

A High-Throughput, Adaptive FFT Architecture for FPGA-Based Space-Borne Data Processors

Kayla Nguyen, Jason Zheng, Yutao He and Biren Shah
Jet Propulsion Laboratory, California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099
(818) 354 - 5185
{Kayla.Nguyen, Xin.Zheng, Yutao.He, Biren.N.Shah}@jpl.nasa.gov

Abstract—Historically, computationally-intensive data processing for space-borne instruments has heavily relied on ground-based computing resources. But with recent advances in functional densities of Field-Programmable Gate-Arrays (FPGAs), there has been an increasing desire to shift more processing on-board; therefore relaxing the downlink data bandwidth requirements. Fast Fourier Transforms (FFTs) are commonly-used building blocks for data processing applications, with a growing need to increase the FFT block size. Many existing FFT architectures have mainly emphasized on low power consumption or resource usage; but as the block size of the FFT grows, the throughput is often compromised first. In addition to power and resource constraints, space-borne digital systems are also limited to a small set of space-qualified memory elements, which typically lag behind the commercially available counterparts in capacity and bandwidth. The bandwidth limitation of the external memory creates a bottleneck for a large, high-throughput FFT design with large block size. In this paper, we present the Multi-Pass Wide Kernel FFT (MPWK-FFT) architecture for a moderately large block size (32K) with considerations to power consumption and resource usage, as well as throughput. We will also show that the architecture can be easily adapted for different FFT block sizes with different throughput and power requirements. The result is completely contained within an FPGA without relying on external memories. Implementation results are summarized.

I. INTRODUCTION

In recent years, the trend in digital signal processing (DSP) applications for space-borne instruments is to migrate more data processing power on-board using digital logic devices such as field programmable gate arrays (FPGAs). Compared to software-based processing that has been heavily relied on in the past, this trend allows real-time processing for computation-intensive algorithms. A major part of this growth requires an increase in the performance of the on-board Fast Fourier Transform (FFT) core. The improvements in FFT capabilities provide a more accurate processing on-board; thus leading to less downlink bandwidth required, which is a highly desired trait in the space industry.

There have been a number of papers describing small-size FFT designs optimizing for low power consumption and high-speed operation of 16-point [1], 256-point [2], and 1024-point [3] FFTs. A good amount of discussion is found in literature focusing on the optimization of computational building blocks of FPGA-based FFTs such as multipliers [4] and reducing the number of memory references [5].

This paper will describe the architecture and implementation of a 32,768 (32K) complex input FFT that meets stringent latency requirements within the resource availability of the Xilinx Virtex-5 XC5VFX130T FPGA. We first discuss the architectural options and comparisons for 32K FFTs along with the overall data flow scheme. A deeper description of the hardware design for the Radix-2 butterfly structure, the FIFO memory, and the twiddle factors follow. Next, adaptability schemes to meet different requirements are proposed for the Multi-Pass Wide Kernel (MPWK) FFT. Finally, resource usage results and power consumption estimations are provided.

A. Requirements

The processing power for a future space mission requires a 32K FFT plus a 32K inverse FFT (iFFT), each to be performed within 156 μ s on a Xilinx XC5VFX130T FPGA. The data processing requirements constraints are listed in Table I.

TABLE I
DATA PROCESSING REQUIREMENTS AND CONSTRAINTS

Parameter	Requirement
FFT size	32K Complex Input FFT + 32K Complex Input Inverse FFT
Maximum Latency	156 μ s
Input Data Speed	64MHz
FPGA Technology	Xilinx XC5VFX130T
Available # of 36K BRAM blocks	298 / 10.728Mb
Available # of DSP48 blocks	320

Since both the FFT and iFFT must fit onto one FPGA, the FPGA resources must be split between the FFT and the iFFT blocks. Therefore, there are 156 BRAMs (5.36Mb) and 160 DSP48 blocks available for the FFT calculation.

