A High Throughput Hardware Accelerator for FFTW Codelets: A First Look

Larry Tang, Siyuan Chen, Keshav Harisrikanth, Guanglin Xu, Ken Mai, Franz Franchetti
Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA
{lawrenct, siyuanch, kharisri, guanglix, kenmai, franzf}@andrew.cmu.edu

***Abstract*—The Fast Fourier Transform (FFT) is a critical computation for numerous applications in science and engineering. Its implementation has been widely studied and optimized on various computing platforms, with the FFTW library becoming the standard interface in HPC. In this work, we propose hardware acceleration of the FFTW library by putting a software *codelet* into hardware. The hardware is exposed to the user through an FFTW-compatible software library while actual computation takes place behind the scenes on a custom accelerator. To demonstrate a first look at this idea, we design a high throughput accelerator for FFTW twiddle codelets. The FFT hardware is automatically generated using SPIRAL and a test chip is fabricated in a TSMC 28nm process. We provide measured results of the test chip and discuss many opportunities for future work.**

I. Introduction

The Fast Fourier Transform (FFT) is one of the most fundamental algorithms in scientific computing, finding widespread use in domains such as signal/radar processing, digital communications, machine learning, and scientific HPC simulations. For example, in telecommunications a number of wireless standards require a 1D complex FFT/IFFT for OFDM transmission [1]. Some fast convolution algorithms via FFTs compute over inputs of real numbers that are zero-padded, such as 2D convolutions in convolutional neural networks [2]. Scientific applications include simulation of flow [3] and FFT-based partial differential equation solvers for computation of material properties [4].

To satisfy various application requirements, FFT performance has been highly optimized on many different computing platforms. One of the most well known FFT software libraries is FFTW [5] which supports DFTs of arbitrary size and dimension and is further optimized for real/complex data. Although software implementations are suitable for many applications, custom hardware solutions still provide advantages over software. For example, a real-time DSP application may require a dedicated FFT accelerator that can achieve low latency while also being highly energy efficient.

The implementation of FFT hardware accelerators has been well studied in prior literature [6]. Many recent hardware accelerators for variable-length FFTs support a variety of wireless standards [7] [8]. Another recent implementation [9] targets a broader range of scientific applications by supporting FFT sizes from 4 to 4^{10} and single-precision floating point. Industry designs, such as a TI DSP [10], are another option for applications needing FFT acceleration. In general, custom FFT hardware is designed as its own functional block and then integrated into a system or used as a standalone design. Once the design is implemented or generated the functionality becomes fixed to certain sizes and is relatively inflexible.

A RISC-V system now enables one to tightly integrate an accelerator with the processor through custom instructions and interfaces. Thus unlike previous FFT hardware designs, we propose a different design paradigm for FFT accelerators based on the FFTW kernel approach. We propose hardware acceleration of only the computational kernel of FFTW, where the software still invokes and defines how the kernels are tied together. In this work, we observe that the FFTW approach of combining and executing highly-optimized, small size pieces of FFT code known as *codelets* can be hardware accelerated. The hardware accelerator carries out the equivalent computation of a codelet in a non-blocking manner and can be exposed to the user through a FFTW-like C library. Defining the procedure for the FFT computation is left to the software, enabling an enormous amount of flexibility similar to that of FFTW.

Contributions In this paper we discuss the hardening of FFTW codelets and demonstrate a first look with a fabricated test chip and measured results. We make the following contributions:

- The idea that FFTW codelets can be accelerated in hardware with the composition still defined in software to maintain the flexibility of FFTW.
- A codelet accelerator test chip that is fabricated in a TSMC 28nm process.
- An end-to-end system in which the codelet accelerators are exposed to the user as an FFTW-like library.

First we discuss relevant background and in Section III detail the idea of hardening an FFTW codelet. Section IV goes over the design of a high throughput codelet accelerator test chip and Section V discusses the measured results. Finally, we conclude with avenues for future work including integration in a RISC-V based SoC and additional functionality.

II. Background

This section discusses the background related to the Fast Fourier Transform algorithms and recent software frameworks.

