput of multiplication and addition operations.

By supporting independent buses that connect multiple memory blocks to the data path, multiple reads and writes can be scheduled during the same clock cycle. In addition, data characteristics can be exploited to reduce switching on memory buses during scheduling of memory reads and writes.

The use of a power hungry register file has been avoided by allowing operands to be loaded to functional unit inputs directly from memory and functional unit outputs to be stored directly to memory. Statically scheduled bypassing even allows functional unit outputs to be routed to functional unit inputs during multiply accumulate operations. The registers that would normally be found in a register file are instead distributed over the data path. Only four general purpose registers are included in the data path. The registers at the inputs of the functional units provide more register storage and a method of disallowing switching in idle data path elements.

Registers at the inputs of adders and multipliers as well as the pipeline registers in the multipliers have a clock signal input with a period of 172 ns. The barrel shifter, memory accesses, and register accesses are all assumed to take less than 86 ns to execute at a supply voltage of 1.1 V. Therefore, another clock signal with a period of 86 ns is supplied for these functions to take advantage of their speed.

Data path Component	Area estimate (sq mm)	Number	Total area (sq mm)
Pipelined Multiplier	0.242791155	2	0.48558231
CLA	0.0833247	2	0.1666494
Barrel Shifter	0.11243988	1	0.11243988
256X16 bit Memory	0.56538519	6	3.3923111
Entire Data Path			4.15698269

Table 4: Area estimates for the data path

3.2.3 Control

A VLIW-based control scheme is recommended for the architecture design to support parallel execution of instructions in a power conscious manner. The reason for this recommendation is complex scheduling and allocation (which exploit parallelism) can be done at compile time rather than at run time when they can cost significant power. The functional units, namely, the multipliers, adders, registers, and barrel shifter are controlled by dedicated instruction streams that are derived from one very long instruction word stream. The instruction word length has an upper bound of 184 bits. This figure is derived assuming all data path outputs and inputs are allowed to connect with one another and all control signals are stored rather than generated at run time. However, as mentioned previously, conservation of connections should be sought unless there is significant justification for not doing so. VLIW instructions can also be compressed to reduce program memory requirements and then decompressed during decoding.

3.2.4 Execution-Time, Area, and Power Estimates

Given the architecture design shown in Figure 2, the time for a frame of speech to be encoded by the VSELP algorithm is broken down by module in Table 2. This table shows that the encoder architecture is capable of processing speech frames in about 13.7 ms, which is well within the 20 ms requirement of the standard. The area for the data path of the processor is estimated to be 4.16 mm^2 for Motorola's 0.5 micron, 5 metal technology. This estimate results from the data shown in Table 3. Assuming control and interconnect require an additional 30% of the data path area, the processor area is estimated to be approximately 5.41 mm^2 .

Data Path Component	Power Consumption (μW for $\alpha_{0 \rightarrow 1} = 25\%$)	Number	% time active	Overall Consumption (μW)
Pipelined Multiplier	5777.22	2	100	11554.44
CLA	1321.60	2	100	2643.20
Barrel Shifter	969.60	1	25	242.4
Register	257.40	16	75	3088.80
10:1, 16 bit Mux	451.45	12	100	5417.40
4:1, 16 bit Mux	180.58	4	100	722.32
256X16 bit Memory	5352.10	6	33	10597.16
Entire Data Path				34265.72

Table 5: Power estimates for the datapath

The data path power estimate is obtained by estimating the power consumed by the individual components in the data path. Table 4 shows the estimate for the power consumption of each component, the number of components in the data path, and the percentage of execution time the components are active while the processor is encoding a single speech frame. The value for each component includes a power estimate for the interconnect leading to the inputs of the component. This additional power is most significant for the memory modules, where 50% of power consumed is due to the memory buses. A conservative transition activity factor of 25% throughout the data path is assumed (this corresponds to all possible word transitions being equally likely). As will be discussed further shortly, the VSELP data path design supports scheduling and allocation that can exploit the data characteristics to reduce activity well below 25%.

