A GENERATOR OF MEMORY-BASED, RUNTIME-RECONFIGURABLE $2^N3^M5^K$ FFT ENGINES

Angie Wang, Jonathan Bachrach, Borivoje Nikolić

University of California, Berkeley

ABSTRACT

Runtime-reconfigurable, mixed-radix FFT/IFFT engines are essential for modern wireless communication systems. To comply with varying standards requirements, these engines are customized for each modem. The Chisel hardware construction language has been used in this work to create a generator of runtime-reconfigurable $2^n3^m5^k$ FFT engines targeting software-defined radios (SDR) for modern communications, but with flexibility to support a wide range of applications. The generator uses a conflict-free, in-place, multi-bank SRAM design, and exploits the duality of decimation-in-frequency (DIF) and decimation-in-time (DIT) FFTs to support continuous data flow with only 2N memory blocks. DFT decomposition using the prime-factor algorithm (PFA) followed by the Cooley-Tukey algorithm (CTA) reduces twiddle ROM sizes. A programmable Winograd's Fourier Transform (WFTA) butterfly supporting radix-2/3/4/5/7 operations reuses radix-7 hardware to support reconfigurability with minimal area penalty. The generated FFTs use 50% less memory than iterative FFTs from Spiral. The twiddle ROM size of the generated LTE/WiFi FFT engine is 16% smaller than that of a 2048-pt Spiral design.

Index Terms— Fast Fourier Transform, Cooley-Tukey, Winograd's Fourier Transform, Prime Factor Algorithm, Reconfigurable Hardware Generator.

1. INTRODUCTION

OFDM-based wireless transceivers need reconfigurable IFFT/FFT engines to support a multitude of channel bandwidths and modulation schemes. The need for adaptation to a wide range of channel bandwidths requires support for many different FFT sizes, which are often not decomposable to 2^n, making reuse of these blocks difficult. For example, in order for an SDR to support both WiFi and LTE with single-carrier frequency-division multiple access (SC-FDMA), the maximum powers of (2, 3, 5) that must be supported are (2048, 243, 25) [1]. The need for a plethora of FFT engines has been addressed by building commercial reconfigurable, application-specific FFT cores. However, they are generally not resource efficient, which is problematic when users only need a subset of FFT sizes and are otherwise severely hardware constrained. Additionally, they cannot be easily modified to support new standards. The terrestrial DTV system in China, for example, requires a 3780-pt FFT and a radix-7 butterfly [2],[21]. A step forward is the development of FFT generators; however, all generators available in the open literature are of fixed size and not applicable to SDR.

FFT generators such as Spiral exploit design regularity, enabling usage in diverse applications with ideally no design overhead [3],[20]. Highly parameterizable generators also enable rapid and extensive design exploration, allowing throughput/area tradeoffs [17], memory access methods, fixed point optimizations (as in [4]), etc. to be studied in detail. However, no generator in open literature supports all these requirements. Thus, we have designed a generator of memory-based, runtime-reconfigurable $2^n3^m5^k$ FFT engines. The design is hardware-optimized: it relies on butterfly reuse, supports continuous data flow with 2N memory blocks, and stores fewer twiddle factors than designs from state-of-the-art generators like Spiral.

2. DESIGN METHODOLOGY

2.1. Memory-Based Architecture

Hardware FFT designs often use either in-place memory-based or pipelined architectures. For large FFT sizes, memory-based designs are smaller in area—usually relying on a single iterating butterfly—but support lower throughput [2]. Pipelined designs use more processing elements to improve throughput and more easily support continuous data flow [16], but often overprovision butterflies (i.e. $\log_2$N radix-2 butterflies for 2^n-pt FFTs) to achieve this.

Memory-based designs with conflict-free scheduling schemes ease area/throughput tradeoffs and can compute an N-pt FFT in

$$C = SP + \sum_{i=0}^{S-1}\lceil\frac{N}{r_iB}\rceil \quad (1)$$

clock cycles, where B is the # of parallel butterflies, r_i is the radix at the i^{th} calculation stage, S is the # of stages, and P represents the pipelining between memory accesses (butterfly pipelines + sequential memory read delay). The first term shows that all *i-1* operations must finish before stage *i* calculations begin. As the pipeline is flushed out, calculations must be stalled. Eqn. (1) shows that only two radix-2/4 butterflies are needed to complete a 2048-pt FFT in <2048 cycles. Memory-based architectures require more complex control logic, but adapt more easily to reconfigurable, non-2^n flavors.