II. DISCRETE FOURIER TRANSFORM METHODS

An N sample Discrete Fourier Transform (DFT) is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\omega_N^{nk},$$

where $k = 0, 1, \dots, N - 1$ and $\omega_N = e^{-j2\pi/N}$, also called the “twiddle factor” [6]. By using this definition to calculate the DFT, the number of operations needed is $\mathcal{O}(N^2)$ [7]. For a large size N , a simplification is needed to reduce the number of operations.

A. Fast Fourier Transform

In 1965, Cooley and Tukey published a convenient and fast algorithm to compute the Discrete Fourier Transform on a computer [7], and the algorithm is called the Fast Fourier Transform (FFT). The number of operations required is reduced to be proportional to $N \log(N)$ compared to the traditional method that required N^2 operations. The simplest and common form of the Cooley-Tukey FFT algorithm is the Radix-2 algorithm. Figure 1 shows the data flow for an 8-point decimation-in-frequency FFT using Cooley-Tukey’s Radix-2 algorithm. Note that the input to the FFT structure is in normal order, while the output is in reverse binary order.

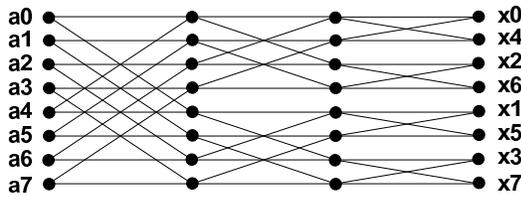


Fig. 1. 8-point Radix-2 FFT Using the Cooley-Tukey Algorithm

B. Singleton’s Parallel FFT Architecture

Using the Cooley-Tukey FFT algorithm as a reference, Singleton devised the parallel FFT architecture in 1967 [8] in which the traditional FFT data flow diagram is rearranged such that each stage has the same geometry (see Figure 2). This arrangement allows the implementation of the FFT to have sequential data access, as well as being able to use the same butterfly every pass (refer to Figure 4).

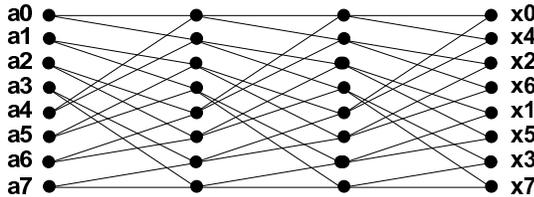


Fig. 2. Singleton’s Rearranged Data Flow Graph for an 8-point FFT

An alternative data flow graph showing the butterfly number in each of the FFT stages is shown in Figure 3. Notice that with each stage of the FFT pass, butterfly B1 takes instantaneous inputs from a0 and a4, and the outputs are mapped to the first and second outputs, which is similar for butterflies B2, B3, and B4. This arrangement allows the three stages to be collapsed down to one stage with four butterflies, as illustrated in Figure 4. The data passes through the single structure of butterflies three times to complete the 8-point FFT.

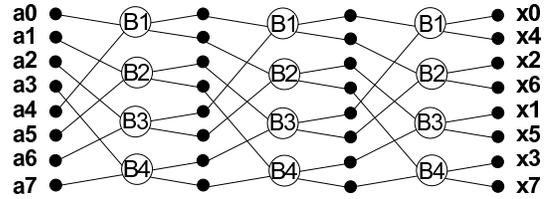


Fig. 3. Singleton’s Data Flow Graph for an 8-point FFT Alternate View

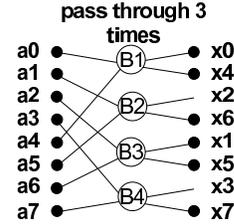


Fig. 4. Singleton’s Data Flow Graph for an 8-point FFT Collapsed Stages

C. Singleton’s Single Butterfly Method

Singleton’s single butterfly method further simplifies the FFT architecture by collapsing the four butterflies in Figure 4 into one butterfly, as shown in Figure 5. The inputs are divided into two FIFOs, the first four inputs (a0, a1, a2, a3) are stored into the first FIFO, while the last four inputs (a4, a5, a6, a7) are stored into the second FIFO. The output are denoted as $x_{b,1,k}$ and $x_{b,2,k}$, where b is the butterfly number from Figure 4 and k is the FFT stage number from the set $\{1, 2, 3\}$ for an 8-point FFT.