Though power estimation is being done for designs where control power is significant [16], the VLIW controller is not well defined at this point. As a rough estimate, the assumption has been made that the controller requires an additional 20% of the power consumed in the data path. This assumption gives an average power consumption of about 41 mW while the encoder is executing.

In the processor data path, switching costs are most substantial for multipliers and memory. Based on Landman’s total effective capacitance model and example of capacitive coefficients for an array multiplier [7], it is reasonable to estimate that 80% of the multiplier power consumption estimate in Table 4 is directly proportional to activity. For example, if $\alpha_{0 \rightarrow 1}$ for multiplier inputs is 10% below 0.25, multiplier power will be reduced by 8% on average. As for the memory itself,

reducing accesses (a special case of activity reduction) saves a great deal of power. However, as mentioned previously, 50% of the memory power consumption estimate in Table 4 is due to memory bus capacitance being switched at $\alpha_{0 \rightarrow 1} = 25\%$. Thus, a 10% reduction in this activity will result in a 5% reduction in power. Next, it is shown how average power consumption of these components can be significantly reduced through application of the activity-related power reduction heuristic to scheduling and allocation exploration.

3.3 Scheduling and Allocation Exploration

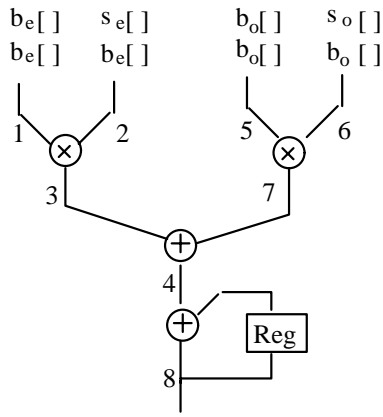
With the basic architecture design defined, the algorithm can be mapped onto hardware by specifying controller behavior. The goals of doing scheduling and allocation are to minimize activity-related power consumption throughout the data path and meet the throughput requirement of the speech codec. These goals can be met by taking advantage of instruction-level parallelism (ILP) to speed execution of the algorithm and data characteristics to reduce transition activity.

In the rest of this section, key places to look for opportunities in source code to exploit data characteristics during scheduling and allocation are covered. A technique that guarantees an optimal solution is not presented in this paper. However, it is shown through the VSELP codec example that using the activity-related power reduction heuristic to narrow down the design space to a small number of good candidate designs can result in significant power reduction. The importance of Step 2 of the heuristic, i.e. transformations, is specifically focussed on.

The most computation intensive portions of the VSELP codec are the lag and codebook searches performed at the encoder. These modules account for 64% of the multiplications in the encoder, and thus represent one of the best opportunities to reduce overall power consumption with a small amount of design. This information partially satisfies Step 1 of the heuristic, because it identifies where in the VSELP algorithm power reduction could be significant to the entire application. Note, however, that the heuristic is still needed to determine whether or not there is potential to achieve significant activity-related power reduction in these modules.

The basic form of the expensive processing that takes place during a lag or codebook search is shown below in terms of C code:

```
for( $i = 0; i < N; i++$ )
{
```



Node	$\alpha_{0 \rightarrow 1}$	α_{OS}	n_{IS}	n_{CMD}	n_{IT}
1	0.2008	0.2174	5.7208	0.6507	0.3606
2	0.1895	0.2458	6.1849	0.6502	1.2441
3	0.1320	0.1673	11.8011	0.3679	8.0175
4	0.1347	0.1502	10.8295	0.4161	7.9762
5	0.2009	0.2199	5.7576	0.6372	0.3776
6	0.1894	0.2449	6.1906	0.6522	1.2556
7	0.1318	0.1684	11.8705	0.3593	8.0274
8	0.1253	0.0448	7.6970	2.7273	7.9091

Figure 5: Scheduling and allocation choice made without activity consideration

```

g[i]=0;
for(j = 0; j < M; j++)
    g[i]=g[i]+b[i][j]*b[i][j];
c[i]=0;
for(j = 0; j < M; j++)
    c[i]=c[i]+s[j]*b[i][j];
}

```

The reason so much processing is required for this computation is N and M are typically quite large, resulting in $2MN$ multiply-accumulate (MAC) operations per speech subframe.