2.2. Cooley-Tukey & Prime-Factor Algorithms

The widely-used Cooley-Tukey algorithm (CTA) computes 2^n-pt FFTs with a complexity of O(NlogN) [2],[14]:

$$X[k_1,k_2] = \sum_{n2} W_N^{n_2k_1}\{\sum_{n1} x[n_1,n_2]W_{N_1}^{n_1k_1}\}W_{N_2}^{n_2k_2} \quad (2)$$

As shown in [15], The DIF FFT is constructed with ordered inputs, digit-reversed outputs, and twiddle multiplication ($W_N^{n_2k_1}$) at the butterfly output. The opposite is true for a DIT FFT.

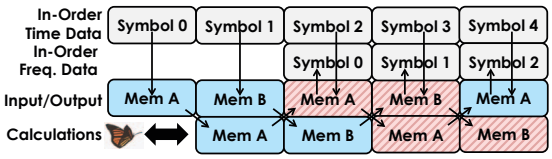

Fig. 1. Memory access timing with 2N memory (DIF↔DIT every 2*a*th symbol)

The CTA can be used for coprime DFT decompositions, but the prime-factor algorithm (PFA) achieves better efficiency by eliminating twiddle multiplications [5],[19]:

$$X[k_1,k_2] = \sum_{n2}\{\sum_{n1} x[n_1,n_2]W_{N_1}^{n_1k_1}\}W_{N_2}^{n_2k_2} \quad (3)$$

In [2], Hsiao *et al.* propose the following input/output index-mapping scheme, which we adopt in our design:

$$n = (N_2n_1 + A_1n_2)\text{mod}N,\ A_1 = p_1N_1 = q_1N_2 + 1 \quad (4)$$

$$k = (B_1k_1 + N_1k_2)\text{mod}N,\ B_1 = p_2N_2 = q_2N_1 + 1 \quad (5)$$

When N_1 and N_2 are coprime, (4-5) simplify to one of several possible PFA mappings. Otherwise, using $A_1, B_1 = 1$, the equations reduce to the CTA mapping. Ordering the factorization of N so that the PFA decomposition occurs before the CTA minimizes twiddle multiplications [5]. This allows the twiddle ROMs to be partitioned by coprimes to ease reconfigurability, and reduces the necessary ROM size. Without further optimizations, the # of twiddle factors stored for radix-2/3/4/5 support is:

$$T = 3{\times}2^{n,max-2} + 2{\times}3^{m,max-1} + 4{\times}5^{k,max-1} \quad (6)$$

Only 1718 twiddle factors are needed for all WiFi/LTE FFTs, so the twiddle ROM can be 16% smaller than that of a 2048-pt Spiral design [3].

2.3. DIF ↔ DIT Duality for Memory Reduction

Typically, calculations can be done in-place, but I/O cannot because input/output mappings are digit-reversed. A naïve ping-pong memory-based FFT architecture thus requires at least 3N memory entries to support continuous data flow. The duality of the DIF/DIT decompositions (and extensions to the PFA) enables 2N memory usage [6]. As shown in Fig. 1, reversing the decomposition order every 2*a*th symbol allows $x_{2a+2}[n]$,$x_{2a+3}[n]$ to be written to memory as $X_{2a}[k]$,$X_{2a+1}[k]$ is read out, in order, with n=k. The generated design uses 50% less BRAM (in an FPGA) than a Spiral equivalent.

