AN ENERGY SCALABLE FUNCTIONAL UNIT FOR SENSOR SIGNAL PROCESSING

Liping Guo, Mackenzie Scott, Rajeevan Amirtharajah

Dept. of Electrical and Computer Engineering, University of California, Davis CA 95616

ABSTRACT

Low power signal processing functionality is required for wireless sensor network nodes due to their limited battery life. Previously, we have proposed a reconfigurable array of DSP acceleration functional units for such a sensor node. The array maximizes operating life by matching system power consumption to available energy through power scalable approximate signal processing [1]. This paper presents the detailed architecture and implementation of the functional unit in the array. The use of low power building blocks and bit serial processing enables energy scalable implementation of several DSP functions. Post-layout simulation of a semicustom implementation in 0.25 μm CMOS technology demonstrates a factor of three power scalability with input bitwidth for an FIR matched filter.

Index Terms— reconfigurable architecture, arithmetic, microsensors, smart sensor, adaptive signal processing

1. INTRODUCTION

Wireless sensor network technology promises to provide a new information gathering and distribution infrastructure, enabling numerous applications in medical monitoring, environmental science, and security. When wireless communication dominates the sensor node power consumption, signal processing the gathered information before transmission to other nodes is necessary to maximize operating lifetime. We have proposed an energy scalable computational array for such sensor signal processing [1] which consists of functional units embedded in an island-style interconnect structure. The energy scalability of the computational array stems from the energy scalable implementation of the functional units. The functional unit block diagram is shown in Fig. 1. It contains a 4x16 SRAM-based input shift memory, a 16x32 SRAM-based lookup table (LUT), a 32-bit adder, a 32-bit accumulator, and a controller. It supports nine linear/nonlinear functions which include addition/subtraction, complex addition/subtraction, vector dot product (VDP), serial multiply, division, complex multiply, square root, base-2 logarithm, and power function. These functions are the building blocks for a wide range of sensor DSP applications. For instance, VDP computes the multiply-accumulate operation for FIR filters in parallel. FFT requires complex addition and multiplication. Low power techniques at both the architecture and circuit levels have been applied to the functional unit design. At the architecture level, clock gating, block partitioning, guarded inputs, and memory banking reduce power consumption. At the circuit level, an SRAM-based multiported register file replaces a flip-flop-based input shift register and significantly reduces active power. The functional unit provides energy scalable computation by varying (1)input bitwidth, (2)LUT word width, and (3)the number of operation iterations. A configuration word is dedicated to energy scalability control. In Section 2, we describe two custom circuit blocks that enable mechanisms (1) and (2). The implementation of mechanism (3) is described in Section 3. Results and conclusion are presented in Section 4 and Section 5.

This work is supported by the MARCO Interconnect Focus Center under Subcontract No. B-12-M06-S12, the Xilinx University Program, and Xilinx Research Labs.

