The Design of a Reconfigurable Continuous-Flow Mixed-Radix FFT Processor

Anthony T. Jacobson, Dean N. Truong, and Bevan M. Baas
Department of Electrical and Computer Engineering
University of California, Davis

Abstract— **The design of a highly configurable continuous flow mixed-radix (CFMR) Fast Fourier Transform (FFT) processor is presented. It computes fixed-point complex FFTs and inverse FFTs (IFFTs), and utilizes a flexible addressing scheme to enable runtime configuration of the FFT length from 16-points to 4096-points. A configurable block floating point (BFP) unit increases numerical performance. Compared to a floating point Matlab FFT function, the accuracy of the proposed architecture is 80 dB for a 64-point FFT and 74 dB for a 1024-point FFT with random complex input data.**

I. Introduction

Applications such as 802.11a/g/n [1], and 802.16 (WiMAX) [2], which use OFDM modulation, require real-time high data throughput as well as increased flexibility to adapt to real life communication environments [3]. Moreover, digital audio/video broadcasting [4], [5], and asymmetrical digital/very-high-speed digital subscriber line (ADSL/VDSL) [6] standards have also adopted OFDM modulation [7]. Because OFDM makes heavy use of FFTs, efficient FFT implementations are becoming requirements in many DSP systems.

To achieve the high throughput necessary for these standards, pipelined FFT architectures have been proposed which rely on additional processing elements (PEs) in lieu of memory banks [8], [9]. Generally, this comes at the cost of increased area and power [10]. For larger FFTs, the additional number of PEs needed increases by a factor of $(r-1)*log_r(N)$ multipliers, where r is the radix, and N is the number of points of the FFT (i.e., size) [11].

With memory-based architectures, only a single radix-r processing element is present. It takes N/r iterations to complete a full N-point FFT. Thus, higher frequencies are necessary to process large FFT sizes while meeting the same throughput requirements, although the area is smaller when compared to pipelined architectures.

Memory architectures can have reduced data storage requirements through an "in-place" addressing scheme where output data are written to the same memory locations from where input data are read [12]. Furthermore, a *continuous flow* architecture improves the throughput of memory-based FFTs, where the PE is kept busy regardless of the memory contention between the PE and processor I/O. In-place addressing allows continuous flow architectures to need only two main memory banks, each of size N for a total of $2N$ complex memory locations. This architecture was first introduced using addresses that alternate between Decimation In Frequency (DIF) and Decimation In Time (DIT) FFTs [13]. An alternate in-place address generation method using modulo arithmetic has also been devised [14].

The primary computational unit of the FFT processing element is the butterfly, which performs both the complex multiplication of the data with a set of constants called "twiddle factors" (W_N), and the addition & subtraction of these products as seen in Fig. 1. The butterfly diagram in the figure depicts a radix-4 DIT butterfly. Because

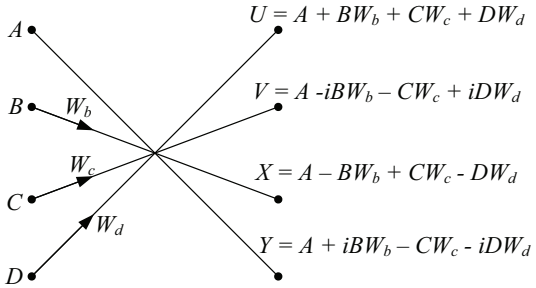

Fig. 1. A radix-4 Decimation in Time (DIT) butterfly

25% fewer complex multiplications are needed when performing radix-4 computations [15], recent efficient FFT implementations often use radix-4 butterflies rather than radix-2 to reduce the number of computational iterations for memory-based FFT architectures. Higher-radix PEs further reduce the number of iterations, and thus lower the number of computations and memory read and write accesses. On the other hand, PE area, address generation, and datapath control become large and complex, which limit practical designs to lower radices [11].

Radix-r implementations are limited to r^n-point FFTs, and therefore radix-2 computation must also be employed for a radix-4 processor to cover all 2^n-point FFTs. A mixed-radix FFT (MRFFT) architecture combines multiple radices—typically radix-2 and radix-4—and has been utilized in previous FFT processors [7], [14].