If the code is stepped through sequentially and an attempt is made only to exploit the parallelism of the architecture, one of the most obvious ways to map these operations onto the VSELP codec architecture is shown in Figure 5 for one iteration of the i loop. All of the multiplications corresponding to even j indices are done by one multiplier, while all of the odd j -indexed multiplications are done by the other. In the figure, $b_e[]$ and $s_e[]$ are vectors of even j -indexed $b[i][j]$ and $s[j]$ values, respectively. Similarly, $b_o[]$ and $s_o[]$ are vectors of the odd values. The loop that calculates $g[i]$ is computed first, followed by the $c[i]$ loop. Model parameter values (α_{OS} , n_{IS} , n_{CMD} , n_{IT}) and $\alpha_{0 \rightarrow 1}$ resulting during Lag Search processing for a standard phonetically balanced speech segment are shown in the table included in Figure 5. Results for Codebook 1 and Codebook 2 Search processing are similar. The speech segment used is a male speaker uttering "Card games are fun to play." Other standard phonetically balanced speech segments from male and female subjects were

tested as well and yielded similar characteristics.

The activity data in this table shows lower than nominal activity for all nodes. A significant number of intersecting sign bits exist between consecutive values on all nodes due to downscaling introduced to avoid overflows during accumulation. The nature of the data is such that opposite sign activity is low for all nodes. Significant activity reduction at nodes 3, 4, 7, and 8 is due to truncation that is performed at the output of the multiplier to make the result only 24 bits wide. While these are nice results, they hint that activity and, more importantly, power can be further reduced by more thoughtful scheduling and allocation using the heuristic.

To complete the first step of the activity-related power reduction heuristic, it is necessary to determine how much activity-related power reduction is significant for this part of the design. Assume that an average power reduction of 10% or more during encoding, compared to the case where the obvious implementation discussed above is employed, is considered significant. Since the lag and codebook searches use about 64% of the multiplications and the elements that influence power consumption the most are multipliers and memory (see Table 4), it may be possible to achieve the 10% reduction by only considering these datapath elements. Multipliers and memory account for approximately 50% of overall power consumption in the obvious implementation of the lag and codebook searches. Thus, their combined power consumption would have to be reduced by about 31% to achieve the 10% average power reduction for the encoder ($31\% * 50\% * 64\% \approx 10\%$).

3.3.1 Power Reduction through Exploitation of Data Characteristics in Straight-Line Code

Next, the importance of performing transformations to achieve activity reduction (Step 2) will be demonstrated by first applying the heuristic to the source code without any transformations. Significant power reduction can typically be achieved when judicious scheduling and allocation is done for straight-line code contained within a loop, simply because the computation is repeated multiple times during execution. There are two independent straight-line code segments in this example:

$$g[i]=g[i]+b[i][j]*b[i][j];$$

and

$$c[i]=c[i]+s[j]*b[i][j];$$

The functionality of both of these segments is the same, so only the $g[i]$ segment will be discussed.

In the straight-line code for the $g[i]$ loop, there are two opportunities for activity reduction. First, the two values being multiplied are identical. This is a special case of data varying so slowly that it does not change at all (Step 3c of the heuristic). To significantly reduce switching activity in this case (Step 4 of the heuristic), two different buses should not be used to load the multiplier inputs from memory. The VSELP architecture design should instead allow the same memory bus value to be accessed by both multiplier inputs, so that only one memory read, over one memory bus, is really required. In essence, Steps 4b and 4d of the heuristic have been applied here, because during these operations, transitions no longer occur on the memory bus that is no longer used. In a simulation of Lag Search computation using the same speech segment as was used for the obvious implementation simulation, memory bus activity was reduced with this implementation by almost 26% compared to the obvious implementation. As an added benefit, 1/4 as many memory reads were performed. Thus, average power consumption of memory during lag and codebook searches is expected to be reduced by about 25% compared to the obvious implementation. This is a very good result, but only provides about a 13% reduction in power for these searches compared to the obvious implementation.

