A High Performance Split-Radix FFT with Constant Geometry Architecture

Joyce Kwong, Manish Goel
Systems and Applications R&D Center
12500 TI Blvd
Dallas TX, USA
Email: {jkwong, goel}@ti.com

***Abstract*—High performance hardware FFTs have numerous applications in instrumentation and communication systems. This paper describes a new parallel FFT architecture which combines the split-radix algorithm with a constant geometry interconnect structure. The split-radix algorithm is known to have lower multiplicative complexity than both radix-2 and radix-4 algorithms. However, it conventionally involves an "L-shaped" butterfly datapath whose irregular shape has uneven latencies and makes scheduling difficult. This work proposes a split-radix datapath that avoids the L-shape. With this, the split-radix algorithm can be mapped onto a constant geometry interconnect structure in which the wiring in each FFT stage is identical, resulting in low multiplexing overhead. Further, we exploit the lower arithmetic complexity of split-radix to lower dynamic power, by gating the multipliers during trivial multiplications. The proposed FFT achieves 46% lower power than a parallel radix-4 design at 4.5GS/s when computing a 128-point real-valued transform.**

I. Introduction

Very high performance hardware FFTs have applications in a number of areas such as spectrum analyzers and communication systems using orthogonal frequency-division multiplexing (OFDM). The FFT and IFFT blocks are often one of the most computationally intensive portions of such systems. Due to the demand for higher processing speeds and data rates, there is a need for FFT processors supporting high sample throughput while achieving good energy efficiency.

Given the importance of the FFT, there has been much previous work in FFT processing. In the following discussion we focus on custom hardware solutions, since software-based solutions on DSPs or ASIPs often cannot achieve the highest level of throughput. Recently published high throughput hardware FFTs include [1], [2], and [3], which utilize multipath delay feedback (MDF) architecture, with 4 or 8 parallel pipelines, to achieve throughputs of 1GS/s, 2.4GS/s, and 2.8GS/s respectively. These were targeted for OFDM-based UWB and wireless personal area network (WPAN) communication. An 8-parallel multimode MDF pipeline architecture is also presented in [4] and achieves 2.4GS/s for MIMO OFDM application. Dynamic voltage and frequency scaling is employed to optimize power. The work of [5] utilizes multiple memory banks along with parallel pipelined processing elements to achieve 2.5GS/s throughput.

978-3-9810801-8-6/DATE12/©2012 EDAA

Previous work has mostly focused on pipelined FFT architectures with one or multiple pipelines, each consisting of $\log_r N$ stages of butterfly datapaths (where N and r are the FFT size and radix). FIFOs or memory, along with the associated control/periphery logic, are needed at each pipeline stage to store intermediate results. In contrast, this paper describes a parallel architecture with N/r butterfly datapaths. A constant geometry algorithm is employed such that the wiring between butterflies is the same for each stage of the FFT, reducing multiplexing overhead. Constant geometry algorithms are well-known for radix-2^n FFTs [6]. Our contribution consists of mapping the split-radix FFT algorithm [7] to a constant geometry architecture, and leveraging the lower multiplicative complexity of the algorithm to reduce dynamic power.

Section II of this paper provides background on constant geometry architectures and FFT algorithms. Section III describes the proposed FFT design, while Section IV quantifies the area and power savings of the proposed technique.

II. Parallel FFT Architectures

To aid understanding of the proposed design, we first give a brief overview of relevant FFT architectures and algorithms. The FFT consists of a number of basic computations called a butterfly. The radix refers to the number of inputs processed at a time by the butterfly, and Figure 1 shows block diagrams of radix-2 and radix-4 butterflies. An N-point, radix-r complex-valued FFT contains $\frac{N}{r}\log_r N$ butterfly operations, hence the throughput of an FFT architecture is limited by the number of butterfly operations executed in parallel.

