High-Level Design Synthesis of a Low Power, VLIW Processor for the IS-54 VSELP Speech Encoder

Russell E. Henning and Chaitali Chakrabarti
Arizona State University, Tempe, AZ 85287-5706

Abstract

General purpose DSPs typically used to implement speech coders in digital cellular phones do not allow enough exploitation of the speech coding algorithm itself for power reduction. In this paper, high-level design synthesis of a low power, VLIW (very long instruction word) processor dedicated to implementing the IS-54 VSELP speech encoding algorithm is presented. Significant power reduction is achieved through algorithm dependent techniques, including application specific hardware design, supply voltage reduction through highly parallel execution, and exploitation of data correlation inherent to the algorithm. Preliminary estimates indicate that the design could result in a 5.35 mm^2 processor that executes in real-time with an average power dissipation of about 28 mW.

1. Introduction

To reduce the frequency with which battery operated devices must be recharged, operational power dissipation should be lowered. In a system like that of the digital cellular telephony standard IS-54, speech coding is computation intensive and therefore has the potential to dissipate a lot of power in a handset. However, the general purpose digital signal processors typically used to implement speech coders do not offer a way to significantly reduce power by exploiting the algorithm they implement. As a result, operation is not achieved at low enough power levels to meet consumer demands for more convenient use.

One solution to this problem is to replace the general purpose DSP with a low power processor specifically designed for the particular speech coding algorithm being implemented. This paper presents high-level design synthesis of a low power processor dedicated to implementing the IS-54 VSELP speech encoding algorithm. Figure 2 shows a block diagram of the IS-54 VSELP speech encoder. It uses an analysis by synthesis process to do parametric, block coding of 8 kHz sampled speech at 8 kb/s. For background on speech encoding, refer to [1]. A description of the IS-54 VSELP speech coder is given in [2].

The high-level design synthesis process for the speech encoder is broken into four steps that incorporate algorithm dependent power reduction techniques. The first step is algorithm design where power is saved by simplifying the algorithm. The next step is architecture design where a VLIW processor architecture is chosen to speed execution of the algorithm, allowing power dissipation to be reduced through supply voltage reduction. The third step is scheduling and allocation where switching activity reduction is achieved by exploiting data correlation. The final step is validation where power can be saved by iteratively optimizing the instruction sequence through simulation and redesign. General power reduction techniques have been described in [3] and [4].

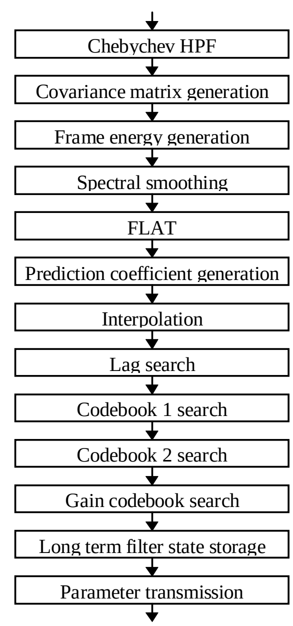

Figure 1: Block Diagram of IS-54 VSELP Speech Encoder

The rest of the paper is organized as follows. Sources of dynamic power dissipation in CMOS circuits are reviewed, and high-level design synthesis is briefly introduced. Then, power saving techniques for each step of the high-level design synthesis process are described and illustrated for the VSELP speech encoder. Finally, power and area performance data is given for the current state of the design.

2. Key Sources of Power Dissipation

Average power dissipation at a node in a conventional digital CMOS circuit can be described by the following equation [8]:

$$P_{average} = P_{switching} + P_{short-circuit} + P_{leakage}. \quad (1)$$

In high-level design, the term of most interest is the switching component

$$P_{switching} = \alpha_{0\to 1} C_L V_{dd}^2 f_{clk}. \quad (2)$$

The node transition activity factor $\alpha_{0\to 1}$ is the average number of times the node makes a 0 to 1 transition in one clock cycle. The rest of the terms in (2) are the capacitive load for the node C_L, supply voltage V_{dd}, and clock frequency f_{clk}. In the next section, the potential for reducing average power dissipation by reducing the node transition activity factor and supply voltage in high-level design synthesis will be discussed.