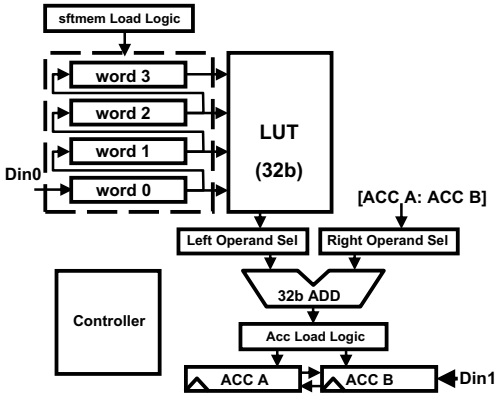

Fig. 1. Functional unit microarchitecture

2. POWER SCALABLE LOGIC BLOCKS

2.1. Lookup Table

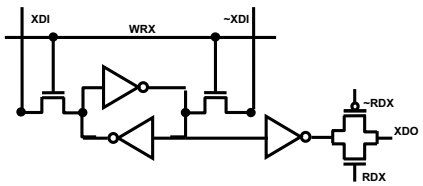

Fig. 2. SRAM-based LUT memory cell

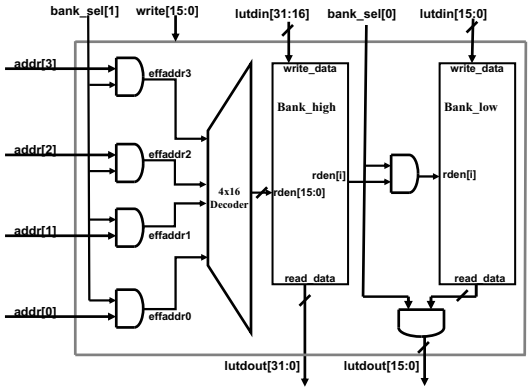

Fig. 3. SRAM-based LUT architecture

The 16x32 SRAM-based LUT is responsible for storing filter coefficients and other constant vectors used in several functions. The memory cell (Fig. 2) has one differential write port and one single-ended read port. The LUT block diagram is shown in Fig. 3. Since LUT write operation happens only at configuration time, the switching activity of the read enable lines accounts for most of the active power. When the LUT is unused, *bank_sel[1]* is asserted to eliminate switching on the address lines and inside the address decoder. In energy scalable computation, if the lower LUT bank is deactivated to conserve power, the read enable lines for the lower-half bank are gated by configuration bit *bank_sel[0]*. The post-layout simulation shows over 40% power savings when the lower LUT bank is deactivated.

2.2. SRAM-based Input Shift memory

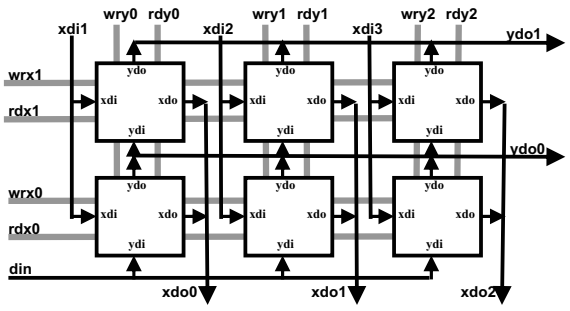

Fig. 4. SRAM-based input shift memory architecture

One way to support power scalability via varying input bitwidth is to implement a shift register using flip-flops and bypass multiplexers [2]. Through appropriate clock gating and multiplexing input selection, the shift register offers a power and input bitwidth tradeoff. However, providing finer-grained power scalability requires multiple clock gating circuits and bypass multiplexers. The incurred overhead may not be justified by the resulting power scalability. Moreover, the flip-flop based shift register does not allow parallel load. For hardware constrained design, the input shift memory must

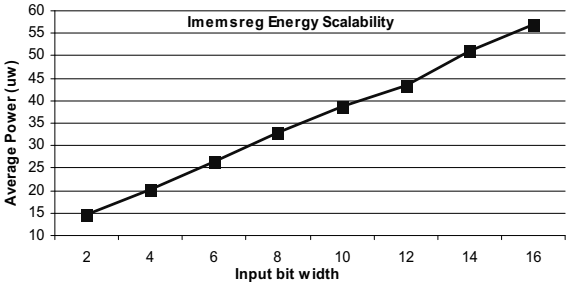

Fig. 5. SRAM-based shift memory power vs. bitwidth

be used as a temporary storage for operands and intermediate results, for which parallel load and readout capabilities are indispensable. The 4x16 SRAM-based input shift memory is designed to allow parallel load and variable bitwidth input. The memory cell is an augmented version of the LUT cell shown in Fig.2 with addition of one differential write port *ydi* and two read ports *ydo* and *xdoe*. The bitlines for the X and Y ports route orthogonally to allow parallel loading along the X direction, shifting multiple parallel bit streams in the Y direction. The y-axis read port, YDO, addresses the LUT. A 2x3 memory block (Fig.4) illustrates the architecture of the input shift memory. By controlling the activation sequence of the read and write signals on the wordlines, arbitrary bitwidth inputs are allowed, which leads to bit-level power scalability with minimum overhead. When read and write signals are asserted for a half clock cycle, the shift memory achieves the same throughput as its flip-flop-based counterpart. Fig.5 shows the post-layout power simulation results for variable input bitwidth. SRAM-based shift memories offer an approximately 10X power reduction over flip-flops [3].