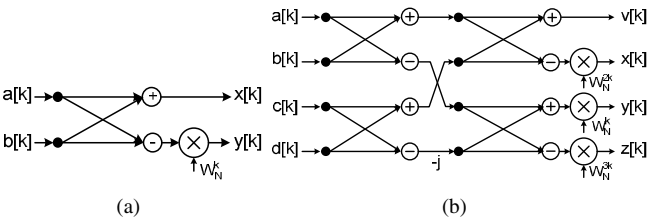
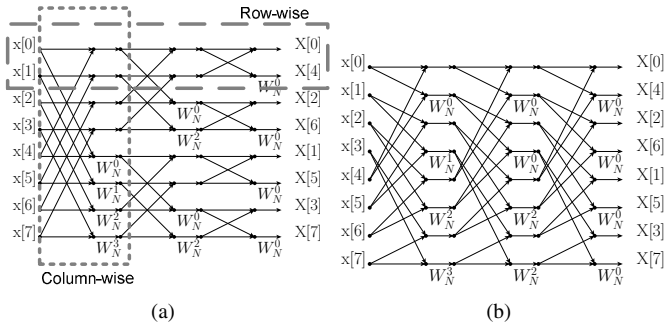


Fig. 1. Block diagram of (a) radix-2 and (b) radix-4 butterfly.

For illustration, the signal flow graph (SFG) of an 8-point, decimation-in-frequency (DIF) FFT is shown in Figure 2(a). Pipelined FFT designs employ $log_r N$ datapaths to compute



(a) (b)

Fig. 2. Signal flow graph (SFG) of an 8-point FFT using (a) Cooley-Tukey and (b) constant geometry algorithms. Dashed boxes in (a) indicate computations performed in parallel datapaths in row- and column-wise parallel architectures.

one row of the SFG, with memory elements at each stage to store the butterfly inputs and outputs and ensure that they enter the next stage in the correct order. Typically in high throughput designs such as [1]-[4], multiple pipelines are used to speed up the FFT computation.

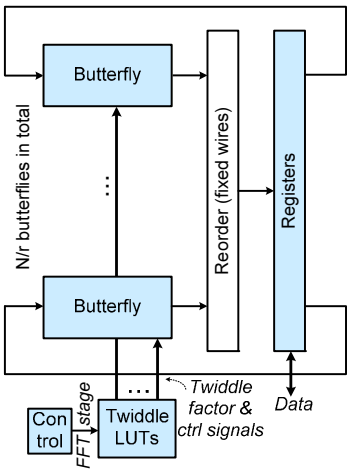


Fig. 3. Parallel constant geometry FFT architecture.

Instead of the row-wise parallelization in pipelined FFTs, it is also possible to parallelize along a column of the SFG as shown in Figure 2(a). In other words, N/r butterflies are instantiated to compute one stage (i.e. one column) of the FFT at once, and one column of registers is used to register their outputs. In the classic Cooley-Tukey algorithms such as in Figure 2(a), values from different registers must be multiplexed into the butterfly datapaths depending on the FFT stage (e.g. the top butterfly requires $x[0], x[4]$ in the first stage and $x[0], x[2]$ in the second). The resulting area and delay overhead imposed by these multiplexers is undesirable for high-throughput designs. On the other hand, constant geometry algorithms avoid this overhead since they have the same geometry in each stage, as shown in Figure 2(b). This maps to the hardware realization in Figure 3, where butterfly outputs are reordered by fixed wiring and then registered. In addition to radix-2, constant geometry algorithms for other power-of-2 radices can be found in [8].

III. Split-radix Constant Geometry FFT Design

A. Mapping the split-radix algorithm

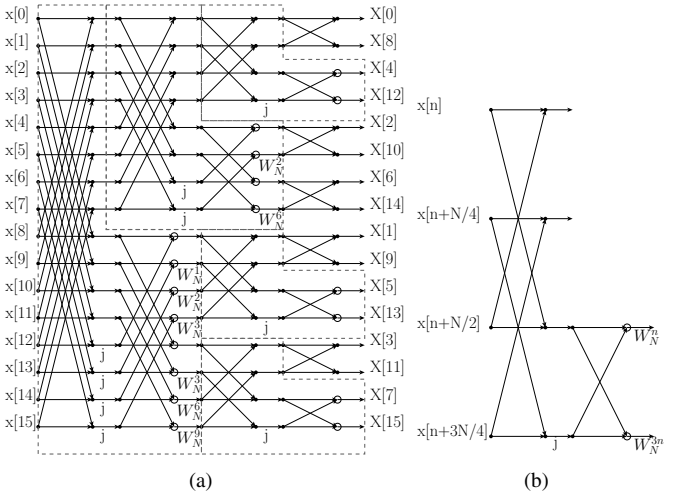
In the proposed FFT, we map a split-radix algorithm onto a constant geometry interconnect structure. As will be shown in Section IV, this has the benefits of lower power, smaller area, and shorter critical path than a radix-4 constant geometry architecture. The split-radix algorithm was proposed in [7], and provides the advantage of fewer non-trivial complex multiplications and additions than radix-2, radix-4, and radix-8 algorithms. The split-radix algorithm is based on successively decomposing an N-point DFT into an $N/2$-point DFT and two $N/4$-point DFTs as follows:

$$X_{2k} = \sum_{n=0}^{\frac{N}{2}-1}(x_n + x_{n+N/2})W_N^{2nk} \quad (1)$$

$$X_{4k+1} = \sum_{n=0}^{N/4-1}[(x_n - x_{n+N/2}) - j(x_{n+N/4} - x_{n+3N/4})]W_N^n W_N^{4nk} \quad (2)$$

$$X_{4k+3} = \sum_{n=0}^{N/4-1}[(x_n - x_{n+N/2}) + j(x_{n+N/4} - x_{n+3N/4})]W_N^{3n} W_N^{4nk} \quad (3)$$

where $W_N^k = e^{-j2\pi k/N}$ and will be referred to as twiddle factors.



(a) (b)

Fig. 4. (a) Signal flow graph of 16-point split-radix FFT. (b) L-shaped butterfly. In both figures, hollow circles indicate the end of the second stage of the L-shape.

The signal flow graph of a 16-point split-radix FFT is shown in Figure 4(a). The basic unit of the split-radix algorithm is an L-shaped butterfly shown in Figure 4(b). The dashed lines on Figure 4(a) indicate how the FFT is composed of L-shaped butterfly operations (along with some radix-2 operations at the end). It is clear that the scheduling of the L-shaped butterflies is irregular, and hence the split-radix algorithm is less commonly implemented than radix-2^k algorithms.

One previous work [9] has proposed a constant geometry version of the split-radix algorithm. The work in [9] assumes that the 4 outputs of the L-shaped butterfly are available with the same latency. The resulting scheduling is shown in Figure

5. However, this approach is not well suited to high throughput applications. Compared to a parallel radix-4 design, the lower part of the L-shaped butterfly has the same critical path delay, but a parallel radix-4 design takes only $log_4 N$ cycles to complete, whereas the approach in [9] takes $log_2 N$ cycles (or 1 fewer with rescheduling). Consequently, the approach will generally have lower throughput than a parallel radix-4 design.

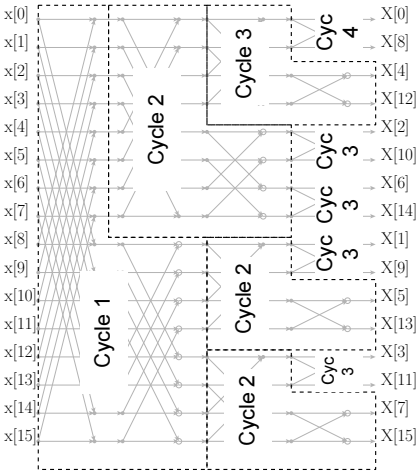


Fig. 5. Scheduling of a 16-point split-radix FFT from [9] using L-shaped butterflies. The annotations indicate the clock cycle in which the butterflies would be computed.

In high-throughput FFTs, it is desirable to avoid the L-shaped butterfly while still map the algorithm to a constant geometry structure. We propose using a pair of radix-2-like butterflies as the basic processing unit. As a result, the split-radix SFG in Figure 4(a) can be thought of as a Cooley-Tukey radix-2 SFG, except with different twiddle factors and details in the butterflies. By extension, the split-radix SFG can be rearranged in the same way that a radix-2 Cooley-Tukey algorithm is changed into a constant geometry algorithm (i.e. the reordering of Figure 2(a) to obtain Figure 2(b)). After this reordering, the SFG of a 16-point split-radix algorithm with constant geometry interconnect is shown in Figure 6.

B. Hardware Implementation

The signal flow graph in Figure 6 can be mapped to the hardware architecture in Figure 3, where $N/2$ butterfly datapaths compute one stage (column) of the SFG per clock cycle. To illustrate, the third butterfly is highlighted in black in the first and third stages, while the fourth butterfly is highlighted in the second and fourth stages. Note that the odd butterflies do not need any multipliers, while the even butterflies requires two due to the presence of two twiddle factors. We will refer to these butterflies as Type-1 and Type-2 respectively.