3. High-Level Design Synthesis Process

The high-level design synthesis process can be broken down into algorithm design, data path design, scheduling and allocation, and validation [6]. Power can be reduced by reducing the terms in (2) as part of this process. The term that typically causes the largest reduction is V_{dd} because of its quadratic effect on switching power. Area can be traded to reduce this term as will be explained in Section 5. Of the remaining terms, $\alpha_{0\to 1}$ is most well suited to reduction through high-level design synthesis of a dedicated processor, because switching depends heavily on the implementation of the algorithm. This term can be reduced in all four steps of the design synthesis process. Capacitive loads are largely dependent upon circuit topology choices and processing technology used and are affected in high-level design only by the data path design. The clock period is approximately the same as the delay of the longest pipeline stage. So, clock frequency f_{clk} is basically fixed once V_{dd} and the data path are specified.

4. Algorithm Design

Algorithm design is the first step in the high-level design synthesis process. In this step it is determined how far precision and range of data types in an algorithm can be reduced without significantly compromising speech quality. It is important to balance this goal with controlling the complexity of the implementation, including the number of different data path components, amount of interconnect, and complexity of control required.

For the VSELP speech coder, a *C* model was used to check output quality versus varied data type precisions and ranges for each module in Figure 2. The precision and ranges required by the algorithm design are summarized in Table 1. By reducing precision and range of data types a number of design goals are achieved. Simplified computation results in less transition activity. Simpler computations can also be executed faster. Thus, algorithm design aids in the goal of meeting real-time execution requirements. Storage requirements are reduced, resulting in less area consumed by RAM and ROM and reduced transition activity due to memory accesses.

Table 1: Operations Used in Algorithm Design Broken Down by Module

Module	Operations Used
Chebychev 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
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

The basic requirements of the data path and controller are also defined. The design calls for mostly 16 bit fixed point multiplications and additions. However, 32 bit fixed and floating point data types and operations, as well as scaling of fixed point values, must be supported. Such a design can be supported by a 16 bit data path consisting of a 16 bit fixed point adder, 16X16 bit fixed point multiplier, and a 32 bit barrel 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. All data can be stored in 16 bit memory locations with 32 bit fixed or floating point values stored in two 16 bit memory locations.

5. Data Path Design

The data path design step of the synthesis process requires the basic data path specifications provided by the algorithm design. Using these specifications, a low power data path is designed that achieves the required throughput for real-time use. Much of the data path design described below is derived from [7].

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 16 bit carry look-ahead adder have been selected for the 16 bit data path specified by the algorithm design. Both of these functional units have array structures that require more area than simpler circuits. This area is traded for speed so that real-time operation can be achieved with a low 2 V supply voltage.

It is assumed that the 16X16 bit Baugh-Wooley multiplier has a 200 ns latency and the 16 bit carry look-ahead adder has a latency of 50 ns for a 5 V supply voltage. At a supply voltage of 2 V, these latencies scale to about 800 ns for the multiplier and 200 ns for the adder. From the algorithm design, it is approximated that 128,500 16X16 bit fixed point multiplications must be performed to encode one frame of speech. Therefore, with V_{dd}=2 V, throughput must be increased by at least a factor of six to have the algorithm execute within the 20 ms per frame requirement of IS-54. Since data dependencies prevent multiplications from being performed every cycle, it is necessary to increase throughput by an even larger factor.

Since the multiplier can be pipelined to four levels of approximately equal latencies, it is possible to make its pipeline latency equivalent to the latency of an adder, 200 ns. This is a desirable quality for the data path since the only unit that needs pipeline registers within its structure is the multiplier. All other hardware require only input registers. However, pipelining the multiplier to four levels only increases throughput by about a factor of four.

To increase the throughput further, parallel execution capability in the form of multiple functional units is introduced into the data path. This is a more area costly solution than pipelining, but it does not require the design of new functional units, only copies. It is estimated that about a factor of eight increase in throughput is adequate to achieve real-time operation, so two identical multipliers pipelined to four levels are included in the data path.

To keep the speedup factor from being any higher than eight, additions must be executed quickly enough during multiply accumulate computations to allow the multipliers to execute with a reasonable number of idle states. The majority of multiply accumulate functions only require 16 bit accumulation which can be handled with minimal delay by providing two adders in the data path, one for each multiplier. A third adder is introduced to speed computation during 32 bit accumulation.

A VLIW architecture has been chosen to support parallel execution of instructions. This choice shifts much of the control complexity from power hungry processor hardware to the compiler which handles scheduling and allocation statically [8]. The control unit has been assumed to be for the most part self contained for now in order to keep the problem of exploiting data correlation tractable. Future work will be devoted to complete design of the controller.