To extend this duality to the PFA [2], first assume N is decomposed in the order N_1,N_2,N_3. Then, from (4-5),

$$n = (N_2N_3n_1 + A_1\tilde{n}_2)\text{mod}N,\ \tilde{n}_2 = (N_3n_2 + A_2n_3)\text{mod}N_2N_3 \quad (7)$$

$$k = (B_1k_1 + N_1\tilde{k}_2)\text{mod}N,\ \tilde{k}_2 = (B_2k_2 + N_2k_3)\text{mod}N_2N_3 \quad (8)$$

When N_1,N_2,N_3 are coprime (GCD=1),

$$A_1 = p_1N_1 = Q_1N_2N_3 + 1,\ Q_1 = q_1 \quad (9)$$

$$A_2 = p_2N_2 = Q_2N_3 + 1,\ Q_2 = q_2N_1 \quad (10)$$

$$B_2 = p_3N_3 = Q_3N_2 + 1,\ Q_3 = q_3N_1 \quad (11)$$

Q_x is calculated via the Scala implementation of the extended Euclidean algorithm $\gcd(a,b) = ax + by$.

Substitution of (9-11) into (7-8) results in the mappings [2]:

$$n = (N_2N_3n_1 + p_1N_1N_3n_2 + p_1N_1p_2N_2n_3)\text{mod}N \quad (12)$$

$$k = (p_2p_3N_2N_3k_1 + N_1p_3N_3k_2 + N_1N_2k_3)\text{mod}N \quad (13)$$

A similar decomposition in the reversed order N_3,N_2,N_1 results in:

$$n = (N_1N_2n_1 + p_3N_1N_3n_2 + p_2p_3N_2N_3n_3)\text{mod}N \quad (14)$$

$$k = (p_1p_2N_1N_2k_1 + p_1N_1N_3k_2 + N_2N_3k_3)\text{mod}N \quad (15)$$

As in the CTA case, to achieve in-place I/O with 2N memory,

$$(k_3,k_2,k_1)_{2a} \rightarrow (n_1,n_2,n_3)_{2a+2} \quad (16)$$

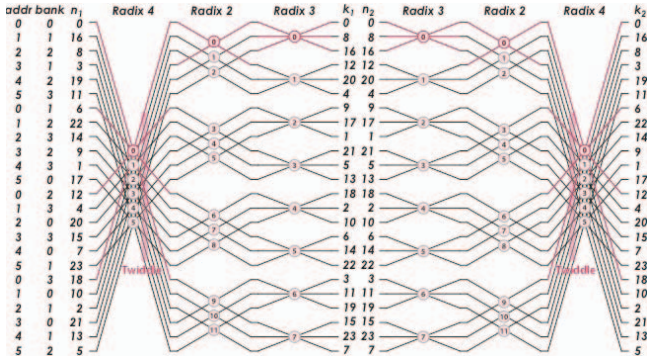

Fig. 2. N=24 SFGs: 4, 2, 3 *(left)* and 3, 2, 4 *(right)* decompositions

The signal-flow graph (SFG) in Fig. 2 illustrates the duality of the forward/reverse combined PFA+CTA decompositions used.

3. FFT HARDWARE GENERATOR

3.1. Translating Constraints into Reusable Hardware

The Chisel hardware construction language was chosen to build the generator over traditional HDLs due to its improved parameterization capabilities, modern programming language features, and more compact code [7]. The generator has two components. Given a set of user constraints (i.e. desired FFT sizes), the "firmware part" uses Scala to calculate parameters and create lists of constants (twiddles, coprimes, etc.) for look-up table (LUT) generation. The LUT construction can be performed in 2 lines of Scala code, without requiring the user to switch to some intermediate representation. The "hardware component" is an FFT template containing blocks that control data flow between I/O, memories, and the butterfly unit, as in Fig. 3. The FFT template uses the calculated parameters to specify memory sizes and distribution, calculation + I/O rates, amount of butterfly pipelining, signal bit widths, and a butterfly to support all needed radices. Parameters can be updated without rewriting code, and synthesizable Verilog is generated directly by Chisel. Chisel also simplifies DSP validation via two simulation modes: 1) floating-point for accurate verification and 2) fixed-point to evaluate realistic hardware metrics.