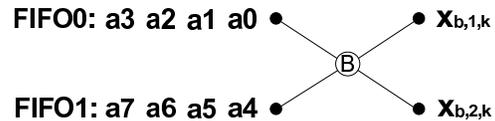


Fig. 5. Singleton’s Single Butterfly Data Flow Graph for an 8-point FFT

The data and FIFO arrangement for the single butterfly method is illustrated in Figure 6. At time t_0 (Figure 6(a)), we assume that all of the data a0-a7 is filled into FIFO0 and FIFO1 in sequential order. Notice that FIFO0 is 1.5 times larger than FIFO1. At time t_1 (Figure 6(b)), the butterfly has processed inputs a0 and a4, and have produced outputs $x_{1,1,1}$ and $x_{1,2,1}$ which are stored into FIFO0. At time t_2 (Figure 6(c)), the butterfly has processed inputs a1 and a5, and the outputs $x_{2,1,1}$ and $x_{2,2,1}$ are stored into FIFO0. At this point, the need for a larger FIFO size for FIFO0 can be seen. At time t_3 (Figure 6(d)), the butterfly has processed inputs a2 and a6, and the outputs $x_{3,1,1}$ and $x_{3,2,1}$ are stored into FIFO1. This is the first step where an output of the butterfly is store into FIFO1. At time t_4 (Figure 6(e)), the butterfly has processed inputs a3 and a7, and the outputs $x_{4,1,1}$ and $x_{4,2,1}$ are again stored into FIFO1.

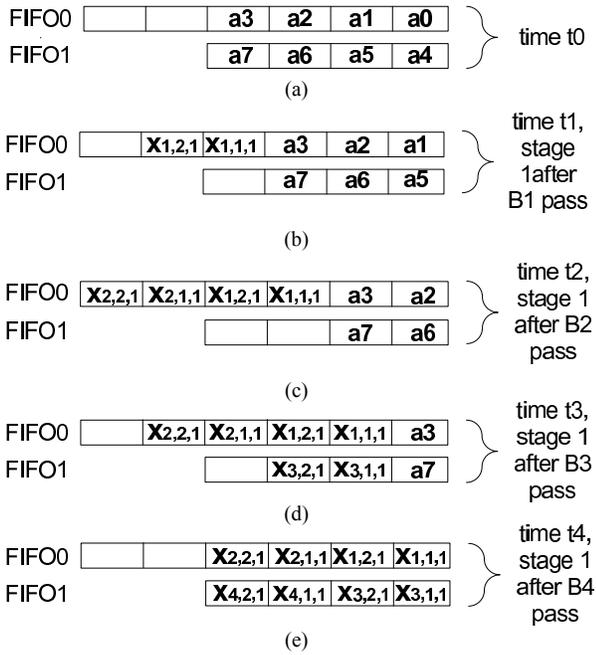


Fig. 6. FIFO scheme for Singleton's Single Butterfly Method. (a) Time t_0 , input data. (b) Time t_1 , a_0 and a_4 processed by the butterfly. (c) Time t_2 , a_1 and a_5 processed by the butterfly. (d) Time t_3 , a_2 and a_6 processed by the butterfly. (e) Time t_4 , a_3 and a_7 processed by the butterfly.

III. MULTI-PASS WIDE KERNEL FFT (MPWK-FFT) ARCHITECTURE

Using Singleton's single butterfly method, the calculated latency for a 32K-point FFT is 245,760 cycles. To meet the latency requirement of 156 μ s with this method, the FPGA clock would need to operate at 1.5GHz. Operating an FPGA at this speed is not only impractical, but unachievable. The slow throughput of this method is due to the fact that there is only one butterfly calculating 32,768 operations 15 times. In order to reduce the number of operations in the butterfly, the Multi-Pass Wide Kernel structure is developed. The MPWK-FFT structure allows users to expand Singleton's single butterfly structure into a wide kernel of multiple butterflies in parallel.

