27.4 A 0.75-Million-Point Fourier-Transform Chip for Frequency-Sparse Signals

Omid Abari, Ezz Hamed, Haitham Hassanieh, Abhinav Agarwal, Dina Katabi, Anantha P Chandrakasan, Vladimir Stojanovic

Massachusetts Institute of Technology, Cambridge, MA

Applications like spectrum sensing, radar signal processing, and pattern matching by convolving a signal with a long code, as in GPS, require large FFT sizes. ASIC implementations of such FFTs are challenging due to their large silicon area and high power consumption. However, the signals in these applications are sparse, i.e., the energy at the output of the FFT/IFFT is concentrated at a limited number of frequencies and with zero/negligible energy at most frequencies. Recent advances in signal processing have shown that, for such sparse signals, a new algorithm called the sparse FFT (sFFT) can compute the Fourier transform more efficiently than traditional FFTs [1].

This paper presents a VLSI implementation of the sFFT algorithm. The chip implements a 746,496-point sFFT, in $0.6mm^2$ of silicon area. At 0.66V, it consumes 0.4pJ/sample and has an effective throughput of 36GS/s. The effective throughput is computed over all frequencies but frequencies with negligible magnitudes are not produced. The chip works for signals that occupy up to 0.1% of the transform frequency range (0.1% sparse). It can be used to detect a signal that is frequency hopping in a wideband, to perform pattern matching against a long code, or to detect a blocker location with very high frequency resolution. For example, it can detect and recover a signal that occupies 18MHz randomly scattered anywhere in an 18GHz band with a frequency resolution of ~24kHz.

The sFFT algorithm has three steps: bucketization, estimation, and collision resolution. _Bucketization_: The algorithm starts by mapping the spectrum into buckets as shown in Fig. 27.4.1. This is done by sub-sampling the signal and then performing an FFT. Sub-sampling in time causes aliasing in frequency. Since the spectrum is sparsely occupied, most buckets will be either empty or have a single active frequency, and only few buckets will have a collision of multiple active frequencies. Empty buckets are discarded and non-empty buckets are passed to the estimation step.

Estimation: This step estimates the value and frequency number (i.e. location in the spectrum) of each active frequency. In the absence of a collision, the value of an active frequency is the value of its bucket. To find the frequency number, the algorithm repeats the bucketization on the original signal after shifting it by 1 sample. A shift in time causes a phase change in the frequency domain of $2\pi f\tau/N$, where f is the frequency number, τ is the time shift, and N is the sFFT size. Thus, the phase change can be used to compute the frequency number.

Collision resolution: The algorithm detects collisions as follows: If a bucket contains a collision then repeating the bucketization with a time shift causes the bucket's magnitude to change since the colliding frequencies rotate by different phases. In contrast, the magnitude does not change if the bucket has a single active frequency. After detecting collisions, the algorithm resolves them by using bucketization multiple times with co-prime sampling rates (FFTs with co-prime sizes). Two numbers are co-prime if their greatest common divisor is one. The use of co-prime sampling rates guarantees that any two frequencies that collide in one bucketization do not collide in other bucketizations, as shown in Fig. 27.4.1.

The block diagram of the sFFT chip is shown in Fig. 27.4.2. A 12b 746,496-point ($2^{10}\times3^6$-point) sFFT is implemented. Two types of FFTs (2^{10} and 3^6-point) are used for bucketization. The input to the 2^{10}-point FFT is the signal sub-sampled by 3^6, while the input to the 3^6-point FFT is the signal sub-sampled by 2^{10}. FFTs of sizes 2^{10} and 3^6 were chosen since they are co-prime and can be implemented with simple low-radix FFTs. Three FFTs of each size are used with inputs shifted by 0, 1 or 32 time samples, as shown in Fig. 27.4.2. In principle, shifts of 0 and 1 are sufficient. However, the third shift is used to increase the estimation accuracy. One 1024-word and one 729-word SRAMs are used for three 2^{10}-point and three 3^6-point FFTs, respectively. SRAMs are triplicated to enable pipelined operation of the I/O interface, bucketization and reconstruction blocks. Thus, 3 sFFT frames exist in the pipeline.

The micro-architecture of the 2^{10}-point FFT is shown in Fig. 27.4.3. Each 2^{10}-point FFT uses one radix-4 butterfly to perform an in-place FFT, which is optimized to reduce area and power consumption as follows: First, the FFT block performs read and write operations at even and odd clock cycles, respectively, which enables the use of single port SRAMs. A single read operation provides three complex values, one for each radix-4 butterfly. The complex multiplication is computed over two clock cycles using two multipliers for each butterfly. Second, a twiddle factor (TWF) control unit is shared between the three butterflies. Third, the block floating point (BFP) technique is used to minimize the quantization error [2]. BFP is implemented using a single exponent shared between FFTs, and scaling is done by shifting in case of overflow. Round-half-away-from-zero is implemented by initializing the accumulator registers with 0.5LSB and truncating the results. The 3^6-point FFTs are similar, but use radix-3 butterflies.