3.2. I/O Control Logic

The index vector generator from [2] has been extended for this design. For decomposition of coprime N_1,N_2,N_3, the counters (n_1', n_2', n_3') and stored $Q_x' = (N_x - Q_x)\text{mod}N_x$ values are used to generate indices:

$$n_1 = (n_1' + Q_1'\tilde{n}_2)\text{mod}N_1,\quad n_2 = (n_2' + Q_2'n_3)\text{mod}N_2 \quad (17)$$

The counter sequence is implemented in the following pseudocode:

$$n_3 = n_3' := \{n_3' + 1\ if\ n_3' \neq N_3 - 1; 0\ otherwise\}$$

$$n_{x<3}' := \{n_x' + 1\ if\ n_x' \neq N_x - 1; 0\ otherwise\}\ \text{on}\ n_{x+1}' \rightarrow 0$$

The counter $\tilde{n}_2$ increments and wraps when $n_2' = N_2 - 1\ \&\ n_3' = N_3 - 1$ (WC). Thus, the coprime index vector generator performs:

$$R_1 := \{0\ if\ WC; else\ (R_1 + Q_1')\text{mod}N_1\}$$

$$R_2 := \{0\ if\ n_3' = N_3 - 1; else\ (R_2 + Q_2')\text{mod}N_2\}$$

$$n_1 = (n_1' + R_1)\text{mod}N_1,\qquad n_2 = (n_2' + R_2)\text{mod}N_2$$

Because $R_x, Q_x', n_x' \in [0, N_x)$, the sums are $< 2N_x - 1$, so a simple subtractor and two-input MUX can compute the modulus, as in [5]. One coprime index vector generator can be used for both DIF/ DIT with N_1,N_2,N_3 reversal, but Q_x's must be found separately.

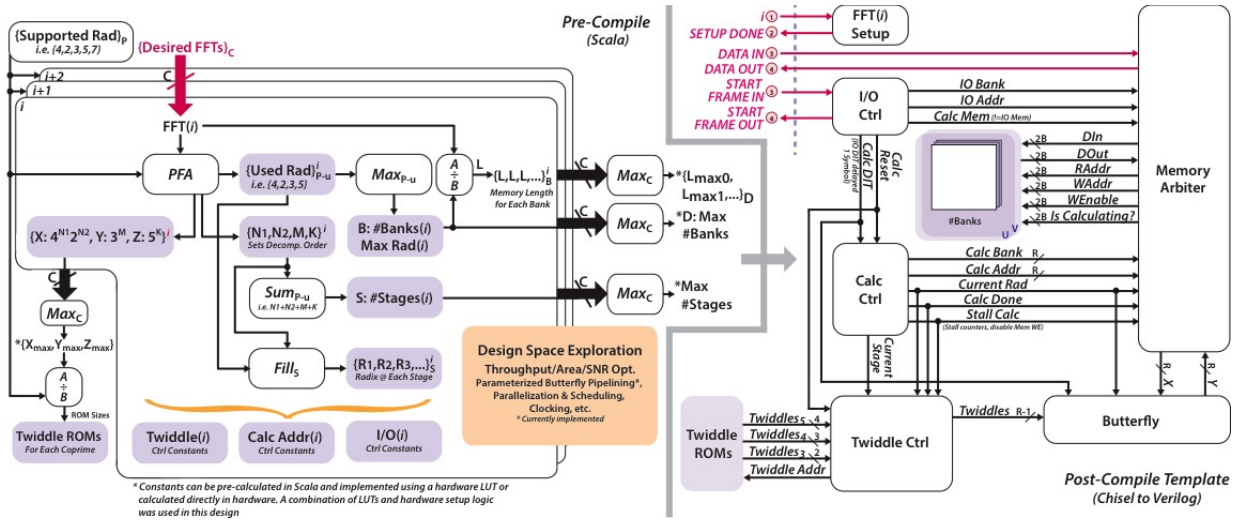

Fig. 3. Generator block diagram consisting of 1) a "firmware part" for parameterization *(left)* and 2) a hardware FFT template *(right)*