A. Architecture Comparison

In order to determine how many parallel butterflies are needed in the MPWK-FFT architecture, a comparison between the limitations of the Xilinx XC5VFX130T FPGA and FFT latency is performed. Figure 7 shows the comparison between latency vs. the number of real multipliers. From the figure, the maximum number of multipliers is shown by the vertical line (160), and the maximum allowable latency is shown in the horizontal line (156 μ s). A comparison between 16 butterflies, 32 butterflies, and 64 butterflies schemes are performed. The only scheme which meets both constraints is the 32 butterflies scheme.

B. 32K MPWK-FFT Data Flow Diagram

The proposed architecture of the 32K Radix-2 Complex-Input MPWK-FFT, shown in Figure 8, is a combination of

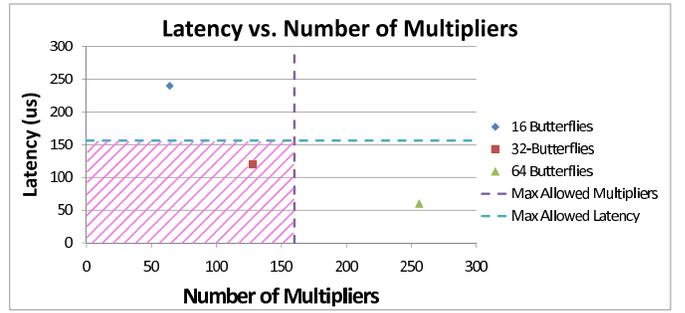


Fig. 7. Comparison between the Number of Multipliers versus Latency at 64MHz with Optimal Shaded Region

Singleton's structure in Figure 4 and Figure 5. The proposed structure contains 32 butterflies in parallel and 64 complex data FIFOs to store the input and use as memory storage between stages. The inputs to the structure is divided in 64 groups, with input data filling the FIFOs in normal order as shown in Figure 8. The output is valid in reverse binary order after the data passes through the structure 15 times.

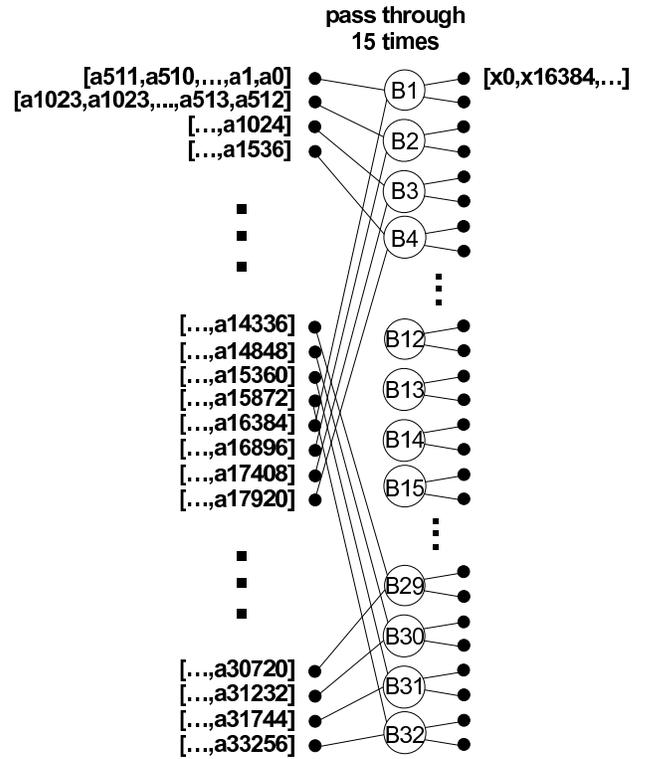


Fig. 8. MPWK-FFT Data Flow Graph for 32K FFT

C. 32K MPWK-FFT Hardware Architecture

The overall 32K MPWK-FFT architecture is shown in Figure 9. The major functioning blocks are: the 32-butterfly structure, the 64 FIFOs for the real inputs, the 64 FIFOs for the imaginary inputs, the complex twiddle factor memory storage, and the FIFO logic.