Discrete Fourier Transform. The Discrete Fourier Transform (DFT) is the critical mathematical transformation be-

tween time and frequency domain representations of finite-length, discrete data that is used in many applications. The DFT is defined by the following formula [11]:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j2\pi kn/N}, \quad (1)$$

where $(x_n)_{n=0}^{N-1}$ is a sequence of N complex inputs that is transformed to a complex, length-N sequence, $(X_k)_{k=0}^{N-1}$. The complex roots of unity, $e^{-j2\pi kn/N}$, are commonly referred to as *twiddle factors* in the context of the DFT.

Computing a length-N DFT using Equation (1) results in a computational complexity of $\mathcal{O}(N^2)$. In practice, a DFT computation is implemented using the Fast Fourier Transform (FFT) algorithm to reduce the complexity to $\mathcal{O}(N \log N)$. The most well known and commonly used FFT algorithm is the Cooley-Tukey algorithm [12], which recursively decomposes the computation to smaller sized FFTs when N is not prime. FFT algorithms have since been extensively studied, resulting in many variants that optimize for arbitrary input size or different compute platforms and application requirements.

FFTW. FFTW [5] is a general-purpose, adaptive software library for computing the FFT and has become widely used in FFT-based scientific applications. The system combines a domain specific language, code generator, and autotuning infrastructure to produce a portable and high performance FFT library. FFTW provides a number of different routines, with some of its key functionalities including FFTs of multi-dimensions, arbitrary size, and of real or complex data.

The core computational kernels of FFTW are called *codelets*, which are automatically generated, optimized code blocks that enable computation of FFTs for various sizes. In the standard FFTW library they compute power-of-2 sizes up to 64 and some smaller prime sized FFTs. The composition of codelets for a particular FFT problem is determined by the planner and specified through a specialized data structure, and is then executed by calling the codelets as specified by the plan. The FFTW API has become the standard interface for computing FFTs, with a number of current vendor FFT libraries, such as Intel MKL [13] and Nvidia cuFFT [14] implementing a part of the FFTW interface.

FFTX and SPIRAL. The FFTX [15] project looks to build a more modern framework for FFT based applications while still maintaining exposure to the familiar FFTW interface. The first component is a front end library interface that maintains backwards compatibility with FFTW but also provides a means to target higher level operations and new hardware platforms. For example, FFTX targets higher level FFT-based applications such as 1D pruned convolutions which require point-wise operations in addition to FFT calls. At the back end, automatic code generation for the application uses SPIRAL [16]. SPIRAL is a code generation system that targets a wide range of numerical and digital signal processing kernels, originally focusing on FFTs. FFTX will make it easier to target new hardware platforms, including accelerators, for high performance FFT applications and exascale computing efforts.

III. Hardware Accelerator for FFTW Codelets

We now discuss the idea of accelerating FFTW codelets in hardware. We then detail the design flow and FFT hardware generation process for the test chip.

Hardening of FFTW Codelets. The fixed, small size transforms computed by codelets are the fundamental kernel of the FFTW approach to computing FFTs. The transform is completely composed by invoking codelets with their respective parameters including I/O base addresses and strides. There exists two types of codelets: a twiddle codelet that computes an in-place fixed size FFT followed by general twiddle factor multiplications and a no-twiddle codelet that computes only a small, out-of-place FFT. Combining codelets in different ways then enables computing FFTs of real/complex data and of arbitrary size or dimension.

The structure of an example FFTW application is shown in Fig. 1. The function `fftw_plan_dft_1d` constructs and stores the plan as a composition of codelets and the actual execution takes place at the `fftw_execute` function call. We observe that once an FFTW codelet is invoked, there is

```
#include <fftw3.h>

int main() {
    // Declare in/out arrays and plan
    fftw_complex *in, *out;
    fftw_plan p;

    // Allocate/Init
    in = (fftw_complex *)
    ↪ fftw_malloc(sizeof(fftw_complex) * N);
    out = (fftw_complex *)
    ↪ fftw_malloc(sizeof(fftw_complex) * N);
    ...

    // Construct 1D complex FFT plan
    p = fftw_plan_dft_1d(2*N, in, out,
    ↪ FFTW_ESTIMATE);

    // Execute FFT plan
    fftw_execute(p);

    // Post-process and cleanup
    ...
    fftw_destroy_plan(p);
    fftw_free(in), fftw_free(out);
}
```

Fig. 1. FFTW example code

no requirement that the computation takes place in software. The codelet can instead be accelerated in hardware with the actual composition of the codelets still defined in software. Accelerating only the kernel of FFTW enables a large degree of flexibility for the FFT computation.