In this paper, we present a dedicated, runtime configurable complex FFT processor capable of performing variable-size FFTs, from 16- to 4096-point. It combines recent advancements to continuous flow memory-based FFT processors, with a dual-memory architecture and configurable addressing scheme. The FFT processor utilizes a single mixed radix-2/4 in-place butterfly, along with a symmetrically-reduced, twiddle-factor ROM. Section II provides an overview of the architecture. Details of the address generation and datapath control are given in Section III. The twiddle-factor ROM is presented in Section IV. Section V summarizes the performance results, and Section VI concludes the paper.

II. Architectural Overview

Figure 3 highlights the basic memory-based continuous flow architecture where data streams are multiplexed between the processor I/O and the PE whose core computational unit is the mixed-radix FFT butterfly. As an example, I/O communicating to Memory Bank 1 will block communication between the FFT PE and Memory Bank 1. The FFT PE reads and writes to Memory Bank 2 instead. When Memory Bank 1 is full of new data, the FFT PE can begin reading and working on the data stored there. The I/O is then blocked from writing to Memory Bank 1 and must now write to Memory Bank 2. The FFT processing element thus alternates between memory banks, and as long as new data are sent to the non-active memory bank (i.e.,

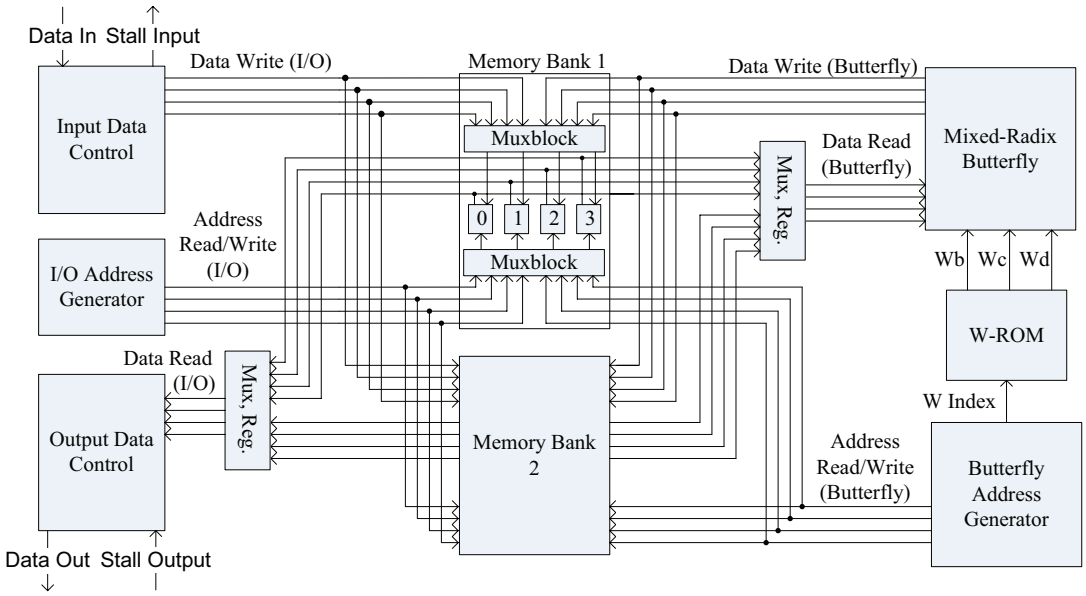

Fig. 2. FFT processor block diagram

the bank not being used by the PE), memory starvation is avoided and continuous operation of the PE occurs, maximizing the throughput of the FFT processor [16].

Notice that in order for the butterfly to read input data and write output data simultaneously, dual-port SRAMs are required. Similarly, the same argument can be said for the processor I/O. Continuous operation on a memory bank of only size N necessitates that the address generation and control guarantee that no address conflicts occur during simultaneous read/writes. The in-place address generation scheme adopted by the processor is further described in Section III.

For a radix-r butterfly, we require r memory subbanks to read and write data to and from the butterfly. As shown in Fig. 1, a radix-4 butterfly must read its four inputs A, B, C, and D from the memory bank while writing to memory its four outputs: U, V, X, and Y.