3. ENERGY SCALABLE SERIAL ARITHMETIC

In older VLSI technologies, bit serial algorithms were used to reduce the area of arithmetic blocks. As CMOS scales, leakage current becomes a major contributor to power consumption and at the low frequencies (kHz-MHz) at which most sensor DSP applications operate, serial implementations are lower power than parallel ones due to reduced transistor count [4]. Moreover, in serial arithmetic, the computation result is successively refined as more bits are processed, which naturally provides a power-precision tradeoff. In this section, we describe three functions to illustrate energy scalable implementation using serial arithmetic.

3.1. Vector Dot Product

The vector dot product is implemented using the bit-serial word-parallel Distributed Arithmetic algorithm [5]. Consider the calculation of the inner product $y = \sum_{k=0}^{M-1} a_k x_k$ where **a** is an M-dimensional constant vector and **x** is an M-dimensional input vector. Using N-bit two's complement representation,

the equation can be rearranged as:

$$y = -\sum_{k=0}^{M-1} a_k b_{k(N-1)} 2^{N-1} + \sum_{n=0}^{N-2} \left[\sum_{k=0}^{M-1} a_k b_{kn} \right] 2^n \quad (1)$$

Since each b_{kn} equals 0 or 1 only, the bracketed term in equation 1 has 2^M possible values, which are precomputed and stored in a LUT. The variable vector traverses through the input shift memory MSB first to address the LUT, whose contents are accumulated to obtain the outer sum of Eq. 1. For an N-bit x_k vector, the final result y is produced after N cycles.

Supposed the bitwidth of the variable vector **x** can be adjusted at single bit granularity. Truncating each trailing bit of **x** eliminates one shift, one table lookup, and one accumulator load. The truncated version can run at slower speed to obtain the same throughput, which also saves dynamic power. The precision degrades due to the increased input quantization noise. Our functional unit implements a 4-tap FIR filter using the VDP function. Fig.6 shows the power consumption vs. event recognition scalability of an FIR matched filter for a biomedical monitoring application using a flip-flop-based input shift memory. Power consumption can be reduced further by using the SRAM-based input shift memory described in Section 2.2.

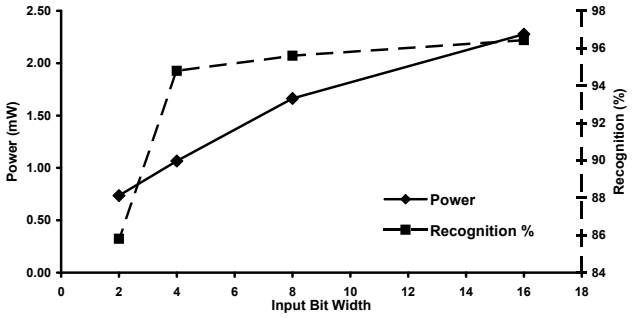

Fig. 6. VDP FIR filter realization with scalable input width.

3.2. Piecewise Linear Operation

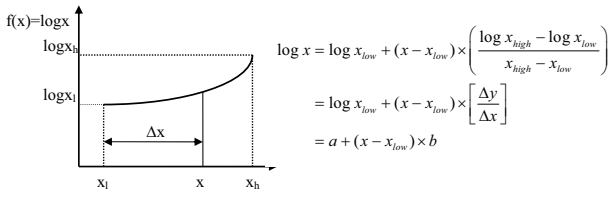

Fig. 7. LOG function linear approximation

A table lookup method can be used to implement any piecewise linear operation. We use the base-2 logarithm (LOG) as an example. The LUT is pre-loaded with a small set of log values for $x \in [1, 2)$ and at regular intervals $\Delta x = 2^{-4}$. To evaluate $log_2(x)$, the lower end point of the interval, $log_2(x_{low})$,