The second opportunity for activity reduction applies to adder inputs and has little effect on the power reduction goal for the lag and codebook searches. However, it will be considered here for the sake of demonstrating the usefulness of the heuristic. This activity reduction opportunity results from the observation that the difference between consecutive $g[i]$ values is the result of a squaring operation. This result is always positive and contains a significant number of sign bits to avoid overflows during accumulation. As a result, $g[i]$ values will vary slowly, and consecutive $g[i]$ values will always make same sign transitions (Steps 3b and 3c of the heuristic). Therefore, consecutive $g[i]$ values should be stored in the same register (Steps 4b and 4d of the heuristic). For the $c[i]$ computation, the same effect will occur, except that there will be some opposite sign transition activity due to the fact that two different values are being multiplied in that loop, rather than a single value being squared.

Figure 6 shows a new implementation that exploits this second opportunity for activity reduction for the $g[i]$ and $c[i]$ computations. Model parameter values and $\alpha_{0 \rightarrow 1}$ resulting from the same

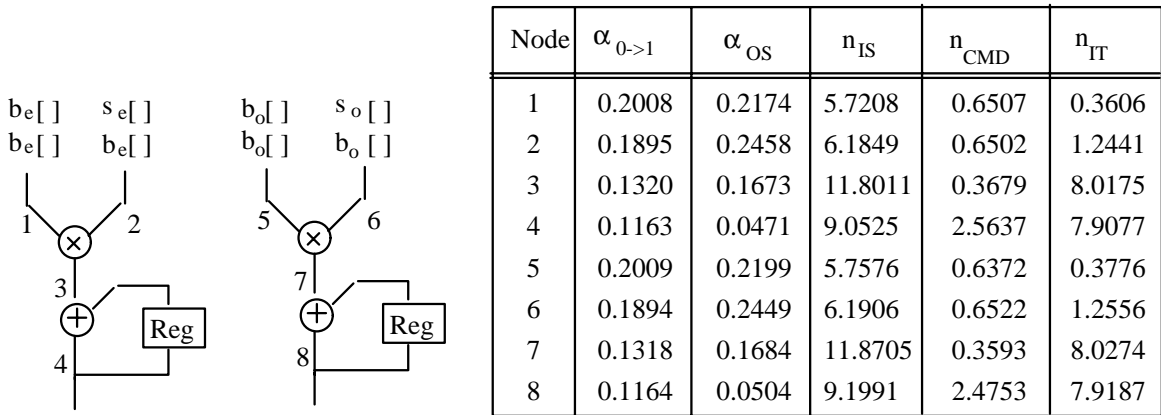


Figure 6: Scheduling and allocation choice made by considering straight-line code

speech signal input and Lag Search computation as for the obvious implementation are shown in the table in Figure 6. Such information allows expected activity reduction to be validated (Step 6 of the heuristic). This implementation does not add intermediate multiplication results before accumulation like the obvious implementation did. (Note, however, that once all even and odd indexed values are accumulated separately, the results must be added together to get the final values of $g[i]$ and $c[i]$.) As a result, activity at the inputs of the adders is reduced on average by about 5% compared to the obvious implementation, but significant reduction in power is not achieved because adder power consumption is already quite low.

The almost 13% reduction in average multiplication and memory power during lag and codebook searches achieved by consideration of straight-line code that inherently exists in the algorithm is impressive but still less than half of the power reduction goal. Thus, by itself, this reduction is not worth the effort. This is precisely why opportunities must also be considered outside of loop boundaries (Step 2 of the heuristic), as will be demonstrated next.