Our processor contains a single radix-4/-2 mixed-radix (MR) butterfly composed of three complex multipliers, 12 complex adders, and rounding logic. The butterfly input and output utilizes 16-bit real and 16-bit imaginary data, for a combined 32-bit complex FFT word width. The processor calculates FFTs varying in size from 16- to 4096-point. Thus, for FFTs of length $N = 2^r$ with r even (i.e., integer powers of 4), the butterfly performs standard radix-4 computations throughout the entire FFT. For r odd (i.e., N = 32, 128, 512, 2048) the butterfly functions as a radix-4 butterfly for the first $(r-1)/2$ stages, and then switches mode to a radix-2 butterfly for the final stage. The base radix-4 butterfly can be easily turned into a radix-2 butterfly if we take the following two equations from Fig. 1,

$$X = A - BW_b + CW_c - DW_d \tag{1}$$

$$Y = A + iBW_b - CW_c - iDW_d \tag{2}$$

and set inputs B and D to zero [15].

In theory, radix-4 computations like those shown in Eq. 1 and Eq. 2 can grow larger than a factor of four due to the possibility of the input data having a range outside the unit circle on the real-imaginary complex plane. The butterfly by default only shifts the output data by two, but the user can chose to configure the butterfly to be conservative and shift the output data by four. This, however, causes a potential reduction in accuracy. Over the course of several

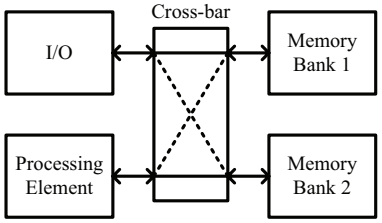

Fig. 3. A continuous flow memory based FFT processor architecture

thousand simulations with random data, overflow has not occurred, so the possibility of overflow will likely appear only under degenerate cases. If overflow does occur while in the default shift-by-2 mode, the data are then saturated accordingly.

The architecture also supports block floating point (BFP) tracking. Therefore, by grouping data to one "exponent," the maximum number of bits is preserved by shifting away the minimum number of redundant bits within that group of data. With BFP, the Signal to Quantization Noise Ratio (SQNR) of a 1024-point FFT can improve over 200%.

Figure 2 shows the final architecture of our FFT processor, which is composed of an I/O subgroup and a FFT subgroup, both centered around two memory banks. The I/O subgroup mainly contains the I/O controllers. The FFT half consists of the address generator, twiddle factor ROM (W-ROM), and the MR butterfly. Each memory bank is composed of simple mux logic and four 1024-word SRAM modules.

III. Address Generation and Datapath Control

A. Processor Input and Output Control

For an FFT, bit reversal must take place either within the FFT, or at the I/O for correct operation. Our architecture does bit reversal initially, which simplifies the output of processor data. For $N = 2^n$ where n is even, bit reversal is done as follows for a data word m:

$$m_{n-1}m_{n-2}...m_3m_2m_1m_0 \;\Rightarrow\; m_1m_0m_3m_2...m_{n-1}m_{n-2} \tag{3}$$

For r odd, where radix-2 computations are done at the final FFT stage, bit reversal is done as follows:

$$m_{n-1}m_{n-2}...m_4m_3m_2m_1m_0 \;\Rightarrow\; m_0m_2m_1m_4m_3...m_{n-1}m_{n-2} \tag{4}$$

TABLE I

FFT INPUT DATA INDEX ADDRESSING SCHEME, POST BIT REVERSAL

Data Index	Memory Subbank 0	Memory Subbank 1	Memory Subbank 2	Memory Subbank 3
0	0	1	2	3
1	7	4	5	6
2	10	11	8	9
3	13	14	15	12
4	19	16	17	18
5	22	23	20	21
6	25	26	27	24
⋮	⋮	⋮	⋮	⋮

An incrementer is used to identify each piece of data for bit reversal. For example, N data points will each have an assigned index chosen from 0 to $N-1$. Each data word m will be given the following index:

$$index_{Data} = m_{n-1}m_{n-2}\ \ldots\ m_3m_2\ m_1m_0 \tag{5}$$

The only difference between computing a FFT vs. an IFFT is that the incrementer increments the data index for an FFT, and decrements for an IFFT. Thus, the architecture is identical for both FFT and IFFT.

For continuous in-place FFT operation, a scheme for mapping addresses with data indices was developed (similar to one by Jo [7]). A portion of the address pattern this scheme generates is shown in Table I. Such an addressing ensures that, regardless of the current stage within the FFT, each of the four required input data will be accessed from different subbanks of memory.