The type-1 and type-2 butterflies differ from radix-2 butterflies in two main ways. First, when the butterfly is used to compute the second half of the 4-point DFT (i.e. lower right corner in Figure 4(b)), a subtraction occurs in the top branch rather than the bottom branch, as per Equation 2 and Equation

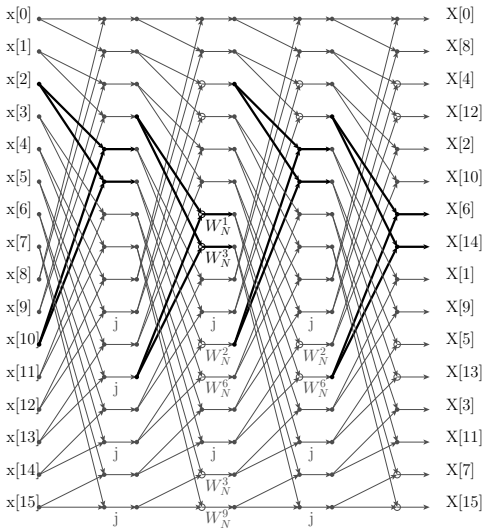


Fig. 6. Rearrangement of a 16-point split-radix algorithm to obtain constant geometry interconnect. The 3^{rd} butterfly is highlighted in black in the 1^{st} and 3^{rd} stages, while the 4^{th} butterfly is highlighted in the 2^{nd} and 4^{th} stages.

3. Second, butterflies are modified to perform multiplication by $\sqrt{-1}$ (j) without having to use the complex multiplier. This is particularly important in the split-radix design because the algorithm decreases the number of non-trivial multiplications at a cost of more multiplications by j. Table I lists the number of multiplications by non-trivial twiddle factors and by j (abbreviated as M_{nt} and M_j) of three algorithms. M_{tot} indicates the total operations performed by complex multipliers. Note that in radix-2 and radix-4 designs, if the butterflies in Figure 1 are used without modification, multiplication by j still involves a complex multiplier. Therefore, in Table I $M_{tot} = M_j + M_{nt}$ for radix-2 and radix-4. The proposed split-radix butterflies do not use multipliers for $\times j$ so $M_{tot} = M_{nt}$. It is seen that the multiplicative complexity of split-radix is much lower than radix-2 and slightly lower than radix-4, and the difference grows when special handling for $\times j$ is taken into account.

Figure 7 shows block diagrams of the proposed split-radix butterflies, with real and imaginary parts shown separately. mj and $R4$ are control signals indicating the two special cases described above respectively. The time and location where trivial multiplications occur are known at design time from the FFT algorithm, hence the sequence of control signals can be hard-coded via small look-up tables, one for each butterfly. The shaded blocks in Figure 7 will be discussed in Section III-C.

Besides the number of operations, the number of instantiated hardware *multipliers* is also important. Table II lists the butterfly datapaths and complex multipliers contained in constant geometry column-parallel FFTs for various algorithms. We account for the fact that the topmost butterfly needs fewer multipliers. It is seen that split-radix requires slightly fewer hardware multipliers than radix-2 and radix-4. However, utilization of these multipliers is substantially lower in split-radix, which can provide power savings as described below.

TABLE I

MULTIPLICATIONS IN DIFFERENT FFT ALGORITHMS FOR VARIOUS FFT SIZES (N). M_j, M_{nt}, AND M_{tot} RESPECTIVELY DENOTE MULTIPLIES BY $\sqrt{-1}$, BY NON-TRIVIAL TWIDDLE FACTORS, AND TOTAL OPERATIONS PERFORMED ON COMPLEX MULTIPLIERS.