3.3.2 Power Reduction through Loop Unrolling

Similar to ILP exploitation, data characteristic exploitation opportunities can many times be identified from source code by using loop unrolling to increase the amount of straight-line code. Consider the $g[i]$ loop again unrolled by a factor of two (Step 2b of the heuristic). The C code for this case is shown below.

```

g[i]=0;
for(j = 0; j < M; j += 2)

```

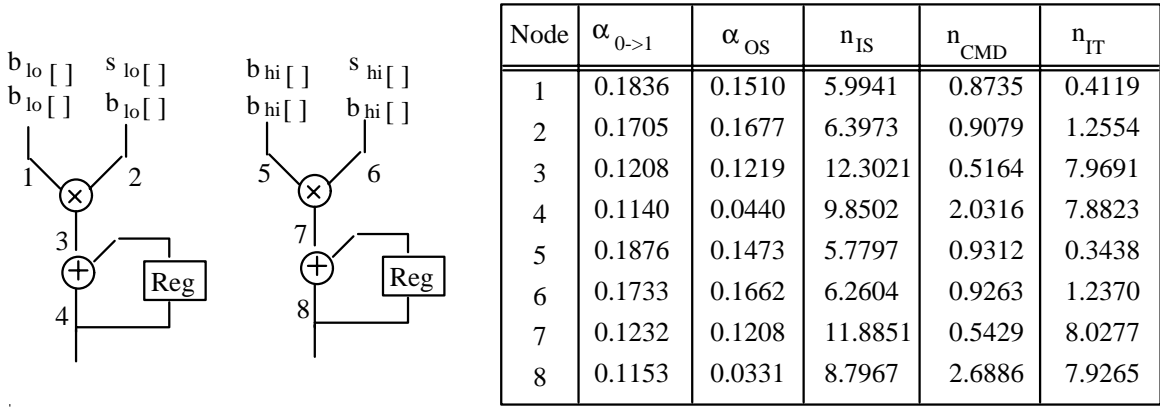


Figure 7: Scheduling and allocation choice made by considering unrolled loops

```

{
  g[i]=g[i]+b[i][j]*b[i][j];
  g[i]=g[i]+b[i][j+1]*b[i][j+1];
}

```

In this form, it can be seen quite easily that consecutive multiplications involve consecutive values from the b array, $b[i][j]$ and $b[i][j+1]$. It can be determined either from the algorithm or simulation that these array elements tend to have low α_{OS} and significantly high n_{IS} (Step 3b of the heuristic). These characteristics are basically due to lowpass filtering and scaling. As a result, allocating the same multiplier to as many consecutive multiplications as possible will reduce activity at the inputs of that multiplier (Step 4b of the heuristic).

An implementation that accomplishes this is shown in Figure 7 for a single iteration of the i loop. The table in Figure 7 shows analysis parameters for this implementation resulting from Lag Search computation with the same sample speech signal again. In this figure, the $b_{lo}[]$ and $s_{lo}[]$ vectors are the lower $M/2$ j -indexed elements of $b[i][j]$ and $s[j]$, respectively, and the $b_{hi}[]$ and $s_{hi}[]$ vectors are the upper $M/2$ j -indexed elements of $b[i][j]$ and $s[j]$, respectively.

In this new implementation, activity is reduced at multiplier inputs by about 8% compared to the obvious implementation considered previously. As a result, average power consumption of multipliers during lag and codebook searches is now expected to be reduced by about 6% compared to the obvious implementation, while memory power consumption is reduced by about 28%. Thus, about 17% reduction in power consumption of multipliers and memory for these searches compared

to the obvious implementation is now estimated.

Propagation of the reduced multiplier input activity results in activity reduction at other nodes in this implementation. However, this activity reduction does not have a significant impact on power, once again, because it affects adder inputs. For the sake of further demonstrating the capabilities of the heuristic, these reductions will be mentioned briefly. Multiplier output activity is reduced by about 7%. This reduction is due to reduced α_{OS} and significantly high n_{IS} at both inputs of each multiplier when computing $c[i]$ and that signs of the two multiplier inputs have no strong relationship with each other (Step 3b of the heuristic). Since the same multiplier is used for consecutive calculations, α_{OS} is low and n_{IS} is high at the output of the multiplier (Step 4b of the heuristic). The reduction at the multiplier outputs, coupled with the reduction achieved previously by exploiting data characteristics of straight-line code results in a 9% decrease in activity at adder inputs compared to the obvious implementation.