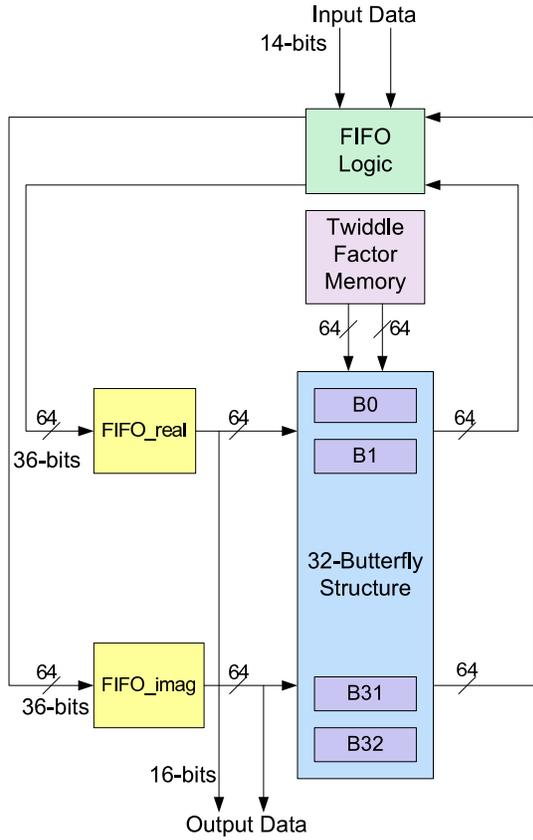


Fig. 9. Modular View of 32K MPWK-FFT Architecture

1) *Radix-2 Butterfly Structure*: Each radix-2 butterfly (B1, B2, etc.) in Figure 8 is shown in detail in Figure 10. The interface signals are:

- Two real inputs - $a0_r$ and $a1_r$
- Two imaginary inputs - $a0_i$ and $a1_i$
- One real and one imaginary twiddle factors - c_r and c_i
- Two real outputs - $x0_r$ and $x1_r$
- Two imaginary outputs - $x0_i$ and $x1_i$

The outputs are given as:

$$\begin{aligned} x0_r &= a0_r + a1_r \\ x0_i &= a0_i + a1_i \\ x1_r &= c_r(a0_r - a1_r) - c_i(a0_i - a1_i) \\ x1_i &= c_i(a0_r - a1_r) + c_r(a0_i - a1_i) \end{aligned}$$

Each butterfly contains two complex multiplications, or equivalently four real multiplications. For a butterfly structure with 32 butterflies, the total number of multipliers used is 128.

2) *FIFO Memory and Logic*: There are 64 FIFO blocks to store the real data and 64 FIFO blocks to store the imaginary data, totaling to 128 FIFOs. Referring to Section II-C and Figure 5 as reference for an extension into 32 butterflies, the output $x0_r$ and $x1_r$ of B1 first fills FIFO0 for 512 samples, then it starts to fill FIFO1 for 512 samples until the end of the current stage (see Figure 8). In order for the output of