	Radix-2			Radix-4			Split-Radix		
N	M_j	M_{nt}	M_{tot}	M_j	M_{nt}	M_{tot}	M_j	M_{nt}	M_{tot}
32	15	34	49		N/A		23	26	26
64	31	98	129	5	76	81	57	72	72
128	63	258	321		N/A		135	186	186
256	127	642	769	21	492	513	313	456	456

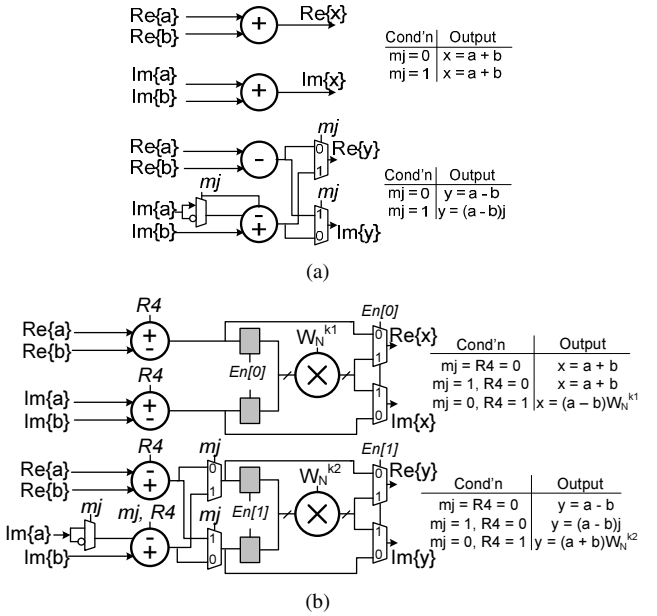


Fig. 7. Two types of butterflies in the proposed FFT. (a) Type-1 with no complex multipliers, and (b) Type-2 with two complex multipliers.

TABLE II

NUMBER OF HARDWARE DATAPATHS AND MULTIPLIERS NEEDED FOR A COLUMN-PARALLEL N-POINT FFT WITH CONSTANT GEOMETRY.

Algorithm	Butterfly Datapaths	Complex Multipliers
Radix-2	$N/2$	$N/2-1$
Split-Radix	$N/2$	$N/2-2$
Radix-4	$N/4$	$3N/4-2$

C. Multiplier Gating for Switching Power Reduction

The lower multiplicative complexity of split-radix can be leveraged to reduce power. Specifically, during trivial multiplications (i.e. multiplying by 1 or $\sqrt{-1}$), we can bypass the complex multiplier in the butterfly datapath. More importantly, during bypass, the multiplier inputs should be held at their values from the previous cycle in order to suppress switching. At one multiplier input, this is done by inserting a latch as shown by the gray boxes in Figure 7(b). When the multiplier is bypassed, the latch holds data from the previous cycle, otherwise the latch is transparent. Like the other control signals, the latch enable signals can be controlled via small, hard-coded lookup tables. The latches can be included during timing verification by forcing the enable signal to "1", in which case they add a small delay to the logic path.

The other multiplier input is controlled by hard-coded twiddle factor lookup tables. Therefore, we can modify the tables to replace "1" or "$\sqrt{-1}$" values with twiddle factor values from the previous cycle. The net result is that during trivial multiplications, the multiplier output remains unchanged from the previous cycle, but this incorrect output is bypassed.

D. Extending to Real-Valued FFT

The proposed architecture can be extended to efficiently compute FFTs on real-valued inputs using the well-known approach described in [10]. In this approach, the even and odd samples of an N-point real sequence are used to form the real and imaginary parts of an $N/2$-point complex sequence. Then, an $N/2$-point complex-valued FFT is computed using the split-radix algorithm as described above. This is followed by real-valued post-processing [11]:

$$A[k] = (Z[k] + Z^*[N/2-k]) \quad (4)$$

$$B[k] = (Z[k] - Z^*[N/2-k])W_N^k \quad (5)$$

$$X[k] = \tfrac{1}{2}[(A_r[k] + B_i[k]) + j(A_i[k] - B_r[k])] \quad (6)$$

$$X[N/2-k] = \tfrac{1}{2}[(A_r[k] - B_i[k]) - j(A_i[k] + B_r[k])] \quad (7)$$

where $Z[k]$ are results of the $N/2$-point complex-valued FFT, and $X[k]$ (with conjugate symmetry) are results of the N-point real-valued FFT. Normally in a radix-2 FFT, $A[k]$ and $B[k]$ can be computed by a DIF butterfly with one complex multiplier in the bottom branch. Recall that in the split-radix architecture, the type-2 butterfly contains 2 complex multipliers while type-1 contains none. However, the type-1 butterfly can be used to compute $A[k]$ and $A[k+1]$ while the type-2 computes $B[k]$ and $B[k+1]$, so the existing hardware can be fully utilized. The hardware configuration for real-valued post-processing is illustrated in Figure 8, where the existing hardware is reused to find A and B. Computation of $X[k]$ and $X[N/2-k]$ are the same in both radix-2 and split-radix designs. In a parallel high-throughput architecture, dedicated adders can be included and only enabled during the post-processing stage, as indicated by the rightmost blocks in Figure 8.