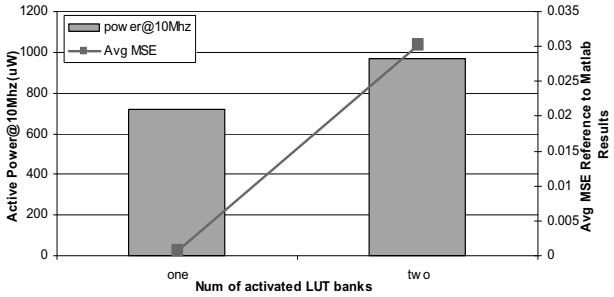

Fig. 8. LOG function power-quality tradoff

is first read from the LUT and Δx is approximated by interpolating between the two end points as shown in Fig. 7. Terms a and b are pre-computed and stored in the higher and lower LUT banks, respectively. The LOG function supports a 16-bit unsigned integer input x. The input is scaled down to Q.15 format to fit in the lookup range $\in [1, 2)$. The computation is derived below:

$$\begin{aligned} log_2(x) &= log_2(2^{15} \times 2^{-15} \times x) \\ &= 15 + log_2(x') \\ &= N + log_2(x') \\ &= int(logx) + frac(logx). \end{aligned}$$

The computable range is further increased by shifting out the leading zeros (equivalent to scaling up x). The integer part of $logx$ becomes $int(logx) = N - lznum$, where $lznum$ is the number of leading zeros in x.

Two levels of energy scalability can be obtained. At the coarse-grained level, disabling the LUT lower bank eliminates the linear approximation, which involves serial multiplication and addition. When linear approximation is activated, controlling the number of serial multiply iterations offers a fine-grained power-precision tradeoff. Fig.8 illustrates the coarse-grained energy scalability.

3.3. Serial Multiply

The signed serial multiplication (SMUL) is computed by iteratively executing a sequence of adds and shifts based on the value of the LSB of the multiplier (*mbit*). If $mbit = 1$, the multiplicand is added to the MSBs of the partial product and the resulting value is right-shifted by one bit with the sign bit preserved; if $mbit = 0$, the partial product is only right-shifted; at the very last iteration, if $mbit = 1$ (multiplier is negative), then the complemented multiplicand is added to the partial product and the sum is right-shifted to produce the final product. The multiplier bitwidth determines the number of iterations. Energy scalability is realized by treating the trailing bits of the multiplier as zeros regardless of their original values. Power is decreased due to the reduced number of operations for the zero $mbit$. Result precision is data-dependent because certain input values are more susceptible to the error introduced by energy scaled operation. For example, 1101×1100 would be more robust to energy scaling

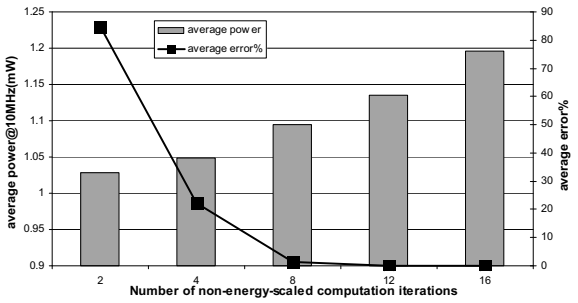

Fig. 9. SMUL average power vs. result quality.

than 1101×1011. Number representation also impacts result accuracy. Energy scaling affects the worst case multiplier error in Q.15 fractional representation much less than in integer representation. Fig. 9 shows the power-quality tradeoff as the number of iterations scales.

4. IMPLEMENTATION AND RESULTS

Power results presented here are based on post-layout simulation of a semicustom implementation in 0.25 μm CMOS. The SRAM-based input shift memory and LUT layouts are created manually. Remaining blocks are implemented with the OSU standard cell library[6]. Cadence Encounter is used to place and route the design (Fig.10) and the extracted netlist is simulated using Synopsys VCS-NanoSim.

