Super-sized Multiplies: How Do FPGAs Fare in Extended Digit Multipliers?

Stephen Craven, Cameron Patterson and Peter Athanas
Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic Institute and State University, Blacksburg, VA 24061
{scraven | cdp | athanas@vt.edu}

Abstract
Extended digit multiplication can be an effective benchmark for comparing contemporary CPUs to other architectures and devices. The Great Internet Mersenne Prime Search (GIMPS), a distributed computing effort to find large prime numbers, has produced highly optimized code for multiplying large, multimillion digit numbers on Pentium processors. This paper presents a hardware large integer multiplier implemented in a Xilinx FPGA. The design, which utilizes an all-integer Fast Fourier Transform, is compared to a Pentium processor running the optimized GIMPS code. The results show that while a speed-up is achieved by the FPGA, the cost of the hardware likely outweighs any performance benefit.

I. Introduction

The ability to implement algorithms directly in custom hardware has made FPGAs attractive to certain niche areas of high-performance computing, such as digital signal processing where parallelism inherent in the algorithms is exploited to obtain significant speed-ups compared to general-purpose computers. However, applications that require or can benefit from high-speed, high precision floating-point multipliers available in modern microprocessors are typically not thought of as arenas in which FPGAs can effectively compete. One such arena is large integer multiplication (on the order of millions of digits) – an operation used extensively by the Great Internet Mersenne Prime Search (GIMPS) [1].

GIMPS is a distributed computing effort to find Mersenne primes – primes of the form $2^q - 1$, where q is also prime. GIMPS recently succeeded in discovering the largest known prime number containing over seven million digits [2]. While Mersenne primes have a use in random number generators, the computationally intensive process of verifying the primality of a Mersenne number is commonly used as a performance benchmark. The algorithm at the heart of GIMPS, the Lucas-Lehmer test, involves repeated squaring of a number modulo the Mersenne number whose primality is being tested. Special techniques, based on the Fast Fourier Transform (FFT), are used to perform the squaring as traditional algorithms scale poorly to numbers with millions of digits.

FFT-based multiplication treats each multiplier input as a sequence of smaller, conveniently sized integers instead of one, multimillion-digit number. Linear convolution of these input sequences carries out the multiplication. As convolution is a simple element-wise multiplication in the frequency domain, the input sequences are run through an FFT, then multiplied element-wise before undergoing an inverse FFT. The GIMPS code is optimized at the assembly level for the Pentium processor to make extremely efficient use of the floating-point multiplier for computing the FFTs. This code optimization makes large integer multiplication a valid benchmark for comparing FPGAs to Pentiums in high-precision multiply intensive applications.

The specific FFT-based multiplication algorithm used by the GIMPS project is the Irrational-Base Discrete Weighted Transform (IBDWT) [3]. Squaring a large integer using the IBDWT involves first constructing a variable-base, power-of-two length sequence from the large number. This sequence is multiplied element-wise with a weight sequence. After computing the FFT of this weighted sequence, element-wise squaring of the sequence occurs. An inverse FFT is performed on the squared, weighted

sequence followed by weight removal. A carry release step is required to return each digit of the sequence to its proper size. Finally, the Mersenne modular reduction is performed on the sequence.

The paper is organized as follows. In Section II, different algorithms for computing the FFT are evaluated. Section III presents the FPGA hardware accelerator design. The implementation's performance is compared to that of a Pentium in Section IV. Conclusions and directions for future work appear in Section V.

II. FFT Algorithm Selection

Several different FFT hardware implementations were considered including a traditional floating-point FFT, an all-integer FFT and a Fast Galois Transform (FGT). The algorithms were compared based on estimated performance when implemented in a Xilinx Virtex-II Pro 100 FPGA.

A. Floating-point FFT

The GIMPS implementation of the Lucas-Lehmer test uses highly optimized double precision floating-point FFTs. While no current FPGA contains dedicated hard-wired floating-point multipliers, several families include dedicated integer multipliers that can be combined to form floating-point multipliers.