The micro-architecture of estimation and collision detection is shown in Fig. 27.4.4. Phase shift and phase detector units use the CORDIC algorithm. The estimation block operates in two steps. First, time shifts of 1 and 32 samples are used to compute the MSBs and LSBs of the phase change, respectively. A 3b overlap is used to fix errors due to concatenation. Since the 5 MSBs of phase change are taken directly from the output of phase detectors, active frequencies have to be ~30dB above the quantization noise to be detected correctly. Frequencies below this level are considered negligible. The frequency number is estimated from the phase change. This frequency number may have errors in the LSBs due to quantization noise. The second step corrects any such errors by using the bucket number to recover the LSBs of the frequency number. This is possible because all frequencies in a bucket share the same remainder B ($B=f$ mod M, where f is the frequency number and M is the FFT size), which is also the bucket number. Thus, in the frequency recovery block associated with the 2^{10}-point FFTs, the bucket number gives the 10 LSBs of the frequency number. However, in the frequency recovery for the 3^6-point FFTs, the LSBs cannot be directly replaced by the bucket number since $M=3^6$ is not a power of 2. Instead, the remainder of dividing the frequency number by 3^6 is calculated and subtracted from the frequency number. The bucket number is then added to the result of the subtraction. In our implementation, calculating and subtracting the remainder is done indirectly by truncating the LSBs of the phase change.

The collision detection block in Fig. 27.4.4 compares the values of the buckets with and without time-shifts. It uses the estimated frequency to remove the phase change in the time-shifted bucketizations and compares the three complex values to detect collisions. In the case of no collision, the three values are averaged to reduce noise. The result is used to update the output of the sFFT in SRAMs.

The testchip is fabricated in IBM's 45nm SOI technology. The sFFT core occupies $0.6mm^2$ including SRAMs. At 1.18V supply, the chip operates at a maximum frequency of 1.5GHz, resulting in an effective throughput of 109GS/s. At this frequency, the measured energy efficiency is 1.2μJ per 746,496–point Fourier transform. Reducing the clock frequency to 500MHz enables an energy efficiency of 298nJ per Fourier transform at 0.66V supply. Energy and operating frequency for a range of supply voltages are shown in Fig. 27.4.5.

Since no prior ASIC implementations of sFFT exist, we compare with recent low-power implementations of the traditional FFT [3-5]. The measured energy is normalized by the Fourier transform size to obtain the energy per sample (the sFFT chip, however, outputs only active frequencies). Fig. 27.4.6 shows that the implementations in [3-5] work for sparse and non-sparse signals while the sFFT chip works for signal sparsity up to 0.1%. However, for such sparse signals, the sFFT chip delivers ~40× lower energy per sample for a 3^6× larger FFT size. Fig. 27.4.6 also shows that the 746,496-point sFFT chip achieves an 88× reduction in run-time compared to a C++ implementation running on an i7 CPU [6].

References:
[1] H. Hassanieh, _et al._, "Simple and Practical Algorithm for Sparse Fourier Transform," _ACM Symp. Discrete Algorithms_, 2012.
[2] G. Zhong, _et al._, "A Power-Scalable Reconfigurable FFT/IFFT IC Based on a Multi-Processor Ring," _IEEE J. Solid-State Circuits_, Feb. 2006.

[3] M. Seok, *et al.*, "A 0.27V 30MHz 17.7nJ/transform 1024-pt Complex FFT Core with Super-Pipelining," *ISSCC Dig. Tech Papers*, 2011.

[4] Y. Chen et al., "A 2.4-Gsample/s DVFS FFT Processor for MIMO OFDM Communication Systems," *IEEE J. Solid-State Circuits*, May 2008.

[5] C. Yang et al, "Power and Area Minimization of Reconfigurable FFT Processors: A 3GPP-LTE Example," *IEEE J. Solid-State Circuits*, Mar. 2012.
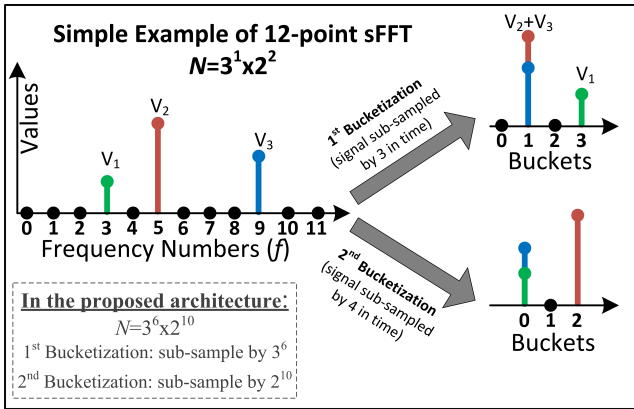
[6] sFFT C++ code. http://groups.csail.mit.edu/netmit/sFFT/code.html