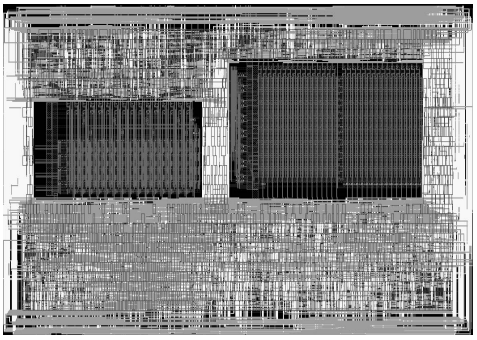

Fig. 10. Functional unit layout view.

Function	OperandBitWidth	AvgPower(mW)
add/sub	32-bit signed	1.163
complex add/sub	16-bit signed real and imag.	0.765
serial mulitply	16-bit signed	1.19575
complex multiply	16-bit signed fractional	4.399
division	16-bit signed	0.697
power function	16-bit fractional	2.7375
logarithm base 2	16-bit unsigned	0.824225
square root	16-bit unsigned	1.087225
vector dot product*	16-bit signed	0.152
*computation @39.1KHz, shiftout@10MHz; other functions @10MHz		

Fig. 11. Average power at 10 MHz.

Fig. 11 lists the average current for each function running at 10MHz and operating on random vectors. With the nominal power supply for the 0.25 μm CMOS process at 2.5 V, the functional unit consumes an average power of a few mW. The most power-hungry function is complex multiply because it involves four multiplications, two additions, and several operand swaps. Based on the device count and a frequency of 10 MHz, we estimate the active power consumption should be around 1 mW. Several factors may contribute to the excessive power consumption, for instance, the under-buffered interface signals between custom blocks and the standard cells and the suboptimal standard cell synthesis of the controller. With a full custom implementation, significant power reduction is expected.

5. CONCLUSION AND FUTURE WORK

We have described a functional unit which can compute several essential DSP functions in an energy scalable way. We have shown the architecture and circuit-level implementation of an SRAM-based input shift memory and lookup table. We have demonstrated the advantage of using serial arithmetic in energy scalable design by describing energy scalable implementations for several functions using serial algorithms. The simulated power for semi-custom implementation, while suboptimal, nevertheless demonstrates the concept of power scalable implementation. In future work, we will focus on controller optimization and transistor-level design of the remaining datapath blocks.

6. REFERENCES

[1] L. Guo, M. Scott, and R. Amirtharajah, "An energy scalable computational array for sensor signal processing," in *Proc. IEEE CICC Conference, San Jose*, 2006, pp. 317–320.

[2] R. Amirtharajah and A. P. Chandrakasan, "A micropower programmable DSP using approximate signal processing based on distributed arithmetic," *IEEE J. Solid State Circuits*, vol. 39, no. 2, pp. 337–347, 2004.

[3] Rajeevan Amirtharajah, Jeff Wenck, Jamie Collier, Jeff Siebert, and Bici Zhou, "Circuits for energy harvesting sensor signal processing," in *Proc. of the 43rd Design Automation Conference*, July 2006, pp. 639–44.

[4] Rajeevan Amirtharajah, Jamie Collier, Jeff Siebert, Bici Zhou, and A. Chandrakasan, "DSPs for energy harvesting sensors: Applications and architectures," *IEEE Pervasive Computing*, vol. 4, no. 3, pp. 72–9, 7-9 2005.

[5] S. A. White, "Applications of distributed arithmetic to digital signal processing: A tutorial review," *IEEE ASSP Magazine*, pp. 4–19, July 1989.

[6] J.E.Stine et. al., "A framework for high-level synthesis of system-on-chip designs," *Int'l. Conf. on Microelectronic Systems Education, IEEE Computer Society*, pp. 11–12, 2005.