By performing loop unrolling, power consumption of multipliers and memory for lag and code-book searches has been reduced compared to the straight-line implementation. However, substantial additional reduction is still required to meet the power reduction goal of 31%. As is shown next, consideration of independently executable blocks of code as well (Step 2a of the heuristic) can achieve this goal.

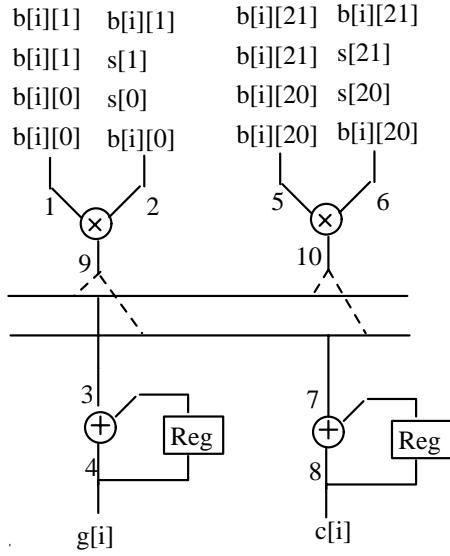
3.3.3 Power Reduction through Parallel Execution of Independent Blocks

In addition to identifying data characteristics in straight-line code of unrolled loops, it is important to identify data relationships between independently executable blocks of source code (Step 2a of the heuristic). Consider the case where both the $g[i]$ and $c[i]$ calculation loops are combined and unrolled by a factor of two:

```

for( $i = 0; i < N; i ++$ )
{
 $g[i]=0;$ 
 $c[i]=0;$ 
  for( $j = 0; j < M; j ++$ )
  {
     $g[i]=g[i]+b[i][j]*b[i][j];$ 

```



Node	$\alpha_{0 \rightarrow 1}$	α_{OS}	n_{IS}	n_{CMD}	n_{IT}
1	0.0918	0.0755	6.2529	5.1810	0.2059
2	0.2082	0.3346	6.0122	0.5777	1.1061
3	0.1200	0.0734	11.2082	0.4779	7.5272
4	0.1224	0.0439	6.4574	4.3477	3.3330
5	0.0938	0.0736	6.0201	5.3354	0.1719
6	0.2095	0.3261	5.8717	0.5858	1.0403
7	0.1401	0.2533	12.8770	0.4915	7.9614
8	0.1139	0.0512	9.0425	3.0236	7.9290
9	0.1478	0.2950	11.7528	0.3921	7.9819
10	0.1498	0.2932	10.8720	0.3996	7.6343

Figure 8: Scheduling and allocation choice made by considering independent blocks

```

g[i]=g[i]+b[i][j+1]*b[i][j+1];
c[i]=c[i]+s[j]*b[i][j];
c[i]=c[i]+s[j+1]*b[i][j+1];
}
}

```

In this form, it can be seen by inspection that the same iteration of $g[i]$ and $c[i]$ calculations involve the same $b[i][j]$ value (Steps 3b and 3c of the heuristic). As a result, if these calculations are performed consecutively using the same multiplier, activity at one of the inputs can be significantly reduced, because the same $b[i][j]$ value will be used (Steps 4b and 4d of the heuristic). By storing this value at the input register to the multiplier, rather than reading it from memory twice, power is further reduced. However, the real trick to taking advantage of this reduction is keeping activity on the other multiplier input low. This can be accomplished by performing the MAC operations in the following order:

```

g[i]=g[i]+b[i][j]*b[i][j];
c[i]=c[i]+s[j]*b[i][j];
c[i]=c[i]+s[j+1]*b[i][j+1];
g[i]=g[i]+b[i][j+1]*b[i][j+1];