To summarize, the resulting memory addresses are found as follows:

$$address_{Mem} = m_{n-1}m_{n-2}...m_3m_2 \tag{6}$$

The memory subbank is chosen using modulo arithmetic:

$$subbank_{Mem} = (m_{n-1}m_{n-2} + ... + m_3m_2 + m_1m_0) \mod 4 \tag{7}$$

Implementing both Eq. 6 and Eq. 7 in hardware is relatively simple.

Data is read out of the processor similarly. Since bit reversal has already taken place, address generation for data reads is identical to that of Eqs. 6 and 7.

B. FFT Address Generation

When a valid configuration word is received, the address generator is configured to operate on a user selected FFT/IFFT size. It is responsible for controlling memory read and writes to and from the butterfly, as well as provide the index needed to access the appropriate twiddle factor values from the W-ROM.

Each stage of an FFT has a unique group counter and butterfly counter. The values of these counters are derived from the value of a primary address. The first stage of an N-point FFT is composed of $N/4$ radix-4 butterflies within a single group; the second stage consists of $N/16$ radix-4 butterflies for each of the four groups; and so forth. The decomposition of the primary address into group and butterfly counters is shown in Table II.

Addresses rotate in groups of four after the first FFT stage, and butterfly inputs are addressed in such a way as to avoid memory access conflicts. To simplify address generation, a group of butterflies have a common prefix, which is assigned to a group number. The remainder of the address is given a base value for the first butterfly input with an offset added to the other butterfly inputs outside of the first stage. The address thus takes the form:

$$address_{Btfly} = \{group\ number, (base + offset)\} \tag{8}$$

TABLE II

DECOMPOSITION OF THE PRIMARY ADDRESS

Stage	Primary Address decomposition
0	$g_rg_{r-1}...g_1g_0$
1	$g_{r-2}...g_1g_0b_1b_0$
2	$g_{r-4}...g_0b_3...b_0$
⋮	⋮
x-2	$g_1g_0b_{s-2}...b_1b_0$
x-1	$b_sb_{s-1}...b_1b_0$

The offset added to each base value is determined by the memory subbank being accessed. This addressing scheme operates independently of the length of the FFT being performed. As the FFT progresses from one stage to the next, the appropriate adjustments to the length of the base values, offsets, and butterfly and group counters are made. Recall that the outputs of the butterfly are written to the same memory locations as the input data so the addresses for memory writes are the same as that for reads.

Besides input/output data addresses, the correct twiddle factor values have to be obtained for a particular butterfly computation. The twiddle factor ROM address is dependent on the butterfly count signal, and as a result is dependent on the current stage of the FFT. This address takes the form:

$$address_{W-ROM} = \{b_sb_{s-1}...b_1b_0, 000...\} \tag{9}$$

A slight modification to this address is required to obtain twiddle factors for a radix-2 computation.

IV. TWIDDLE-FACTOR ROM

Obtaining twiddle factors can be done through LUTs/ROMs or direct calculation. The CORDIC algorithm presents one efficient method of direct calculation which has been reported to be of lower power and area than ROM implementations [17]. However, a similar memory-based implementation utilizing CORDIC [17] with the same SRAM size as the processor reported here is approximately 5x larger when scaled to the same fabrication technology.

Theoretically, a 4096-point FFT/IFFT needs 4096 distinct complex twiddle factors. In practice the size of this ROM can be reduced considerably by manipulating the symmetry of the twiddle factors between 0 and 2π. A ROM storing the twiddle factors from 0 to $\pi/2$ was used by Chang [18]. Symmetry can be further exploited by only storing twiddle factor values from 0 to $\pi/4$ [19]. Thus, a ROM of 512 complex twiddle factors along with decoding logic is enough to compute a 4096-point FFT—our processor's maximum supported FFT configuration.

For a radix-4 butterfly, the twiddle factors W_b, W_c, and W_d, can be represented by Eq. 10, where $\theta_N = 2\pi/N$.

$$W_N^y = e^{-iy\theta_N} \tag{10}$$

W_c and W_d can be calculated via W_b through the following relationships [20]:

$$W_c = W_b^2 \tag{11}$$

$$W_d = W_b^3 \tag{12}$$

Based on these equations, the W_c ROM address is found simply by doubling the W_b address, and similarly W_d is found by tripling the W_b address, which correlates to squaring and cubing W_b itself, respectively.