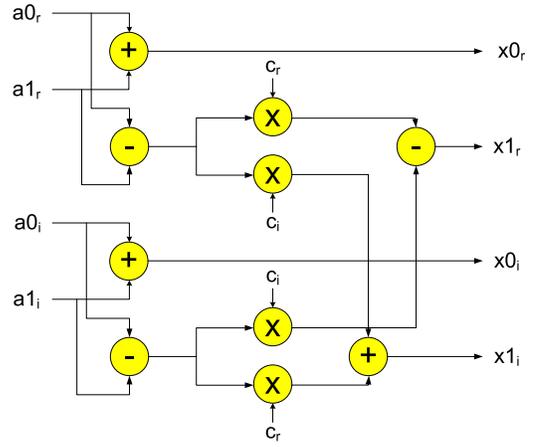


Fig. 10. Radix-2 Butterfly Structure

each butterfly to go to the correct FIFO block, routing logic is placed in the FIFO logic block. Since FIFO0, FIFO2, etc. is filled first with the output of the butterflies at twice the rate of data read from the FIFO, even FIFO depths must be larger than odd FIFOs by 50%. We calculate the depth of each FIFO using equations 1 and 2. Even FIFOs have depth:

$$D_{\text{FIFOeven}} = \frac{1.5N}{2 \times N_{\text{butterflies}}}, \quad (1)$$

which comes out to be 768 deep for an FFT size of 32K. Odd FIFOs have depth:

$$D_{\text{FIFOodd}} = \frac{N}{2 \times N_{\text{butterflies}}}, \quad (2)$$

which comes out to be 512 deep for an FFT size of 32K. Therefore, the total amount of memory needed to store the data is $768 \times 64 + 512 \times 64 = 81,920$ samples. Using the 36K blocks of BRAM, each sample (real and imaginary solutions from the FFT are stored independently) contains 36-bits for a total memory size of 2.95Mbits.

3) *Twiddle Factors*: In order to achieve the required latency of 156us, each twiddle factor must arrive at the multiplier within one clock cycle. Thus, the twiddle factors are stored in an internal ROM lookup table inside the FPGA. For the 32 butterflies MPWK-FFT architecture, 32 strings of real twiddle factors and 32 strings of imaginary twiddle factors are required. Each string has 512 entries, corresponding to the depth of each FIFO. Each twiddle factor sample is 18-bits wide. This corresponds to a total memory size of $18 \times 32 \times 2 \times 512 = 589.9\text{Kbits}$.

IV. ADAPTABILITY OF MPWK-FFT ARCHITECTURE

The block size of the MPWK-FFT architecture can be easily changed by adjusting two parameters. These parameters include the depth of the input FIFO and the width of the butterfly kernel. These two parameters, in turn, will decide the number of multipliers, the number of passes, and the latency. As with the 32K block size, the resource usage, throughput,

and peak dynamic power can be traded through different configurations of the parameters for all block sizes. To help understand the trades, a case study of a 16K FFT is provided in this section.

A. FIFO Depth

The first way to change the block size of the MPWK-FFT is to increase or decrease the depth of the FIFO blocks. For example, to change the 32K block size FFT to a 16K block size, one can reduce the depth from 512/768 (odd/even blocks) to 256/384. As a side effect, the number of passes is also reduced from 15 to 14.

Since the width of the kernel is not changed, the number of multipliers remains the same from the 32K block size. For the same reason, the input/output routing of the kernel is the same as the 32K FFT. On the other hand, as the number of passes is decreased by 1, and the number of data points per pass is halved, the latency is only $\frac{14 \times 256}{15 \times 512} = 47\%$ of that of the 32K FFT.

We now refer to this approach as the depth approach.

B. Kernel Width

Instead of changing the depth of the FIFO, the width of the kernel can be changed to accommodate the new size. Following the example of the 16K FFT, keeping the FIFO depth fixed at 512/768, the width of the kernel can be halved to 16 butterflies.

As a result of the smaller kernel width, the number of required multipliers is also reduced by half. Consequently, the input/output routing of the kernel must be adjusted accordingly. For example, the first butterfly's input used to feed from FIFO #1 and #33; it now should feed from FIFO #1 and #17. Fortunately the routing change can be arranged in a systematic and straightforward fashion.