A hardware accelerator for codelets can be exposed as a FFTW compatible API and can also be targeted by FFTX. While the FFTW API is still maintained to the user, a custom hardware accelerator carries out the actual execution of the codelets. The software infrastructure can be provided through

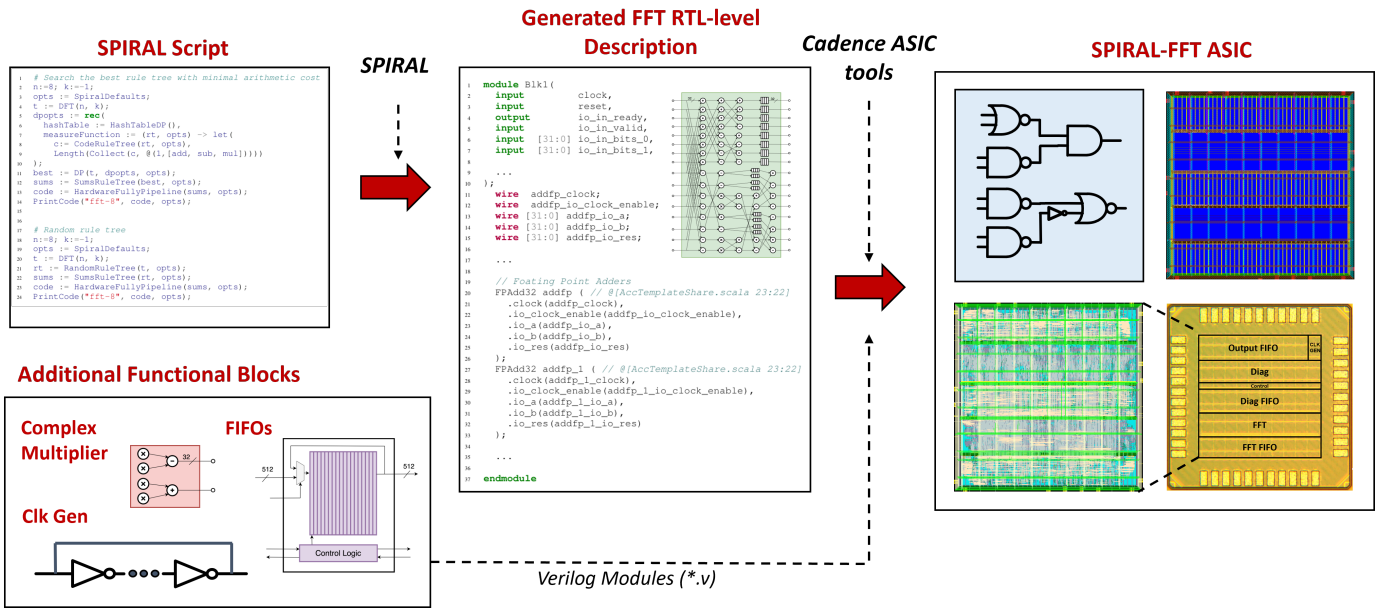


Fig. 2. Overview of the test chip design flow from SPIRAL script to generated RTL code to physical implementation. The SPIRAL script defines the parameters of the generated RTL-level description of the FFT hardware. Other functional blocks are hand designed and integrated with the generated code which then goes through the ASIC design flow.

a C library which looks like FFTW to effectively hide the dispatching of codelets in hardware. Then, the hardware is designed such that it is compatible with the software library and can interpret and execute codelet launches. Fig. 3 shows a software abstraction of how codelets are composed and then invoked in a 4-point FFT example. The first two launches of the hardware kernel computes two DFTs of size 2 with the respective base address and stride parameters. A memory fence is issued due to the data dependency and the final two kernel launches then complete the computation. The software attempts to batch up as many independent codelets as possible, but has the freedom to compose them in any way.

```
// Function executing 4-point DFT
void execute_dft4(_Complex double *X,
  ↪  _Complex double *Y) {

    _Complex double U[4];
    _Complex double T[1] = {__I__};

    // Launch hardware kernels
    dft2_kernel(U, X, NULL, 1, 2, 0);
    dft2_kernel(U+2, X+1, T, 1, 2, 0);

    // Memory Fence
    #pragma pipeline fence

    // Continue invoking kernels
    dft2_kernel(Y, U, NULL, 1, 2, 0);
    dft2_kernel(Y+1, U+1, NULL, 1, 2, 0);
  }

}
```

Fig. 3. Example of a 4-point FFT invoking hardware codelets.