IV. IMPLEMENTATION RESULTS AND DISCUSSION

Three FFTs using parallel constant geometry architecture were implemented: a baseline radix-4 (R4), radix-4 with the multiplier gating technique (R4MG), and split-radix with multiplier gating (SRMG). To satisfy needs of the end application, the designs implement a transform from 64 complex values to 128 real values using the real-valued post-processing hardware described in Section III-D. The datapath width is 12 bits.

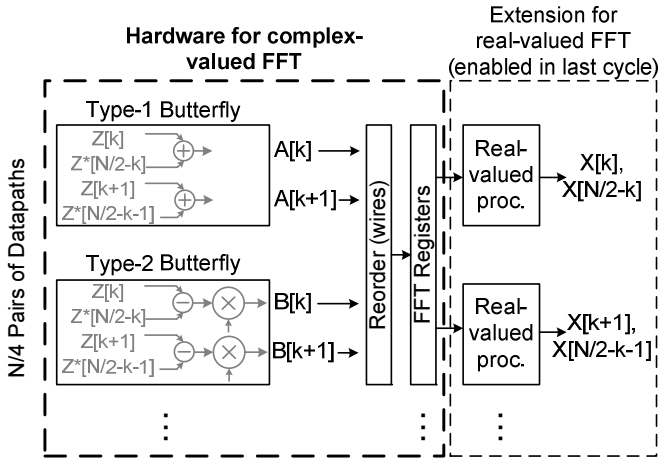


Fig. 8. Hardware configuration for real-valued FFT (to handle real-valued input sequences). Blocks in the left dashed box are standard components for split-radix complex-valued FFTs.

The three FFTs were synthesized and laid out using a 28nm standard cell library at the weak corner. Figure 9 shows the layout of the split-radix FFT. Then, power consumption at the strong process corner is obtained by doing a gate-level simulation of the post-layout netlist, and using the captured switching activity (i.e. value change dump vectors) in the power simulation tool Primetime PX.

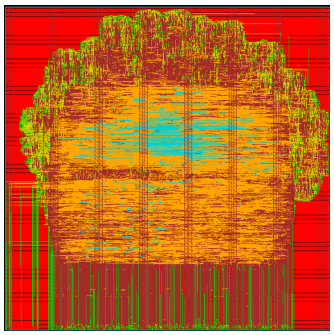
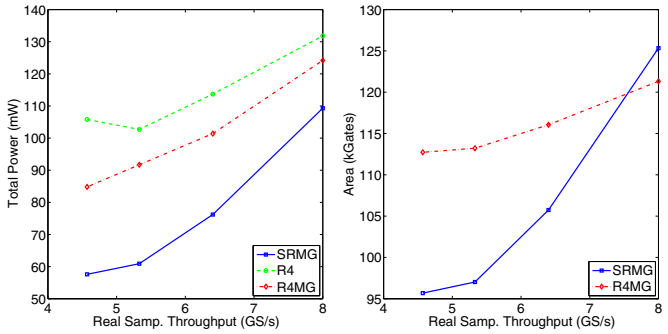


Fig. 9. Layout of split-radix FFT.

Figure 10(a) plots the total power (including switching and leakage) of the three FFTs across throughput in G-Samples/sec (real-valued output samples). The multiplier gating technique reduces the radix-4 FFT power by approximately 10%, but its full benefits are seen in the split-radix FFT. The split-radix (SRMG) design consumes substantially less power than the radix-4 FFT until at very high throughputs. The power reduction can be attributed to two factors. First, complex multipliers in SRMG have lower switching activity due to multiplier gating used in conjunction with the lower complexity of the split-radix algorithm. Second, the split-radix datapath is shorter than the radix-4 datapath, which implies that glitches propagate through fewer levels of logic.