Simple Example of 12-point sFFT
N=3¹x2²

Values

V_1 V_2 V_3

V₂+V₃

V_1

Frequency Numbers (f)
0 1 2 3 4 5 6 7 8 9 10 11

1st Bucketization
(signal sub-sampled by 3 in time)

0 1 2 3
Buckets

2nd Bucketization
(signal sub-sampled by 4 in time)

0 1 2
Buckets

In the proposed architecture:
$N=3^6 \times 2^{10}$
1st Bucketization: sub-sample by 3^6
2nd Bucketization: sub-sample by 2^{10}

Figure 27.4.1: The sFFT algorithm performs bucketization by sub-sampling in the time domain then taking an FFT, which causes aliasing in the frequency domain.
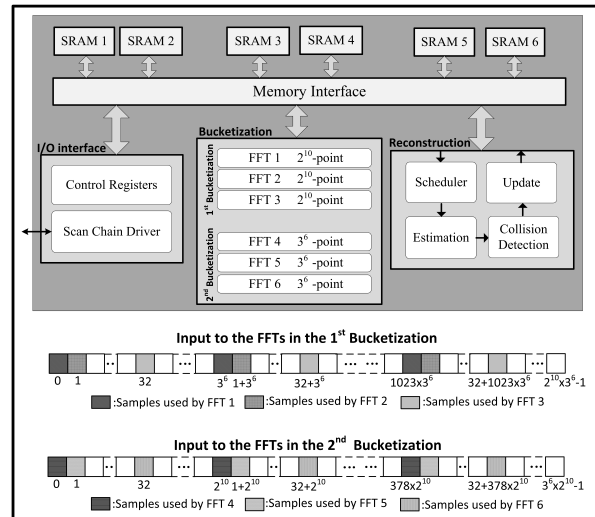
SRAM 1 | SRAM 2 | SRAM 3 | SRAM 4 | SRAM 5 | SRAM 6

Memory Interface

I/O interface
Control Registers
Scan Chain Driver

Bucketization
1st Bucketization: FFT 1 2^{10}-point, FFT 2 2^{10}-point, FFT 3 2^{10}-point
2nd Bucketization: FFT 4 3^6-point, FFT 5 3^6-point, FFT 6 3^6-point

Reconstruction
Scheduler | Update
Estimation | Collision Detection

Input to the FFTs in the 1st Bucketization
0 1 ... 32 ... 3^6 1+3^6 32+3^6 ... 1023x3^6 32+1023x3^6 $2^{10}3^6$-1
:Samples used by FFT 1 :Samples used by FFT 2 :Samples used by FFT 3

Input to the FFTs in the 2nd Bucketization
0 1 ... 32 ... 2^{10} 1+2^{10} 32+2^{10} ... 378x2^{10} 32+378x2^{10} $3^6 2^{10}$-1
:Samples used by FFT 4 :Samples used by FFT 5 :Samples used by FFT 6

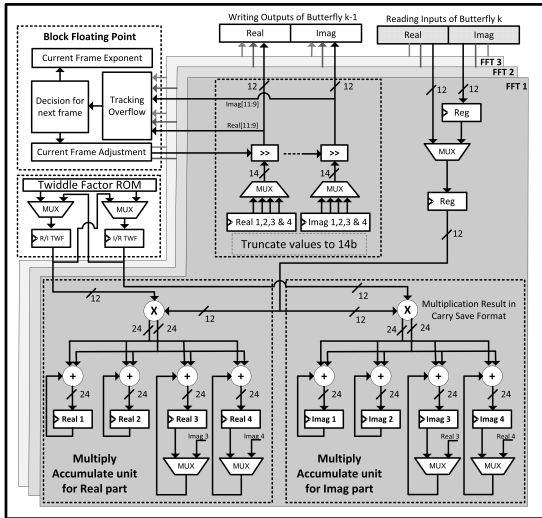**Figure 27.4.2:** A block diagram of the 2^10x3^6-point sparse FFT and input samples to the 6 FFTs.