Fig. 4 then shows how that sequence of codelets would be stored in a specialized data structure and then executed by a separate function call.

```
// Data structure containing codelet
  ↪  parameters
descriptor_t dft4[] = {
    {U, X, NULL, 1, 2, 0},
    {U+2, X+1, T, 1, 2, 0},
    FENCE,
    {Y, U, NULL, 1, 2, 0},
    {Y+1, U+1, NULL, 1, 2, 0},
    DONE
};

// Execute plan
execute(dft4);
```

Fig. 4. Example 4-point FFT codelet plan and execution call.

Such an accelerator can be designed to concurrently process many codelet invocations and also built into a real system targeting FFT applications. Furthermore, the FFT hardware can be automatically generated through the use of SPIRAL. We demonstrate a first look of this idea through a hardware accelerator test chip that executes only the twiddle codelet computation. The accelerator processes independent fixed size FFT computations and twiddle factor multiplications in a pipelined fashion. In the following section we discuss in more detail the design flow for developing this hardware accelerator.

Hardware Generation. To facilitate design of the FFT hardware, we use SPIRAL to automatically generate the RTL code for a radix-8 complex FFT. The design flow proceeds as

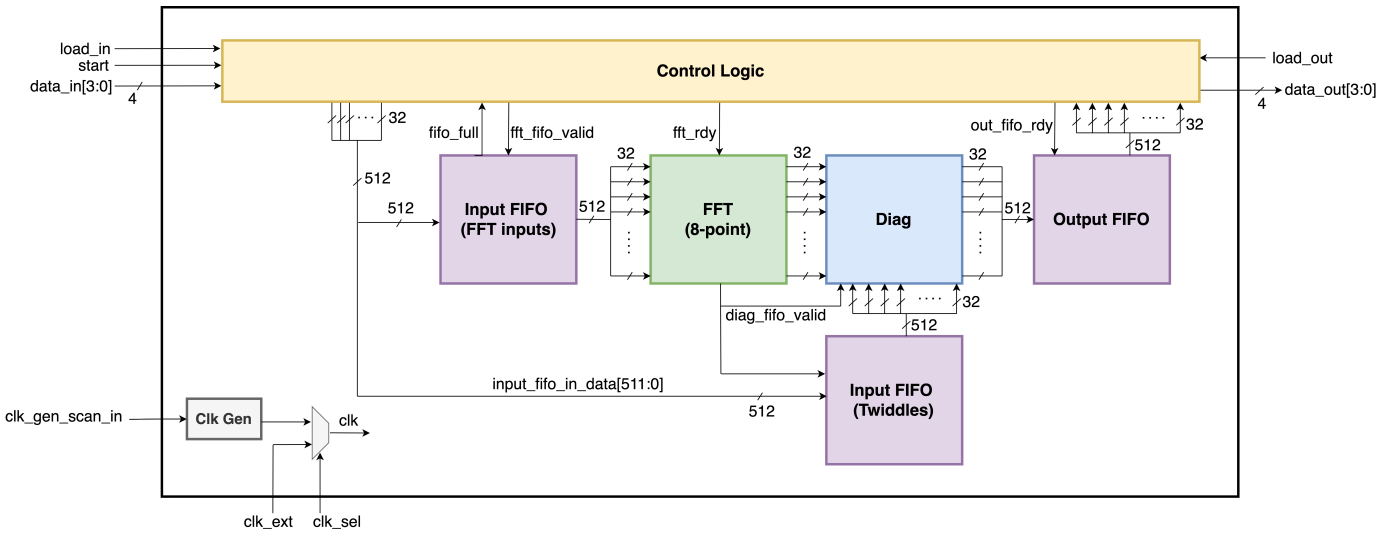


Fig. 5. System block diagram of test chip. The input FIFOs feed 16 single-precision numbers to the radix-8 FFT every cycle and output FIFOs capture 16 single-precision numbers from Diag. The chip can be run in a looping mode for power measurements.

follows and is illustrated in Fig. 2.