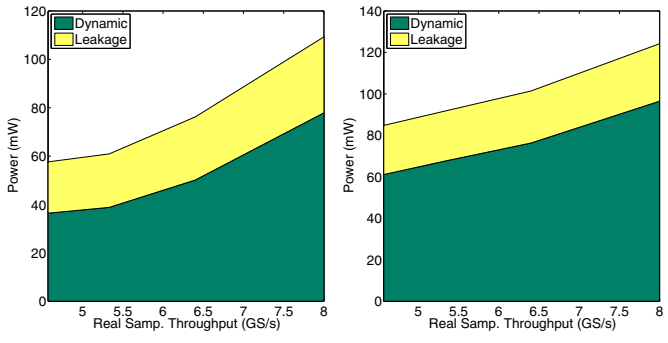
A slight power increase is observed in the baseline R4 design from 5GS/s to 4.5GS/s in Figure 10(a). After examining the circuit netlists, we attribute this to different adder architectures selected by the synthesis tool. At high frequencies, faster tree structures are used, while at lower frequencies, ripple carry structures are used. However, from a power point of view, ripple carry structures exhibit more glitches than tree structures, which cause an appreciable power increase since the area and power are dominated by arithmetic circuits in parallel FFTs.



(a) (b)

Fig. 10. (a) Total power and (b) Area vs. sample throughput in split-radix with multiplier gating (SRMG), radix-4 (R4), R4 with multiplier gating (R4MG). Simulated from post-layout netlist with wiring parasitics at strong PVT corner.

Area of the SRMG and R4MG designs is reported in Figure 10(b). The area is obtained from the post-layout netlist which includes clock tree buffers and repeaters inserted during place and route, and normalized to the smallest NAND2 gate in the library. Below 7GS/s throughput, the parallel split-radix design provides an area advantage over radix-4 since it requires fewer instantiations of complex multipliers. At higher throughputs, large standard cells are used in the critical path to meet clock frequency constraints. The split-radix design reaches this point of rapid area increase sooner than the radix-4 design because the former requires two extra multiplexers in the critical path to realize the more irregular algorithm. The area overhead of latches and multiplexers for multiplier gating is approximately 5% of the total area.



(a) (b)

Fig. 11. Dynamic and leakage power breakdown vs. sample throughput of (a) split-radix with multiplier gating and (b) radix-4 with multiplier gating.

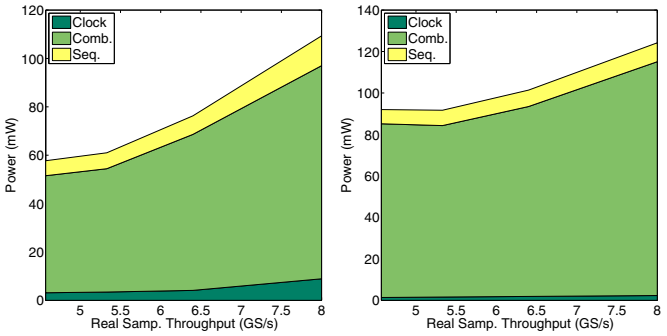
Figure 11 shows the breakdown between dynamic and leakage power of SRMG and R4MG. Note that the power simulation was performed at the strong process and temperature corner, but dynamic power (as opposed to leakage) is still

TABLE III
SUMMARY OF PREVIOUS HIGH THROUGHPUT FFTS AND PROPOSED DESIGN. NOTE THAT THE ENERGY/SAMPLE IS NOT NORMALIZED TO PROCESS TECHNOLOGY. REFERENCES WITH ASTERISK PRESENTED SILICON MEASUREMENTS.

	[1]*	[2]*	[4]*	[5]	Proposed
Datapath Width (bits)	10	8	10	12	12
FFT size	128	128	256	512	128
Architecture	4x MDF pipeline	8x MDF pipeline	8x MDF pipeline	parallel mem.-based	column-wise parallel
Throughput (GS/s)	1	0.4096	2.4	2.59	5.33
Area / Norm. Area to 28nm (mm^2) $^{\diamond}$	2.67 / 0.0833	0.452 / 0.0565	2.23 / 0.279	2.46 / 0.308	0.152 / 0.152
Process (nm)	180	90	90	90	28
Power (mW)	175	6.8	119.7	103.5	60.9
Energy/samp. (pJ)	175	16.6	49.9	40.0	11.4

$^{\diamond}$ Area is normalized to 28nm assuming 2× area decrease per process node. For [1]-[4] area of test circuitry is excluded.
• Comparison with [9] is not available since it focused on algorithms and did not include implementation results.

the dominant contributor. Therefore it is beneficial to reduce multiplier utilization in order to save dynamic power. Figure 12 shows the breakdown between the clock tree, sequential circuit and combinational circuit power. The combinational circuits (i.e. from the datapaths) dominate the total power, indicating that most of the power is efficiently spent on actual computation.