To support coprime factorization and then CTA, the vector $(n_{1,x}, n_{1,x-1}, \dots, n_{1,0}, n_{2,x}, \dots, n_{2,0}, n_{3,x}, \dots, n_{3,0})$ can be obtained by using the (n_1, n_2, n_3) values to address corresponding decimal-to-base-r LUTs. However, we have chosen instead to store the Q'_xs in base-r notation and implement base-r adders. The coprime modulus operations are obtained "for free" by simply masking out the appropriate base-r digits, and the final index vector is directly obtained without additional LUTs. As an example, if N_3 is 5^k, n_3 is decomposed into the base-5 digits $(n_{3,x}, n_{3,x-1}, \dots, n_{3,0})$. For an $N_1', N_2', N_3' = N_3, N_2, N_1$ decomposition, n_1 uses base-5 and the index vector is reversed for memory address/bank generation: $(n_{3,0}, \dots, n_{3,x}, n_{2,0}, \dots, n_{2,x}, n_{1,0}, \dots, n_{1,x})$. The individual coprimes are vectorized in digit-reversed order.

3.3. Calculation + Memory Access Control

The DIF FFT uses S radix stages. Fig. 1 shows that the number of radix-r_i butterflies at stage i is N/r_i. For $i > 0$, butterflies occur in $\prod_{x=0}^{i-1} r_i$ groups of $\prod_{x=i+1}^{S-1} r_i$ ordered operations. For the DIT-equivalent FFT, the DIF stages are traversed in reverse. A set of n_x counters is used to index operands and track the butterfly iterations per stage:

$n_{S-1} \coloneqq \{n_{S-1} + 1 \; if \; n_{S-1} \neq r_{S-1} - 1 \; \& \; i \neq S-1; \; else \; 0\}$
$n_{x<S-1} \coloneqq \{n_x + 1 \; if \; n_x \neq r_x - 1, \; i \neq x; \; else \; 0\}$ on $n_{x+1} \to 0$

Because $r_i \leq r_{max}$, each butterfly operand indexed by $j < r_i$ is guaranteed to come from a different bank b_j [2], given by:

$$b_0 = (\sum_{x=0}^{S-1} n_x) \mathrm{mod} r_{max} \quad (18)$$

$$b_{0<j<r_i} = (b_0 + j) \mathrm{mod} r_{max} \quad (19)$$

Mod operations are more complex than the XOR logic in [8], but they simply extend XOR (add mod 2) to more general base-r's, which is useful in mixed-radix designs. Because the N data are split amongst r_{max} banks, data in each bank are mapped to $[0, N/r_{max})$. A possible addressing scheme if $r_0 = r_{max}$ (where a_j is the address of the jth butterfly operand) is:

$$A_{S-1} = 1, \; A_{x<S-1} = A_{x+1} \times r_{x+1} \quad (20)$$

$$a_0 = \sum_{x=0}^{S-1} A_x n_x \quad (21)$$

$$a_{j>0} = a_{j-1} + A_i \quad (22)$$

With pipelined butterflies, calculations are temporarily stalled and memory writes are disabled, so stages can compute on fresh data.

3.4. Twiddle Address Generation

For an N-pt DIF FFT, the non-trivial i^{th} stage twiddles $W_N^{jn_{tw}}, j \in [1, r_i)$ repeated in each butterfly group are addressed by:

$$n_{tw,i=0} = [0, N/r_0) \quad (23)$$

$$n_{tw,i>0} = (\prod_{x=0}^{i-1} r_x) \times [0, N/(\prod_{x=0}^{i} r_x) - 1] \quad (24)$$

As shown in [15], the first CTA DIF stage requires the most twiddle factors. Because the DFT is decomposed into smaller sizes, each subsequent stage uses fewer unique twiddles, and the last stage requires none. The twiddle index is renormalized back to the full ROM range (i.e. $W_4^1 = W_{16}^4$). Since the range of twiddle addresses for a given radix-r is set by its largest associated coprime ÷ r, all values of $2^n < 2^{n,\max}$, $3^m < 3^{m,\max}$, $5^k < 5^{k,\max}$ are supported by only 9 ROMs with address renormalization (i.e. the renormalization factor for an initial radix-4 stage is $2^{n,max}/2^n$).