SPIRAL uses a script that first defines the parameters for the generated FFT hardware. Within the SPIRAL system an algorithm is chosen and the problem is represented in an Operator Language (OL) formula. Through a rewrite system, the OL formula is translated to lower-level DSLs for analysis, compiler optimizations, and pipeline register insertion until the generated code is wrapped in an RTL module.

For this first test chip, we hand design all other functionality including complex multipliers, FIFOs, control circuitry, and test infrastructure that surrounds the radix-8 FFT hardware. The final generated code is integrated with the other hardware modules and then passed to the ASIC design and verification process to produce a test chip that can be fabricated. We use a standard-cell based ASIC tool flow for synthesis and physical design, and verification is performed against our own Python golden models.

IV. SPIRAL-FFT ASIC

An initial test chip of a FFTW codelet accelerator kernel is designed and taped out in a TSMC 28 nm process. The core computation consists of a fixed radix-8 FFT followed by eight general twiddle factor multiplications. In this section we discuss the function and micro-architecture of each of the main components of the test chip.

A. Radix-8 FFT Architecture

The SPIRAL-generated FFT hardware is a fully unrolled radix-8 architecture, giving the highest possible throughput at one full vector of outputs per cycle. Additionally, the design uses single-precision floating point arithmetic to support a wider variety of scientific applications which may have different dynamic ranges.

As shown in Fig. 6, the FFT hardware implements exactly the signal flow diagram of an 8-point FFT where every arithmetic operation is mapped to a dedicated adder/multiplier. The dataflow is also optimized such that the multipliers in the FFT unit are constant multipliers with one fixed input since the twiddle factors within the radix-8 FFT are fixed constants. Pipeline registers are automatically inserted to ensure that each set of FFT computations remains aligned in the datapath. Each floating point adder is further pipelined to three stages: one for the exponent comparison, another for the mantissa addition, and a final stage for normalization. The floating point multiplier is pipelined to two stages where one stage performs the mantissa multiplication and the other completes the normalization stage. The radix-8 FFT is aggressively pipelined to 13 stages to increase throughput of the accelerator.

B. Diag Unit

The eight complex outputs of the FFT are fed directly to the Diag unit, which consists of eight complex multipliers to perform all complex twiddle multiplications in parallel. General twiddle multiplication at the end of the fixed 8-point FFT forms the FFTW equivalent of a ”twiddle codelet“. A complex multiplier is implemented using four floating point multipliers and two floating point adders. The design of each adder is exactly the same as those in the FFT core, however the multiplier is instead a two-input single-precision multiplier since the twiddle factors are not fixed constants in the Diag unit. All complex multiplies occur in parallel to maintain steady-state pipeline, the Diag unit is pipelined to a total of 5 stages.

C. Memory

In order to sustain steady-state pipeline operation, we design 18-stage shift register FIFOs which directly input and capture complex single-precision numbers to and from the FFT/Diag units every cycle. One set of FIFOs holds the input FFT data, another holds complex twiddle factors, and the final set holds