On the other hand, the number of passes is reduced by 1 but the number of data points per pass remain the same. Hence the latency of the 16K FFT is only 512 clocks less than the latency of the 32K FFT, which corresponds to a decrease of only 7%.

We now refer to this approach as the width approach.

C. Resource, Latency, and Dynamic Power Trade

From Sections IV-A and IV-B, we see that the trade between the depth and the width is fundamentally a trade between latency and multiplier usage. The faster the FFT needs to be, the more multipliers are required, and vice versa. Note that the total size of the FIFO is the same for a fixed FFT block size, regardless of the trade.

As for the dynamic power consumption of the two approaches, trading depth for width is almost a fair trade. Assuming that the multipliers' clocks can be individually turned off while idle, reducing the depth shortens the amount of active time of the multipliers by half, but uses twice as many multipliers as the approach of reducing the width. As long as both approaches can meet the throughput requirements levied on the FFT, the average dynamic power or energy consumption is not expected to differ between the two approaches.

However, there is a difference with respect to the peak dynamic power consumption; specifically the depth approach uses roughly twice the peak dynamic power as the width approach. Although the average dynamic power is roughly the same, the higher concentration of heat dissipation of the depth approach (due to dynamic power) could lead to a higher die temperature, and therefore higher static (leakage) power consumption.

D. Load Balance

The static power consumption of an FPGA with a fixed die area is largely tied to the die temperature, which, in turn, can be determined based on ambient or case temperature and the distribution of the heat resulting from the dynamic power dissipation of the FPGA. Spatial and temporal concentration of such dynamic power creates local hotspots that increase the die temperature and consequently static power consumption.

To reduce the hotspot creation, a third adaptation approach is presented. This approach is similar to the width-reduction approach in that the FIFO depth is not changed, and the kernel width is reduced by half. However, the difference is that the work done by each butterfly is carried out by two butterflies in an alternate fashion. Each butterfly from the width approach is replaced by two butterflies with a multiplexer on the output ports. This is referred to as the load-balance approach.

Compared to the width and depth approach, the load-balance approach does not reduce the overall resource usage, latency, peak or average dynamic power. However, this approach spreads the heat dissipation of a single butterfly to two butterflies. So the dynamic power consumed per individual butterfly/multiplier is effectively reduced by half. As a result, both spatial and temporal concentration of the dynamic power dissipation is reduced. We believe that this will lead to a lower die temperature and therefore lower static power.

V. APPLICATION AND RESULT

The adaptive FFT architecture presented in this paper will be part of ISAAC iCore library, and is expected to be configured in an FPGA-based data processing hardware such as iBoard [9].

The motivation of ISAAC (Instrument Shared Artifact for Computing) technology [10] is to provide a highly capable, highly reusable, modular, and integrated FPGA-based common instrument control and computing platform that can be shared by multiple Earth Science and Planetary Exploration instruments. This reusable framework offers an unprecedented combination of adaptability, computation power, I/O bandwidth, digital interface standards, and data processing capability in a single common low mass/power and small form factor platform with significantly reduced non-recurring cost and risk to Earth Science instruments such as SMAP Mission (Soil Moisture Active-Passive) and other future NASA Planetary Exploration instrument of diverse requirements.

ISAAC's unique technical innovations are embodied in its six key components: iBoard - the FPGA-based hardware substrate; iCore - the library of Register-Transfer-Level

(RTL) Intellectual Property (IP) cores implementing common computationally-intensive instrument control and computing functions, such as the MPWK-FFT presented in this paper; iPackage - the collection of software functions that implements common non-computationally-intensive instrument control and computing functions; iBus - the standard and unified hardware/software interface; iBench - the suite of benchmark instrument data streams for performance validation and tuning of a completely-configured system; and iTool - the integrated tool-chain providing a familiar and end-to-end design flow for digital system designers. Collectively, ISAAC provides instrument electronics designers with a reusable and integrated framework that enables to configure a complete instrument control and computing system on a per-application-basis to match various instrument requirements.