When combined, these optimizations result in a ROM implementation that is only 3.4% of the total processor area.

TABLE III
PROCESSOR PERFORMANCE SUMMARY

FFT Size N	SQNR (dB)	Cyles per FFT
16	82.94	13
32	82.08	37
64	80.17	58
128	79.24	170
256	77.65	271
512	76.59	783
1024	74.13	1305
2048	72.54	3609
4096	71.90	6174

V. RESULTS

Compared to other architectures, the presented design is most similar to the one proposed by Jo [7]. The defining feature of our processor is its high SQNR, reconfigurability, and relatively high throughput. A reconfigurable "ring" FFT/IFFT processor obtains a high SQNR of 61 dB [21]; however, it needs 5280 cycles to complete a 1024-pt FFT. The FFT processor of [7] has a throughput of one 512-pt FFT per 640 cycles, and one 1024-pt FFT per 1280 cycles. For a 64-pt FFT it achieves an SQNR of 65 dB. Our implementation requires 783 and 1305 cycles while having a high SQNR of 76.59 and 74.13 for a 512-pt and 1024-pt FFT, respectively (see Table III). In contrast, fixed N high-speed architectures have achieved only a modest SQNR of 27 dB [22] and 32.7 dB [23].

This FFT processor was fabricated in 65 nm and occupies a 1 mm^2 area. Initial results show that at 866 MHz, it dissipates on average of 35 mW at 1.3 V [24]. The performance results are summarized in Table III. This processor has been successfully integrated into a 167-processor platform and has been used in software radio applications such as a 54 Mbps 802.11a receiver [25].

VI. CONCLUSION

This paper introduces a runtime configurable, continuous-flow, mixed-radix memory based FFT processor. FFTs and IFFTs of size 16- to 4096-point can be performed on 32-bit (16+16) fixed-point complex data. High throughput is achieved through an in-place radix-2/4 butterfly computational unit. While running at 866 MHz, the processor requires 1.5 μs to compute a 1024-pt complex FFT (thus achieving a throughput of over 680 Msamples/sec) while having an average SQNR of 74 dB.

ACKNOWLEDGMENTS

The authors thank the members of the VCL for helpful discussions and assistance. This work was supported by SRC GRC Grant 1598.001 and CSR Grant 1659.001, NSF Grant No. 0430090 and CAREER Award No. 0546907, ST Microelectronics, UC Micro, IntellaSys, Intel, and SEM.

REFERENCES

[1] M. K. A. Aziz, P. N. Fletcher, and A. R. Nix, "Performance analysis of IEEE 802.11n solutions combining MIMO architectures with iterative decoding and sub-optimal ML detection via MMSE and zero forcing GIS solutions," *IEEE Wireless Communications and Networking Conference, 2004*, vol. 3, pp. 1451–1456, Mar. 2004.

[2] A. Ghosh and D. R. Wolter, "Broadband wireless access with WiMax/802.16: current performance benchmarks and future potential," *IEEE Communications Magazine*, vol. 43, no. 2, pp. 129–136, Feb. 2005.

[3] S. Yoshizawa, K. Nishi, and Y. Miyanaga, "Reconfigurable two-dimensional pipeline FFT processor in OFDM cognitive radio systems," in *IEEE International Symposium on Circuits and Systems, (ISCAS '08)*, May 2008, pp. 1248–1251.

[4] B. Le Floch, R. Halbert-Lassalle, and D. Castelain, "Digital sound broadcasting to mobile receivers," *IEEE Transactions on Consumer Electronics*, vol. 35, no. 3, pp. 493–503, Aug. 1989.

[5] U. Reimers, "DVB-T: the COFDM-based system for terrestrial television," *Electronics and Communications Engineering Journal*, vol. 9, no. 1, pp. 28–32, Feb. 1997.

[6] J. A. C. Bingham, *ADSL, VDSL and Multicarrier Modulation*, John Wiley and Sons, Hoboken, NJ, 2000.

[7] B. G. Jo and M. H. Sunwoo, "New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, no. 5, pp. 911–919, May 2005.