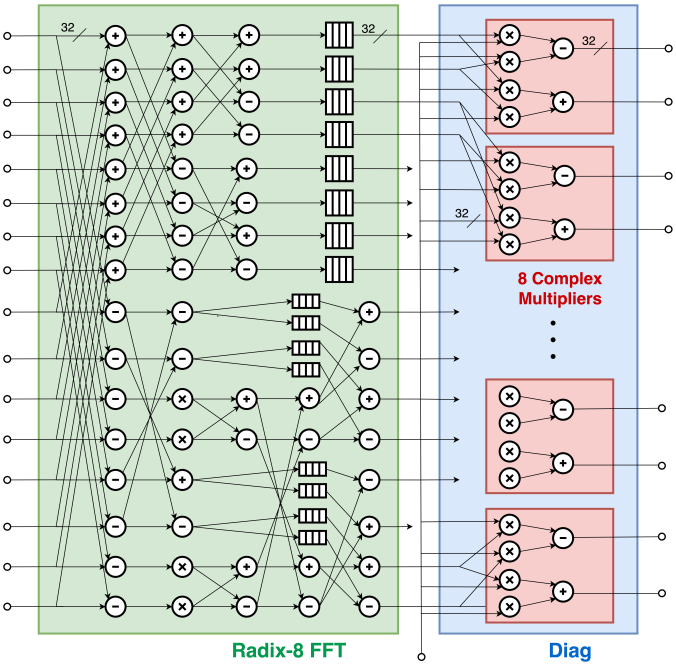


Fig. 6. Internal architecture of the Spiral-generated radix-8 FFT and Diag complex multiplier block. The FFT unit uses single-precision adders and constant multipliers. The Diag unit has 8 complex multipliers, where each complex multiplier has four multipliers and two adders.

the output data coming from the Diag block. The input FIFOs are designed so that their outputs can loop back to the input so that the system continuously run. The design contains a total of 3.375 kB of FIFO memory which is implemented as synthesized memories. The on-chip memory is designed in this way so that we can demonstrate the achievable throughput performance of the hardware kernel.

D. Control Unit

The control unit processes incoming data from off-chip through an asynchronous I/O protocol to load in test vectors to the FIFOs. We load 4-bit nibbles which the control unit then assembles together to form the 32-bit FP numbers which populate the FIFOs. The load-in data includes a set of bits allocated as metadata to inform the control unit which mode to operate the chip in. The FFT/Diag core can be configured to run in "compute" mode to run through one full set of test vectors in the input FIFOs and write to the output FIFO, or "looping" mode which continually runs the test inputs through the FFT/Diag units to obtain power measurements.

V. Measured Results

The FFT accelerator test chip is fabricated in a TSMC 28nm process and has a core size of 0.7mm x 0.7mm. The radix-8 FFT core occupies 0.078mm^2 and the Diag unit occupies 0.088mm^2, with the remaining area occupied by FIFO memory and control/test infrastructure. Figure 8b shows a more detailed area breakdown by functional block. Since the design is a fully unrolled architecture, it achieves a throughput of $8 \times f_{clk}$ complex samples per second.

Functionality is verified with test vectors consisting of 18 sets of eight complex single-precision floating point data and eight complex twiddle factors to completely fill up the input FIFOs. After running the computation we read out the contents of the output FIFO from the test chip and compare against the corresponding output obtained from HDL simulation. A micrograph of the die is shown in Fig. 7 and the chip implementation details are summarized in Table I.

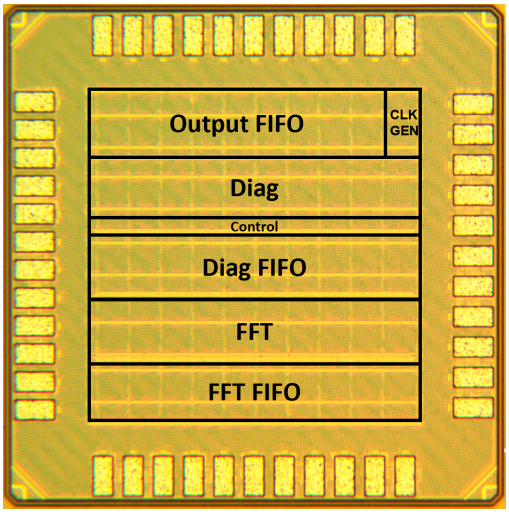


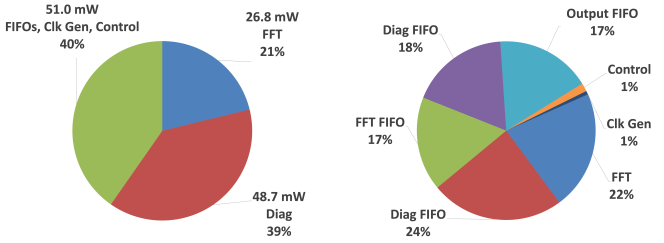
Fig. 7. Die micrograph of 1mm × 1mm Spiral-FFT test chip in TSMC 28nm. The chip has 45 I/O pads and a core area of 0.7mm × 0.7mm. FIFO memory holds a total of 3.375 kB of data.