A block diagram of the VSELP speech encoder architecture is shown in Figure 2. It has a few power saving features in addition to those already mentioned. Independent busses connecting memory blocks to the data path have been used to preserve correlation between signals read from and written to memory. Only four registers are included in the data path to reduce chip area and control complexity. More storage and a method of disallowing switching in idle data path elements are provided by the input registers. The controller can disable input registers to data path elements at any time, causing the last value to remain at the input of the data path element until it is enabled again. Since the inputs do not change, no switching will occur in the functional unit and the input or output value can be effectively stored by the data path unit. It is worthwhile to note that clock load is not varied to achieve functional unit suspension using this method.

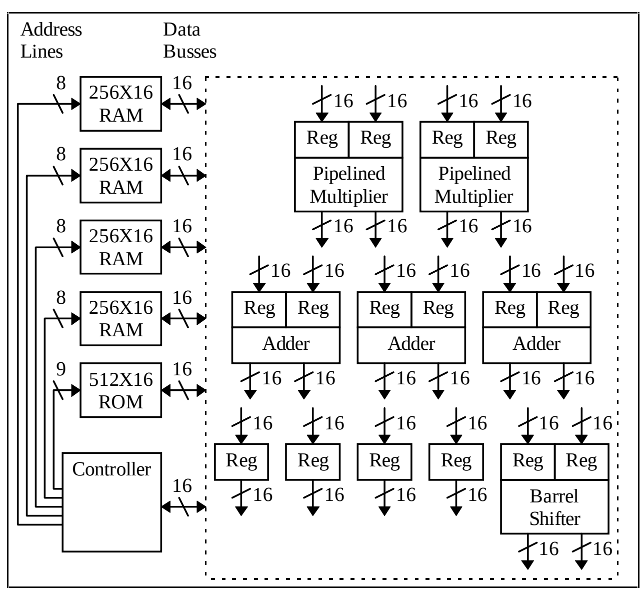

Figure 2: Block Diagram of Processor Design

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 200 ns. The barrel shifter, memory accesses, and register accesses all take less than 100 ns to execute at a supply voltage of 2 V. Therefore, another clock with a period of 100 ns is supplied to the registers at the inputs of their respective units to take advantage of the speed of these functions. With this architecture, the time for a frame to be generated can be estimated from multiplier usage for each module. The execution time estimate for each module is given in Table 2 along with the number of multiplications. This table shows that the encoder architecture is capable of processing speech frames within the 20 ms requirement of the standard.

Table 2: Time Estimate and Number of 16X16 bit Multiplications Used per Frame in IS-54 VSELP Speech Encoder Design Broken Down by Module

Module	Execution Time per Frame (ms)	16X16 bit Multiplications per Frame
Chebychev HPF	0.6415	4160
Covariance matrix generation	0.2928	1870
Frame energy generation	0.0019	0
Spectral smoothing	0.0066	66
FLAT	0.4112	2574
Prediction coefficient generation	0.0078	55
Interpolation	0.3598	471
Weighting filter	0.7200	4800
Lag search	4.7800	46400
Codebook 1 search	2.8672	27280
Codebook 2 search	2.9074	28560
Gain codebook search	1.5776	10172
Long term filter state storage	0.0000	0
Parameter transmission	0.3140	2092
All modules	14.8878	128500

The nodes where switching costs the most power in the processor data path include functional unit inputs and outputs and the data busses. At functional unit inputs, any change in an input value will trigger more switching inside the unit. Outputs of functional units and data busses have large capacitive loads causing transition activity to have a greater effect on average power. In the next section, it is shown how switching can be reduced at high cost nodes through judicious scheduling and allocation.

6. Scheduling and Allocation

With the data path design complete, the algorithm can be mapped onto hardware by specifying controller behavior. In doing so, the sequence of operations for implementing the algorithm is scheduled in time and specific functional units from the data path are allocated to perform these operations. The two goals of doing scheduling and allocation are to reduce switching activity at high power dissipating nodes and meet the throughput requirements of the speech encoder.

Exposing parallelism to speed execution of the algorithm and taking full advantage of signal correlation to reduce transition activity are the two main techniques that can be used to meet the goals of scheduling and allocation. Data correlation and parallelism can be exploited through static scheduling and allocation in ways that would be very costly in terms of power and area for hardware to attempt dynamically. For the VSELP speech encoder design, static scheduling and allocation is being done by hand. It is, however, more desirable for a compiler to be designed to do this job.

The most obvious way to exploit data correlation and parallelism is to identify data dependencies for operations contained in in-line code by inspection or through the use of a data flow graph. Independent operations of such code can be scheduled and allocated in a flexible manner so switching is reduced by preserving correlation and hardware resources are used as close to full capacity as possible.