To extend this addressing scheme to combined PFA/CTA decompositions, the twiddle address of a radix-r_i stage is held for z_i butterflies. If r_i's corresponding coprime is $C_t, t \in [0, D)$ where D represents the number of coprimes in the decomposition, then

$$z_i = \prod_{x=t+1}^{D-1} C_x \quad (25)$$

Note that when $t = D - 1$, $z_i = 1$.

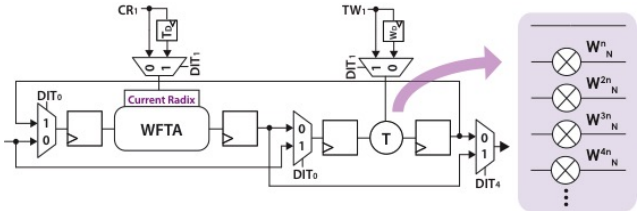

Fig. 4. Reconfigurable processing element for DIF/DIT

3.5. WFTA Butterfly Supporting Hardware Reuse

The WFTA butterfly uses fewer multipliers at the expense of extra adders [18]. To minimize area, the adders/multipliers of a radix-7 butterfly can be reused for radix-2/3/4/5, as in [9]. Fig. 5 shows a modified version of [9], used here. It supports optional logic pipelining per computation stage as required by the FPGA/ASIC and bypasses adders/multipliers when radices are unused. To support full reconfigurability, constant multipliers are not used, but

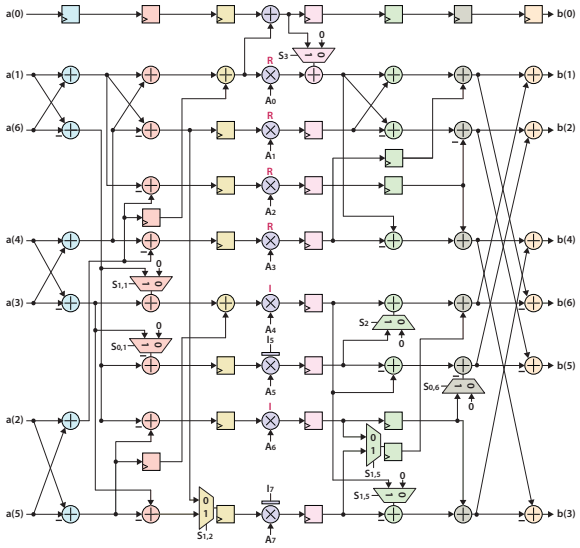

Fig. 5. Radix-2/3/4/5/7 WFTA butterfly with reuse [9]

only products of complex inputs and a few real/imaginary values are needed. As Fig. 4 shows, the processing element (butterfly + twiddle multipliers) supports DIF/DIT reconfiguration using one set of twiddle multipliers with at least one stage of pipelining between memory accesses.

4. DISCUSSION

4.1. Scheduling Limitations

For continuous data flow with LTE/WiFi FFTs, butterflies are computed at 2x the I/O clock rate. By using a single butterfly at this clock rate, the N=648 and N=864 calculations (requiring >1350 and >1728 computation cycles respectively) cannot be completed in time. Since the radix-2 stage (no twiddles) requires the most clock cycles (CCs), halving the iterations during that stage remedies this problem. This is achieved by operating two radix-2 butterflies in parallel, reusing radix-4 hardware, and shuffling butterflies to prevent bank conflicts. For the radix-2 stage of an N=24 FFT, Fig. 2 shows that the b^{th} butterflies in groups 0/2 and 1/3 use the same addresses and non-conflicting memory banks. Additional logic derived from this observation is used to guarantee FFT computation in N I/O CCs for LTE/WiFi.

Only one butterfly may be used with the implemented banking scheme. In order to support area/throughput flexibility and an arbitrary number of butterflies in parallel, it is necessary to extend butterfly scheduling further. This is especially difficult when one desires a general solution for *runtime-reconfigurable*, mixed-radix FFTs without adding considerable complexity to the I/O (as would be incurred in [10]). However, it is not necessary to change operand locations in order to prevent bank conflicts. Instead, butterfly operations within a radix-r stage may be reordered to minimize conflicts.