Technology	TSMC 28nm
Core Supply	0.9 V
Chip Area	1 mm^2
Core Area	0.49 mm^2
Max. Frequency	270 MHz
I/O	45
Memory	3.375 KB (FIFOs)
Power	126 mW
Throughput	2.08 GS/s
FFT8 Latency	68 ns

TABLE I
Implementation Summary

At nominal 0.9V the chip runs at 260 MHz and consumes a total of 126 mW when computing 8-point FFTs and twiddle factor multiplications at a throughput of 2.08 Gsamples/sec. The FFT and Diag units consume 75 mW and the FIFOs and clock generator consume 51 mW. A detailed breakdown of the measured dynamic power consumption is also shown in Fig. 8a.

Across a core voltage range of 0.65V-1.15V, the chip successfully operates at 155-270MHz. The Shmoo plot in Fig. 9 demonstrates the full voltage-frequency operating points at which the test chip is functional. The first test chip tapeout using SPIRAL as a part of the design flow was successful and we look to expand on this work.



(a) Power (b) Area

Fig. 8. Breakdown of power consumption at 0.9V and area by block

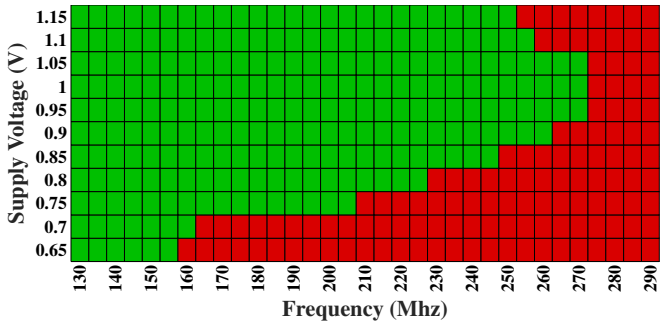


Fig. 9. Shmoo plot of SPIRAL-FFT accelerator. Green areas represent the points at which the chip is functional. At nominal 0.9V the chip runs at 260 MHz and at a supply range of 0.65-1.15V the chip operates at 155-270MHz.

VI. Discussion and Future Work

In this section we discuss a system model to integrate the accelerator into a RISC-V SoC and future directions to take this work.

A Multi-Core RISC-V SoC. We look towards designing a RISC-V SoC with a hardware codelet accelerator to build a high performance FFT processor that could be suitable for a number of different applications. In this section we discuss a system model which would enable one to incorporate the custom accelerator with low area overhead. Consider a multi-core RISC-V machine that contains the proposed codelet accelerator, dedicated FFT cores, system-management cores, on-chip cache, and an interface to access off-chip main memory. Suppose each core has a private, multi-banked L2 cache which the accelerator can compute out of.

For example, we may want to incorporate a double-precision radix-8 twiddle codelet accelerator into a multi-core RISC-V SoC. Such an accelerator could target applications in scientific HPC simulations such as PDE solvers. Assuming the codelet accelerator runs at its theoretical peak throughput, eight complex double-precision inputs and twiddle factors need to be fed every cycle. To enable this, we can allocate eight FFT cores each with a multi-banked L2 cache and four additional system management cores that keep the system running. The codelet accelerator can be attached to the FFT cores through a specialized interface such as the Rocket Chip [17] RoCC interface. With a multi-banked L2 cache that has enough banks, we can perform two reads from independent banks and a write to the third bank in parallel, where a bank would read/write a double-precision complex number per cycle. Another bank can then be used as a double buffer to read and write off-chip data into and out of L2 cache. The system can be further designed such that the accelerator compute time is balanced with the off-chip memory bandwidth to hide the memory latency.

SPIRAL. The test chip is the first to be fabricated using SPIRAL as a part of the design flow to generate the HDL source code for the FFT kernel. We look to further leverage the SPIRAL code generation system as a part of the design flow. First, SPIRAL can be used in a design generator that automatically produces the RTL-level description of the codelet accelerator. Second, the algorithmic knowledge of the SPIRAL generation system can be used to generate high performance FFT plans targeting the accelerator.