Parallelism and correlation can, however, also be exploited at the loop level. Loop unrolling is a well known method of exposing independent operations. In addition, signal correlation can be made available for achieving significant switching reduction. Take, for instance, the following *C* code showing a loop executing multiply accumulate operations in the VSELP encoder algorithm.

```
ac[0][i]=0;
for(i=0;i<=10;i++)
    for(k=10;k<170;k++)
        ac[0][i]=ac[0][i]+s[k]*s[k-i];
```

A possible schedule and allocation choice for this high-level code is shown in Table 3 for the memory reads and multiplications performed in one iteration of the loop. The table shows the signals that are provided to the inputs of data path units for a sequence of clock cycles. A clock cycle corresponds to the variable k changing in the code. Data is only latched into the multiplier on odd numbered clock cycles, whereas the other components shown latch data every clock cycle.

At the beginning of this iteration i=5 and k=10. We assume speech samples are being read from the same memory bank represented by the array s[] and consecutive speech samples have been written to consecutive addresses. The RAM address input of Table 3 indicates that these samples are not read in an order that exploits correlation between speech samples to reduce switching.

Table 3: Schedule and Allocation Choice for Loop before Unrolling

Clock Cycle	Mult 1 Input 1	Mult 2 Input 2	Reg 4 Input	RAM 1 Read Enable	RAM 1 Address Input
0					
1				1	&s[10]
2	RAM 1			1	&s[5]
3		RAM 1		1	&s[11]
4	RAM 1			1	&s[6]
5		RAM 1		1	&s[12]
6	RAM 1			1	&s[7]
7		RAM 1		1	&s[13]
8	RAM 1			1	&s[8]
9		RAM 1		1	&s[14]

Now, consider the same loop unrolled by a factor of 2. The *C* code for this case is shown below.

```
ac[0][i]=0;
for(i=0;i<=10;i++)
    for(k=10;k<170;k+=2){
        ac[0][i]=ac[0][i]+s[k]*s[k-i];
        ac[0][i]=ac[0][i]+s[k+1]*s[k-i+1];}
```

From the body of this loop it is possible to see that consecutive multiplications require speech sample inputs that are highly correlated. The values s[k] and s[k+1] are highly correlated as are s[k-i] and s[k-i+1]. Since memory reads are always independent (there are no read after read hazards) speech samples can be read in an order that reduces switching. The preferred schedule and allocation choice is shown in Table 4. It reads sample values in correlated pairs. In this schedule, register 4 now stores one of the speech samples that is read until it is needed at the multiplier input. Switching in this register is low since its value changes only every four clock cycles and the values stored are fairly correlated because they are only one sample away from their original order. So, the biggest cost of implementing reads in this way is that one of only four registers available is tied up during execution. The gain is a significant reduction in switching on the RAM 1 data bus.

In addition to in-line code and loops, it is important to identify data correlation within and independence between functions or large blocks of high-level code. The way the architecture is designed allows large pieces of independent computations to be done in parallel with separate subsets of functional units. In doing so, correlation within each function or block can be preserved to reduce switching in each subset of the data path. For example, the schedule and allocation choice shown in Table 4 only uses one multiplier. Thus, a multiplier, adder, and some other data path units are not used in executing the loop. In our implementation, almost all the unused units other than the multiplier are used to execute a square root computation and a quantization procedure in parallel with this part of the algorithm.

Table 4: Schedule and Allocation Choice for Loop after Unrolling

Clock Cycle	Mult 1 Input 1	Mult 2 Input 2	Reg 4 Input	RAM 1 Read Enable	RAM 1 Address Input
0				1	&s[11]
1			RAM 1	1	&s[10]
2	RAM 1			1	&s[5]
3		RAM 1		1	&s[6]
4		RAM 1		1	&s[13]
5	Reg 4		RAM 1	1	&s[12]
6	RAM 1			1	&s[7]
7		RAM 1		1	&s[8]
8		RAM 1		1	&s[15]
9	Reg 4		RAM 1	1	&s[14]

Though a multiplier is left idle in the final schedule and allocation choice, there are two practical reasons not to use two multipliers to share the work load. The first is that correlation can be more fully exploited at the inputs of a single multiplier than two working in parallel. The second reason has to do with memory allocation. A second multiplier could not be used in parallel without moving some of the speech samples to another memory bank because RAM 1 is already read from at full capacity using only one multiplier. If memory allocations were made in another block, switching reduction on busses during reads and writes would not be as significant because data correlation could not be as fully exploited. Implementation could also get quite messy. Memory allocation is yet another important consideration in meeting the goals of scheduling and allocation.