[8] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," *International Symposium on Signals, Systems, and Electronics*., pp. 257–262, Oct. 1998.

[9] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," *IEEE Transactions on Signal Processing*, vol. 51, no. 3, pp. 864–874, Mar. 2003.

[10] W. Han, T. Arslan, A.T. Erdogan, and M. Hasan, "Multiplier-less based parallel-pipelined FFT architectures for wireless communication applications," in *IEEE International Conference on Acoustics, Speech, and Signal Processing, (ICASSP '05)*, 2005, vol. 5, pp. v45–v48.

[11] L. Jia, Y. Gao, and H. Tehunen, "A pipelined shared-memory architecture for FFT procesor," in *Proc. IEEE 42nd Midwest Symposium on Circuits and Systems*, 1999, pp. 804–807.

[12] L. G. Johnson, "Conflict free memory addressing for dedicated FFT hardware," *IEEE Transactions on Circuits and Systems*, vol. 39, no. 5, pp. 312–316, May 1992.

[13] R. Radhouane, P. Liu, and C. Modlin, "Minimizing the memory requirement for continuous flow FFT implementation: continuous flow mixed mode FFT (CFMM-FFT)," *IEEE International Symposium on Circuits and Systems*, vol. 1, pp. 116–119, May 2000.

[14] J. H. Baek, B. S. Son, B. G. Jo, M. H. Sunwoo, and S. K. Oh, "A continuous flow mixed-radix FFT architecture with an in-place algorithm," *International Symposium on Circuits and Systems 2003*., vol. 2, pp. 133–136, May 2003.

[15] B. M. Baas, *An Approach to Low-power, High-performance, Fast Fourier Transform Processor Design*, Ph.D. thesis, Stanford University, Stanford, CA, USA, 1999.

[16] B. M. Baas, "A low-power, high-performance 1024-point FFT processor," *IEEE Journal of Solid-State Circuits*, vol. 34, no. 3, pp. 380–387, Mar. 1999.

[17] T.-Y. Sung, "Memory-efficient and high-speed split-radix FFT/IFFT processor based on pipelined CORDIC rotations," *IEEE Proceedings on Vision, Image and Signal Processing*, vol. 153, no. 4, pp. 405–410, Aug. 2006.

[18] Y.-N. Chang and K. K. Parhi, "An efficient pipelined FFT architecture," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 6, pp. 322–325, June 2003.

[19] Y.-T. Lin, P.-Y. Tsai, and T.-D. Chiueh, "Low-power variable-length fast Fourier transform processor," *IEEE Proceedings on Computer and Digital Techniques*, vol. 152, no. 4, pp. 499–506, July 2005.

[20] L. R. Rabiner and B. Gold., *Theory and Application of Digital Signal Processing*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1975.

[21] G. Zhong, F. Xu, and A. Willson Jr., "A power-scalable reconfigurable FFT/IFFT IC based on a multi-processor ring," in *IEEE Journal of Solid-State Circuits (JSSC)*, Feb. 2006, vol. 41, pp. 483–495.

[22] H. Lee and M. Shin, "A high-speed low-complexity two-parallel radix-2^4 FFT/IFFT processor for UWB applications," in *IEEE Asian Solid-State Circuits Conference*, Nov. 2007, pp. 284–287.

[23] Y. Chen, Y.-C. Tsao, Y.-W. Lin, C.-H. Lin, and C.-Y. Lee, "An indexed-scaling pipelined FFT processor for OFDM-based WPAN applications," in *IEEE Transactions on Circuits and Systems II (TCAS)*, Feb. 2008, vol. 55, pp. 146–150.

[24] D. Truong, W. Cheng, T. Mohsenin, Z. Yu, T. Jacobson, G. Landge, M. Meeuwsen, C. Watnik, P. Mejia, A. Tran, J. Webb, E. Work, Z. Xiao, and B. M. Baas, "A 167-processor 65 nm computational array for highly-efficient DSP and embedded application processing," in *IEEE HotChips Symposium on High-Performance Chips*, Aug. 2008.

[25] A. Tran, D. Truong, and B. Baas, "A complete real-time 802.11a baseband receiver implemented on an array of programmable processors," in *Asilomar Conference on Signals, Systems and Computers*, Oct. 2008.