```

The corresponding implementation is shown in Figure 8. Here, the first four pairs of inputs

are shown for each multiplier with the very first inputs appearing at the bottom of the list. The schedules for Nodes 2 and 6 take advantage of the high n_{IS} (Step 3b of the heuristic) and lower α_{OS} between consecutive values in the $s[]$ vector to keep average activity relatively low (Step 4b of the heuristic). The table of parameters resulting from Lag Search computation with the same sample speech signal again is included in Figure 8 (Step 6 of the heuristic). If $b[][]$ and $s[]$ values were instead alternated at these nodes, activity would be near 0.25 instead of 0.21, because there would be no relationship between signs of consecutive values.

From the parameter table, it can be seen that average activity is reduced by almost 23% at multiplier inputs compared to the obvious implementation. Even more impressive is that average memory bus activity is reduced by almost 56% due to the same activity reductions achieved at Nodes 1 and 5. In addition, only half as many memory reads are required by this implementation as the obvious implementation.

The few tradeoffs associated with this implementation, are expected to be insignificant. One tradeoff is that activity at the outputs of the multipliers is actually larger for this implementation than the version that resulted from loop unrolling alone. Control is slightly more complicated due to the demultiplexing that must go on to accumulate $c[i]$ and $g[i]$ values with separate adders. Activity reduction at the adder inputs is only about 5% compared with the 9% reduction achieved with the design based on loop unrolling only.

With the implementation in Figure 8, average power consumption of memory and multipliers for the lag and codebook searches is estimated to be almost 35% less than the obvious implementation. This reduction is better than the 31% power reduction goal, making it a good design choice (Step 5 of the heuristic). Average power consumption during encoding is estimated to be only 34.9 mW with this design, an almost 11% decrease compared to the case where the obvious implementation is employed. This significant reduction was achieved because the entire activity-related power reduction heuristic was applied, including independently executable block consideration and loop unrolling (Step 2 of the heuristic). By application of the heuristic to the Synthesis Filter (the next most computation intensive module in the encoder) estimated average power consumption of about 32.9 mW during encoding can be achieved. As a result, power reduction of about 15% on average, compared to an obvious implementation, is possible by applying the heuristic to the most

computation intensive modules.

4 Conclusions

This paper presents a systematic new approach to reducing switching activity-related power consumption. At the core of this technique is a new model that relates switching activity of datapath interconnect to characteristics of fixed-point, two's complement data. This model, based on four practical parameters, is more intuitive and general than previous models. An activity-related power reduction heuristic based on the model is presented, and its ability to guide high-level, low power design space exploration is demonstrated. The heuristic aids in determining how activity reduction can significantly reduce power consumption and in efficiently finding such opportunities. Application of this heuristic to an IS-54 VSELP speech codec design example results in up to 56% activity reduction at high energy locations in the datapath and estimated power reduction of about 15% on average during encoding. Thus, using the new model and heuristic in the manner presented here, it is expected that similar designs could efficiently achieve activity-related power reduction of this magnitude as well.

References

- [1] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp. 498-523, Apr. 1995.
- [2] A. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Broderon, "Optimizing power using transformations," *IEEE Trans. on Computer-Aided Design*, vol. 14, pp. 12-30, Jan. 1995.
- [3] A. Raghunathan and N. Jha, "Behavioral synthesis for low power", in *Proc. ICCD*, pp. 318-322, Oct. 1994.
- [4] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis", in *Proc. ISLPED*, pp. 69-74, Apr. 1995.
- [5] A. Raghunathan and N. Jha, "An Iterative Improvement Algorithm for Low Power Data Path Synthesis", in *Proc. ICCAD*, pp. 597-602, Nov. 1995.