7. Validation through VHDL Simulation

Validation of the design is actually an important step in reducing power dissipation. Scheduling and allocation may not be optimal the first time it is done whether a compiler is used or it is done by hand. The validation process can afford the designer an iterative method of finding the best schedule and allocation for low power operation. When the design is modeled in VHDL, not only can it be determined whether proper output is generated, but nodes throughout the circuit can be monitored to determine where and when there is high transition activity. Redesign can then take place to remedy problem points.

8. Power Estimate

Though the synthesis process is not yet complete, preliminary estimates put data path power dissipation at about 28 mW with only the effects of algorithm design, 2 V supply voltage, and input register disables being considered. The power reductions achieved by scheduling correlated values consecutively at the same node will not be considered until validation is complete. At that time, reasonable estimates can be found from simulation output.

The data path power estimate is obtained by estimating the power consumed by the individual components in the data path. Table 5 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 during the 14.8 ms the processor is running. A transition activity factor of 25% is used because signal correlation is not taken into account. The power estimates are for a Motorola 0.5 µm technology.

As a rough estimate, assume the controller requires an additional 20% of the power dissipated in the data path. This assumption gives an average power dissipation of about 37.3 mW while the algorithm is being computed. However, for the 5.2 ms of a frame that the algorithm is not being run, the power consumption is close to zero. So, over a 20 ms frame, about 28 mW of power is consumed on average. Therefore, the processor has an average power dissipation of about 28 mW.

Table 5: Power Estimates for Data Path

Data Path Component	Power Consumption (µW) for $\alpha_{0\rightarrow 1}$=25%	Number	% Time Active	Overall Consumption (µW)
Pipelined Multiplier	5777.22	2	86.31	9972.64
CLA	660.80	3	95.87	1900.53
Barrel Shifter	969.60	1	27.56	267.06
Register	257.40	16	77.47	3190.52
10:1, 16 bit Mux	451.45	12	100.00	5417.40
4:1, 16 bit Mux	180.58	4	100.00	722.32
256X16 bit Memory	5352.10	6	30.00	9633.78
Entire Data Path				31104.25

9. Area Estimate

The estimated area for the data path of the processor, 4.12 mm^2, results from the data shown in Table 6. As a rough estimate, assume control requires an additional 30% of the data path component area. This assumption puts the area of the processor at approximately 5.35 mm^2.

Table 6: Area Estimates for Data Path

Data Path Component	Area Estimate (mm^2)	Number	Total Area (mm^2)
Pipelined Multiplier	0.242791155	2	0.48558231
CLA	0.04166235	3	0.12498705
Barrel Shifter	0.11243988	1	0.11243988
256X16 bit Memory	0.56538519	6	3.3923111
Entire Data Path			4.1153204

Acknowledgments

The majority of this work was supported by the Motorola SABA program. A smaller portion has been conducted through the Center for Low Power Electronics which is supported by the National Science Foundation, the state of Arizona, and industrial partners. The current membership includes Analog Devices, Burr-Brown, Hughes Missile Systems, Intel, Motorola SPS, National Semiconductor, Scientific Monitoring, and Western Design.

References

[1] A. Spanias, "Speech coding," *Proc. IEEE*, vol. 82, pp. 1541-1582, Oct. 1994.

[2] I. Gerson and M. Jasiuk, "Vector sum excited linear prediction (VSELP) speech coding at 8 kbits/s", in *Proc. ICASSP-90*, Apr. 1990, pp. 461-464.

[3] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circ.*, vol. 27, No. 4, pp. 473-484, Apr. 1992.

[4] A. Chandrakasan and R. Brodersen, "Minimizing power consumption in digital CMOS circuits," *Proc. IEEE*, vol. 83, pp. 498-523, Apr. 1995.

[5] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design: A Systems Perspective*. Reading, MA: Addison-Wesley, 1988.

[6] V. Madisetti, *VLSI Signal Processors: An Introduction to Rapid Prototyping and Design Synthesis*. Boston, MA: Butterworth-Heinemann, 1995.

[7] M. Rajesh, *Low Power Architectures for the IS-54 VSELP Algorithm*, MS thesis, Arizona State University, Tempe, Arizona, 1995.

[8] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. San Francisco, CA: MorganKaufmann, 1996.