(a) (b)

Fig. 12. Clock tree, combinational and sequential power vs. sample throughput of (a) split-radix with multiplier gating and (b) radix-4 with multiplier gating.

Details of recently published high throughput FFTs are listed in Table III along with the proposed design. Note that a direct energy comparison is difficult due to different technologies, FFT sizes and datapath widths. In addition, some cited work presented silicon results, although we attempt to provide realistic simulations results by including clock and wiring power post-layout, and simulating at the strong process, voltage and temperature corner. With this in mind, it can be seen from Table III that the proposed FFT is competitive with previous work in terms of throughput and energy efficiency.

V. CONCLUSION

In this paper we have presented a new FFT architecture based on mapping the split-radix algorithm to a parallel constant geometry structure. The conventional "L-shaped" split-radix datapath has uneven latencies and is thus not suited for high throughput operation. Instead, this work proposes a pair of radix-2-like datapaths with shorter latencies. Using these datapaths, the split-radix signal flow graph can be reorganized to have a constant geometry structure. The proposed architecture enabled power reduction in two ways. First, it achieves lower multiplicative complexity than radix-4 (and radix-2) algorithms while using a shorter datapath than radix-4; a shorter datapath reduces glitch power which is prevalent in large arithmetic circuits. Second, the split-radix algorithm involves many trivial multiplications, during which the complex multipliers can be gated to save dynamic power. The proposed FFT achieves 46% lower power than a parallel radix-4 design at 4.5GS/s when computing a 128-point real-valued transform.

ACKNOWLEDGMENT

The authors thank Arthur Redfern and Raul Blázquez for helpful discussion.

REFERENCES

[1] Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee, "A 1-GS/s FFT/IFFT processor for UWB applications," *IEEE Journal of Solid-State Circuits*, vol. 40, no. 8, pp. 1726 – 1735, Aug. 2005.
[2] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT Processor for OFDM-Based WPAN Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 6, pp. 451–455, June 2010.
[3] T. Cho, H. Lee, J. Park, and C. Park, "A high-speed low-complexity modified radix-2^5 FFT processor for gigabit WPAN applications," in *IEEE International Symposium on Circuits and Systems*, May 2011, pp. 1259–1262.
[4] Y. Chen, Y.-W. Lin, Y.-C. Tsao, and C.-Y. Lee, "A 2.4-Gsample/s DVFS FFT processor for MIMO OFDM communication systems," *IEEE Journal of Solid-State Circuits*, vol. 43, no. 5, pp. 1260–1273, May 2008.
[5] S.-J. Huang and S.-G. Chen, "A green FFT processor with 2.5-GS/s for IEEE 802.15.3c (WPANs)," in *International Conference on Green Circuits and Systems*, June 2010, pp. 9 –13.
[6] M. C. Pease, "An adaptation of the Fast Fourier Transform for parallel processing," *Journal of the ACM*, vol. 15, pp. 252–264, April 1968.
[7] P. Duhamel and H. Hollmann, "'Split radix' FFT algorithm," *Electronics Letters*, vol. 20, no. 1, pp. 14 –16, May 1984.
[8] M. Corinthios, "The design of a class of Fast Fourier Transform computers," *IEEE Transactions on Computers*, vol. C-20, no. 6, pp. 617–623, June 1971.
[9] F. Argüello and E. Zapata, "Constant geometry split-radix algorithms," *Journal of VLSI Signal Processing*, 1995.
[10] H. Sorensen, D. Jones, M. Heideman, and C. Burrus, "Real-valued Fast Fourier Transform algorithms," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 35, no. 6, pp. 849–863, June 1987.
[11] R. Matusiak. (2001, Aug.) Implementing Fast Fourier Transform algorithms of real-valued sequences with the TMS320 DSP platform. [Online]. Available: http://focus.ti.com/lit/an/spra291/spra291.pdf