A. Resource Usage Results

Resource usage results are gathered for the Xilinx XC5VFX130T. Results are summarized in Table II.

TABLE II
32K MPWK-FFT RESULT

Parameter	Used / Available	%
# of LUTs	23,417 / 81,920	28%
# of Registers	12,801 / 81,920	15%
# of Slices	7,552 / 20,480	36%
# of DSP48s	128 / 320	40%
# of BRAMs	128 / 298	43%

Note that all of the resource usage percentage is less than 50%. The maximum achievable clock speed for FPGA speed grade 1 is 103.4MHz and speed grade 2 is 114.6MHz, which meets our input data rate of 64MHz. The 32K MPWK-FFT has a total latency of only 120us while operating at a 64MHz clock which meets the latency requirement of 156us.

B. Power Consumption Results

Power consumption estimation is made using Xilinx XPower tool basing off of a clock running at 100MHz at industrial temperature grade. The estimation is made for the Xilinx XC5VFX130T for speed grades 1 and 2. The power consumption results are shown in Table III.

TABLE III
32K MPWK-FFT POWER CONSUMPTION

Parameter	XC5VFX130T-1	XC5VFX130T-2
Dynamic	3.1W	3.1W
Quiescent	2.4W	2.6W
Total	5.6W	5.7W

VI. CONCLUSION

The architectural design decisions for the 32K MPWK-FFT presented in this paper are made based on the stringent performance requirements given in Table I. The proposed design meets the 156us latency requirement, while utilizing only half of the Virtex-5 FX130T FPGA resources. Three adaptability MPWK-FFT schemes are present for designs that require either a different FFT size, different latency constraints, or a different FPGA architecture. Two different approaches are proposed to adapt the current FFT architecture to other block sizes with consideration to constraints such as latency, resource usage, and dynamic power. Adaptability with considerations for die temperature is also discussed, and a third approach is proposed that will lead to a more distributed heat dissipation and lower leakage power. This proposed FFT architecture is a part of the iCore library for the ISAAC technology.

ACKNOWLEDGMENT

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

REFERENCES

- [1] M. Kannan and S. Srivatsa, "Low power hardware implementation of high speed FFT core," *Journal of Computer Science*, vol. 3, no. 6, pp. 376–382, 2007.
- [2] B. S. Son, B. G. Jo, M. H. Sunwoo, and Y. S. Kim, "A high-speed FFT processor for OFDM systems," *Circuits and Systems, IEEE International Symposium on*, vol. 3, pp. 26–29, May 2002.
- [3] B. Baas, "A low-power, high-performance, 1024-point FFT processor," *Solid-State Circuits, IEEE Journal of*, vol. 34, no. 3, pp. 380–387, Mar 1999.
- [4] L. S. Cheng, A. Miri, and T. H. Yeap, "Efficient FPGA implementation of FFT based multipliers," in *Electrical and Computer Engineering, 2005. Canadian Conference on*, May 2005, pp. 1300–1303.
- [5] M. A. Aboleaze and A. I. Elnaggar, "Reducing memory references for FFT calculation," *International Conference on Computer Design, Proceedings of*, June 26–28 2006.
- [6] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Prentice-Hall, Inc., 1975.
- [7] 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, April 1965.
- [8] R. Singleton, "A method for computing the fast fourier transform with auxiliary memory and limited high-speed storage," *Audio and Electroacoustics, IEEE Transactions on*, vol. 15, no. 2, pp. 91–98, Jun 1967.
- [9] Y. He and M. Ashtijou, "iBoard a highly-capable, high-performance, reconfigurable FPGA-based platform for flight instrument digital electronics," in *NASA/ESA Conference on Adaptive Hardware and Systems*, vol. in Submission, June 2010.
- [10] Y. He, C. Le, J. Zheng, K. Nguyen, and D. Bekker, "ISAAC a case of highly-reusable, highly-capable computing and control platform for radar applications," in *IEEE 2009 Radar Conference*, May 2009.