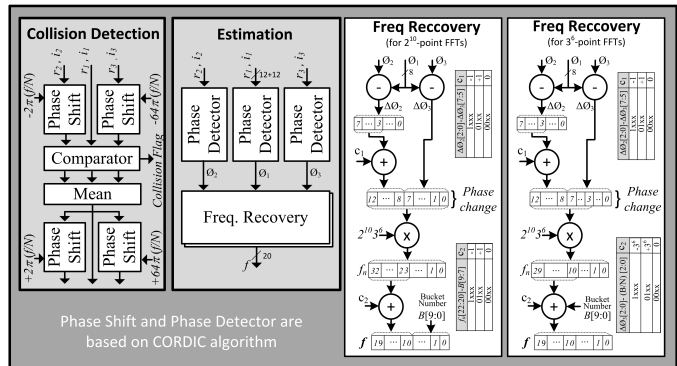**Figure 27.4.3:** The micro-architecture of the 2^{10}-point FFTs.

Figure 27.4.4: The micro-architecture of collision detection and estimation. The complex values (r1, i1), (r2, i2) and (r3, i3) are the output of bucketization for time-shifts 0, 1 and 32 samples.
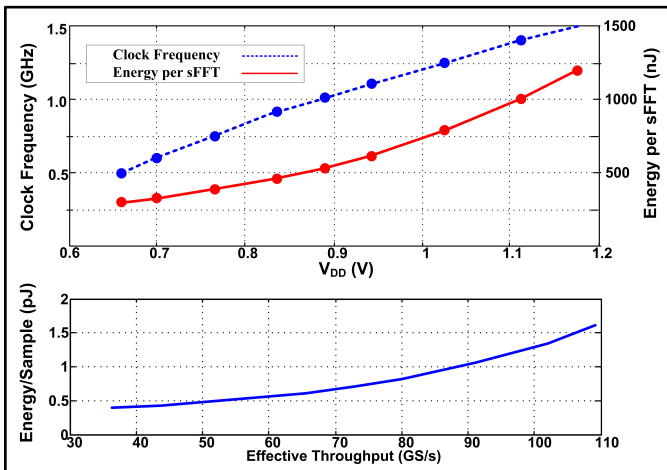
Figure 27.4.5: Measured energy and operating frequency for a range of voltage, and throughput versus energy per sample for computing a 746,496-point sparse Fourier transform.

	[3]	[4]	[5]	**This work**
Technology	65 nm	90 nm	65nm	45 nm
Signal Type	Any Signal	Any Signal	Any Signal	Freq.-Sparse signal
Size	2^{10}-point	2^8-point	2^7 to 2^{11}-point	$3^6 \times 2^{10}$-point
Word width	16 bits	10 bits	12 bits	12 bits
Area	8.29mm²	5.1mm²	1.37mm²	0.6mm²
Effective Throughput	240MS/s	2.4GS/s	1.25-20MS/s	36.4-109.2GS/s
Energy/Sample	17.2pJ	50pJ	19.5-50.6pJ	0.4-1.6pJ

Processor	Intel Core i7 Quad Core (3.4GHz)	This work (1.5GHz)
Run-time	600μs	6.8μs

$2^{10}x3^6$ point sparse Fourier transform

Figure 27.4.6: Measured energy efficiency and performance of the sFFT chip compared to published FFTs.

Figure 27.4.7: Die photo of the testchip.
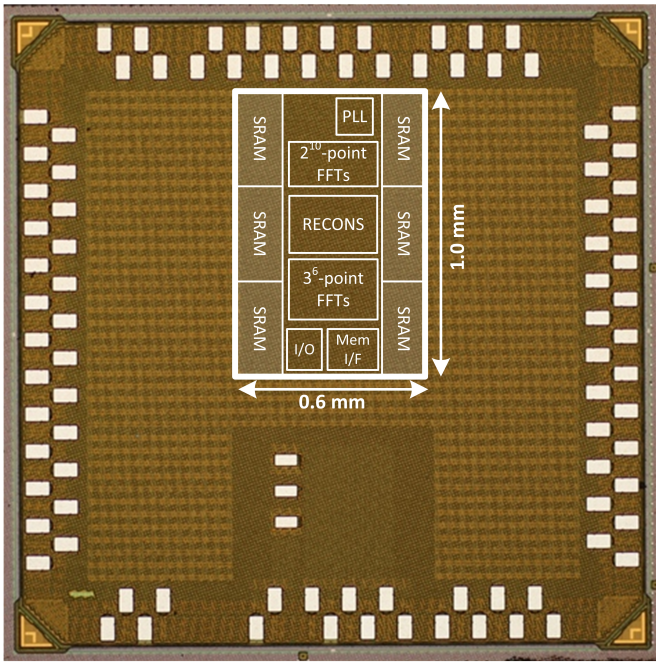
Chip Features

Technology	45nm SOI CMOS
Core Area	0.6mm x 1.0mm
SRAM	3x 75kbits 3x 54kbits
Core Supply Voltage	0.66-1.18 V
Clock Frequency	0.5-1.5 GHz
Core Power	14.6-174.8 mW (PLL in not included)