FFTW Compatibility. We currently have a model which performs software emulation of the hardware system. Parameters are passed to a software abstraction of the hardware at each invocation of a codelet through a set of 'descriptors' that are specified during plan construction. The descriptors are designed to be extensible enough to support the variety of functionalities included in the FFTW library. This allows the overall plan composition to be entirely described through a set of descriptors that specify the parameters that are passed to the hardware. The descriptors may include information on I/O base addresses, strides at which to access data, data dimensionality, data type or format, and more.

Additional hardware functionality can also be built around the accelerator. The hardware could be designed to process and launch kernel invocations, where each kernel launch is called in a pipelined fashion. We also look to add support for real and pruned FFTs and eventually target multi-dimensional FFTs as well.

VII. Conclusion

FFTs play a critical role in modern scientific and engineering applications. In this paper we proposed a hardware accelerator for an FFTW codelet that can be tied into a real application specific RISC-V system. The hardware can be invoked through a familiar FFTW user interface, maintaining the flexibility offered by the FFTW library. We presented a first look at the hardware system with the design of a test chip that accelerates a radix-8 FFT and twiddle multiplications and is partially designed using the SPIRAL code generation system. Our measured results demonstrate that the accelerator successfully operates at a high throughput of one vector per cycle. Ultimately we look to build an SoC that uses the accelerator to target FFT-based applications.

References

[1] H. G. Myung, "Introduction to single carrier FDMA," in *2007 15th European Signal Processing Conference*, 2007, pp. 2144–2148.

[2] M. Mathieu, M. Henaff, and Y. LeCun, "Fast Training of Convolutional Networks through FFTs," 2013. [Online]. Available: https://arxiv.org/abs/1312.5851

[3] D. F. Martin and P. Colella, "A cell-centered adaptive projection method for the incompressible euler equations," *Journal of Computational Physics*, vol. 163, no. 2, pp. 271–312, 2000.

[4] H. Moulinec and P. Suquet, "A fft-based numerical method for computing the mechanical properties of composites from images of their microstructures," in *IUTAM Symposium on Microstructure-Property Interactions in Composite Materials*, R. Pyrz, Ed. Dordrecht: Springer Netherlands, 1995, pp. 235–246.
[5] M. Frigo and S. Johnson, "The Design and Implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.
[6] S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, *Handbook of signal processing systems*. Springer, 2013.
[7] S. Liu and D. Liu, "Design Space Exploration of 1-D FFT Processor," *J. Signal Process. Syst.*, vol. 90, no. 11, p. 1609–1621, nov 2018. [Online]. Available: https://doi.org/10.1007/s11265-018-1393-4
[8] C.-L. Hung, S.-S. Long, and M.-T. Shiue, "A low power and variable-length FFT processor design for flexible MIMO OFDM systems," in *2009 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2009, pp. 705–708.
[9] X. Chen, Y. Lei, Z. Lu, and S. Chen, "A Variable-Size FFT Hardware Accelerator Based on Matrix Transposition," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 10, pp. 1953–1966, 2018.
[10] Texas Instruments, "FFT Implementation on the TMS320VC5505, TMS320C5505, and TMS320C5515 DSPs," 2013. [Online]. Available: https://www.ti.com/lit/an/sprabb6b/sprabb6b.pdf
[11] A. V. Oppenheim, R. W. Schafer, and J. R. Buck, *Discrete-Time Signal Processing (2nd Ed.)*. USA: Prentice-Hall, Inc., 1999.
[12] J. W. Cooley and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.
[13] Intel, "Math kernel library." [Online]. Available: software.intel.com/mkl
[14] Nvidia, "Nvidia cufft." [Online]. Available: https://developer.nvidia.com/cufft
[15] F. Franchetti, D. G. Spampinato, A. Kulkarni, D. Thom Popovici, T. M. Low, M. Franusich, A. Canning, P. McCorquodale, B. V. Straalen, and P. Colella, "FFTX and SpectralPack: A First Look," in *2018 IEEE 25th International Conference on High Performance Computing Workshops (HiPCW)*, 2018, pp. 18–27.
[16] F. Franchetti, T. M. Low, D. T. Popovici, R. M. Veras, D. G. Spampinato, J. R. Johnson, M. Püschel, J. C. Hoe, and J. M. F. Moura, "SPIRAL: Extreme Performance Portability," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1935–1968, 2018.
[17] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, Apr 2016. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html