4.2. Results & Comparisons with State-of-the-Art

A design has been verified in a cycle-accurate fashion via Chisel's built-in tester. As shown in Fig. 6, two generated FFT engines (24,24-bit I/O) were synthesized with 28nm standard cells and a clock target of 3.9ns to meet LTE/WiFi requirements. There is a

Table 1. Resource comparison for fixed N = 2048

	This	***[3]$_1$***	***[3]$_2$***	***[3]$_3$***	***[13]***
D. Mem	2N	7.3N	7.99N	4N	N-1*
T. ROM	0.75N	0.99N	0.99N	N	-
CCs	~3600	512	1024	11,287	2048
# BFs	1 Rad-4/2×2	5 Rad-4 + Rad-2	11	1	11
# Muls**	12	56	40	4	40

1: Rad-4 streaming; 2: Rad-2 streaming; 3: Rad-2 iterative
* No memory allocated for I/O unscrambling
**Real muls in data path; complex multiplier = 4 real muls

Table 2. Resource comparison for reconfigurable LTE/WiFi FFTs

	This	***[5]****	***[2]****	***Xilinx****
D. Mem	2N	2N	2N	2N
T. ROM	1718	-	-	-
Calc. Clk/IO Clk**	2x	1x	2x	4x
# Muls	26	44	26	16

* Comparison #'s taken from [5]
** Ratio of calculation to IO clock rates for continuous data flow

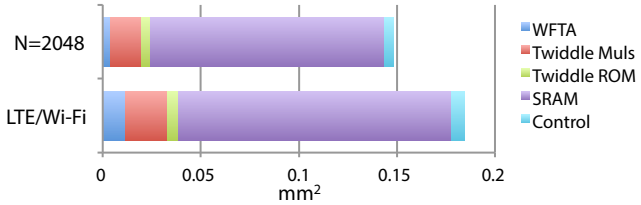

Fig. 6. FFT area breakdown

25% area penalty (compared to a fixed N=2048 engine) to support mixed-radix reconfigurability. The radix-4/2 WFTA butterfly is smaller than the reconfigurable 2/3/4/5 butterfly, and memories occupy a vast majority of the area. Support for all LTE/WiFi FFTs requires 2×(240-length memory + 4 banks of 512-length memory).

The proposed generator can create FFT engines with performance comparable to state-of-the-art reconfigurable FFTs [2],[5]. Wang *et al.* [11] propose a pipelined architecture that can be reconfigured/scaled, but it isn't implemented as a generator. While achieving higher throughput, the generated FFT engine uses 50% less data memory and 25% smaller twiddle storage than the radix-2 N=2048 iterative FFT from Spiral's online generator [3]. Memory savings are even greater relative to Spiral's streaming FFTs, but with a clear area/throughput tradeoff (Table 1). Designs in Table 2 with fewer multipliers require higher calculation clock rates. Genesis [12] and the fixed pipelined architecture from [13] do not address I/O and its impact on memory.

4.3. Conclusion

This work reports a generator of memory-based, runtime-reconfigurable $2^n3^m5^k$ FFT engines, developed entirely in Chisel. The FFT design instances are competitive with state-of-the-art reconfigurable architectures, satisfying the WiFi and LTE operating modes while supporting continuous data flow with 2N memories. This work can be extended to support 7^l coprimes and more generalized FFT scheduling for exploring area/throughput tradeoffs of reconfigurable architectures.

The authors acknowledge DARPA CRAFT (HR0011-15-1-0010), NSF-GRFP (DGE-1106400), ASPIRE, and BWRC faculty, staff, and students (esp. S. Bailey, P. Rigge, S. Twigg) for their support.

5. REFERENCES

[1] 3GPP TS 36.211, "Physical Channels and Modulation," V 12.0.0, December 2013.

[2] C.-F. Hsiao, Y. Chen and C.-Y. Lee "A Generalized Mixed-Radix Algorithm for Memory-Based FFT Processors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 57, no. 1, pp. 26-30, 2010.