- [6] EIA/TIA-PN2398 (IS-54), "The 8 kbits/s VSELP algorithm", 1989.
- [7] P. Landman, *Low-Power Architectural Design Methodologies*, Ph.D. Dissertation, UC Berkeley, August 1994.
- [8] S. Gupta and F. N. Najm, "Power Modeling for High-Level Power Estimation", *IEEE Trans. on VLSI Systems*, vol. 8, no. 1, pp. 18-29, Feb. 2000.
- [9] A. Raghunathan, S. Dey, and N. Jha, "Register-Transfer Level Estimation Techniques for Switching Activity and Power Consumption", in *Proc. ICCAD*, pp. 158-165, Nov. 1996.
- [10] D. Shin and K. Choi, "Low Power High Level Synthesis by Increasing Data Correlation", in *Proc. ISLPED*, pp. 62-67, 1997.
- [11] K. Masselos, et al., "Low Power Synthesis of Sum-Of-Products Computation", in *Proc. ISLPED*, pp. 234-237, July 2000.
- [12] H. Kojima and A. Shridhar, "Interlaced Accumulation Programming for Low Power DSP", in *Proc. ISLPED*, pp. 213-216, Aug. 1996.
- [13] A. Chatterjee and R. Roy, "Synthesis of Low Power Linear DSP Circuits using Activity Metrics", in *Proc. of the IEEE Int'l Conf. on VLSI Design*, pp. 265-270, 1994.
- [14] P. Landman and J. Rabaey, "Architectural Power Analysis: The Dual Bit Type Method", *IEEE Trans. on VLSI Systems*, vol. 3, no. 2, pp. 173-187, 1995.
- [15] E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units", in *Proc. ISLPED*, pp. 99-104, Apr. 1995.
- [16] T. Chang, Y. Chu, and C. Jen, "Low-Power FIR Filter Realization with Differential Coefficients and Inputs", *IEEE Trans. Circuits Syst. II*, vol. 47, no. 2, pp. 137-145, Feb. 2000.
- [17] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "Analytical Estimation of Signal Transition Activity from Word-Level Statistics", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, pp. 718-733, July 1997.
- [18] G. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, 1998.

- [19] S. Ramprasad, N. R. Shanbhag, and I. N. Hajj, "Signal Coding for Low Power: Fundamental Limits and Practical Realizations", *IEEE Trans. Circuits Syst. II*, vol. 46, no. 7, pp. 923-929, July 1999.
- [20] R. Henning and C. Chakrabarti, "Relating Data Characteristics to Transition Activity in High-Level Static CMOS Design," in *Proc. 13th Int'l Conf. VLSI Design*, pp. 38-43, Jan. 2000.
- [21] R. Henning and C. Chakrabarti, "Activity Models for use in Low Power, High-Level Synthesis," in *Proc. ICASSP*, vol. 4, pp. 1881-1884, Mar. 1999.
- [22] A. Spanias, "Speech coding," *Proc. IEEE*, vol. 82, pp. 1541-1582, Oct. 1994.
- [23] I. Gerson and M. Jasiuk, "Vector sum excited linear prediction (VSELP) speech coding at 8 kbits/s", in *Proc. ICASSP*, pp. 461-464, Apr. 1990.
- [24] M. Rajesh, *Low Power Architectures for the IS-54 VSELP Algorithm*, MS thesis, Arizona State University, 1995.

List of Figures

1	Transition between (a) two positive 16-bit values and (b) a positive and a negative 16-bit value.	5
2	Estimated activity vs. opposite sign transition activity as the average number of intersecting sign bits varies.	8
3	Block diagram of IS-54 VSELP Speech Encoder	12
4	Block diagram of the processor	15
5	Scheduling and allocation choice made without activity consideration	21
6	Scheduling and allocation choice made by considering straight-line code	24
7	Scheduling and allocation choice made by considering unrolled loops	25
8	Scheduling and allocation choice made by considering independent blocks	27

List of Tables

1	Interesting scheduling options for three data streams allocated to the same bus . . .	9
2	Operations used in algorithm design broken down module by module	14
3	Time estimate and number of 16x16 bit multiplications used per frame in IS-54 VSELP speech encoder design broken down by module	16
4	Area estimates for the data path	18
5	Power estimates for the datapath	19