[3] M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. Singer, J. Xiong, F. Franchetti, A. Gacic, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo, "SPIRAL: Code Generation for DSP transforms," *Proc. of IEEE, special issue on "Program Generation, Optimization, and Adaptation"*, vol. 93, no. 2, pp. 232–275, 2005.

[4] R. Koutsoyannis, P. Milder, C. Berger, M. Glick, J. Hoe, and M. Püschel, "Improving Fixed-Point Accuracy of FFT in O-OFDM Systems," *Proc. of the Intl. Conf. on Acoustics, Speech, and Signal Processing*, pp. 1585-1588, March 2012.

[5] J. Chen, J. Hu, S. Lee, and G. Sobelman, "Hardware Efficient Mixed Radix-25/16/9 FFT for LTE Systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 23, no. 2, pp. 221–229, Feb 2015.

[6] C. S. Burrus and P.W. Eschenbacher, "An In-Place, In-Order Prime Factor FFT Algorithm," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 29, no. 4, pp. 806-817, Aug. 1981.

[7] J. Bachrach, H. Vo, B. Richards, K. Asanovic, and J. Wawrzynek, "Chisel: Constructing Hardware in a Scala Embedded Language," in *Proceedings of the 49th Design Automation Conference (DAC)*, 2012.

[8] H. Sorokin, J. Takala, "Conflict-Free Parallel Access Scheme for Mixed-Radix FFT Supporting I/O Permutations," *Proc. of the Intl. Conf. on Acoustics, Speech, and Signal Processing (ICASSP2011*), Prague, C.Z., pp. 2709-1712, May 2011.

[9] F. Qureshi, M. Garrido, and O. Gustafsson, "Unified Architecture for 2, 3, 4, 5, and 7-point DFTs Based on Winograd Fourier Transform Algorithm," *Electronics Letters*, vol. 49, no. 5, pp. 348–349, 2013.

[10] S. Richardson, D. Marković, A. Danowitz, J. Brunhaver, and M. Horowitz, "Building Conflict-Free FFT Schedules," *IEEE Transactions on Circuits and Systems,* vol. 62, no. 4, pp. 1146-1155, April 2015.

[11] G. Wang, B. Yin, I. Cho, J. Cavallaro, S. Bhattacharyya, and J. Takala, "Efficient Architecture Mapping of FFT/IFFT for Cognitive Radio Networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 3933–3937, May 2014.

[12] "Creating Chip Generators Using Genesis2," Stanford, http://genesis2.stanford.edu/.

[13] J. Löfgren and P. Nilsson, "On Hardware Implementation of Radix 3 and Radix 5 FFT Kernels for LTE Systems," *Proc. NORCHIP Conf.*, pp. 1–4, Nov. 2011.

[14] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965.

[15] C. Sidney Burrus, "Appendix 1: FFT Flowgraphs," OpenStax CNX. September 18, 2009 http://cnx.org/contents/e460644d-c1d6-4dee-a60e-3ee5220e88ba@11.

[16] T. D. Chiueh and P. Y. Tsai, *OFDM Baseband Receiver Design for Wireless Communications*. New York: Wiley, 2007.

[17] C.-H. Yang, T.-H. Yu, and D. Marković, "Power and Area Minimization of Reconfigurable FFT Processors: A 3GPP-LTE Example," *IEEE J. Solid-State Circuits* , vol. 47, no. 3, pp. 757–768, Mar. 2012.

[18] S. Winograd, "On Computing the Discrete Fourier Transform," *Math. Comp*., vol. 32, pp. 175–199, 1978.

[19] I. J. Good, "The Interaction Algorithm and Practical Fourier Analysis: An Addendum," *J. R. Statist. Soc.,* no. 2, pp. 372–375, 1960.

[20] P. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Computer Generation of Hardware for Linear Digital Signal Processing Transforms," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 17, no. 2, p. 15, 2012.

[21] Z.-X. Yang, Y.-P. Hu, C.-Y. Pan, and L. Yang, "Design of a 3780-point IFFT Processor for TDS-OFDM," *IEEE Trans. Broadcast.*, vol. 48, no. 1, pp. 57–61, Mar. 2002.