Floating-point implementations suffer from round-off and truncation errors. There is a limit to the rounding and truncation errors that can occur before the resulting integer convolution is incorrect. To verify that incorrect rounding did not occur GIMPS performs checks after each iteration. Depending on the architecture, these checks add one to five percent to the run time - assuming no errors are found. If an error is found, the iteration is repeated with a longer FFT length.

Using a Xilinx Virtex II Pro 100 with its 444, 17x17 unsigned multipliers an IEEE standard double precision floating-point multiplier could be constructed out of sixteen integer multipliers. A near-double precision floating-point multiplier having only 51 bits in the mantissa could be constructed using nine embedded multipliers. Forty-nine such multipliers could fit on the device. However, as a traditional FFT uses complex arithmetic, four real multiplications are needed to produce a single complex multiply reducing the multiplier count to only twelve complex multipliers. Using performance numbers from the GIMPS project, testing a 40 million-bit number would require a transform length of 2 million [4]. When using all twelve complex multipliers in parallel, 3.7 million clock cycles would be required.

B. All-integer FFT

A more elegant solution would be one that uses the FPGA multipliers as integer multipliers while avoiding the round-off errors altogether. There exist integer Fourier Transforms that, through the use of modular arithmetic, map real integers to real integers. These all-integer transforms, also known as Number Theoretical Transforms, have reduced complexity variants similar in structure to the FFT. It should be noted that the modular arithmetic of this all-integer FFT does not allow for a meaningful analysis of the frequency spectrum; however, convolution and correlation rules still apply.

All arithmetic in an all-integer Fourier Transform is performed modulo a special prime number. This prime must have roots of unity to enable computation of the Fourier Transform. The order of these roots determines the allowable length of the transform. The requirements for an all-integer FFT are stricter. To allow for long FFT runs, a prime that has roots of one with orders that are high powers of two is required. Furthermore, to implement an all-integer IBDWT, a prime modulus that has roots of two with

orders that make desirable transform lengths is needed. An additional requirement is that the prime modulus be of a form that allows the modular reduction to be easily computed without a divide operation.

Craig-Wood [5] has ported the GIMPS algorithm to the ARM processor, which lacks a floating-point unit. Craig-Wood uses primes of the form $2^k - 2^m + 1$. For certain values of k and m these primes meet the criteria set forth above for an all-integer transform. A modulo reduction with such a prime simplifies to a few additions and subtractions.

Suitable primes of this form [6] are shown in Table 1 along with the cycle count required by the FFT engine per each squaring iteration. The table shows the number of multipliers that can be placed on the Virtex-II Pro 100 and the FFT length required to test a 12 million-digit number for different prime moduli. From this table the prime $2^{64} - 2^{32} + 1$ is the most attractive candidate in terms of minimum cycle count. This prime has the additional benefit of using 64 bits per data element, matching the SDRAM bus width and the maximum multiplier size that can be generated by the Xilinx CoreGen tool.

Table 1. All-integer FFT Prime Modulus Comparison

Prime	**# Multipliers**	**FFT Length**	**Iteration time**
$2^{47} - 2^{24} + 1$	49	4 (10^6)	1.9 (10^6) cycles
$2^{64} - 2^{32} + 1$	26	2 (10^6)	1.7 (10^6) cycles
$2^{73} - 2^{37} + 1$	17	2 (10^6)	2.6 (10^6) cycles
$2^{113} - 2^{57} + 1$	9	1 (10^6)	2.3 (10^6) cycles

The selected prime, $2^{64} - 2^{32} + 1$, allows for DWT lengths of up to 2^{26} or 64 million. Filling the Virtex-II Pro 100 device with modular multipliers would allow an FFT iteration to be completed in 1.7 million cycles.

C. Fast Galois Transform (FGT)

An extension on the previous method uses complex modular integer arithmetic of the form $a + ib \pmod{p}$ where p is a Mersenne prime. The advantage of the complex math required by this method is that the real input sequence can be nested into a complex sequence of half the original length. Furthermore, modular reductions by Mersenne Primes are easy to implement in hardware, consisting of one addition of size p with another possible addition by a single bit. Two Mersenne primes suitable for such an application are $2^{61} - 1$ and $2^{89} - 1$.

Using $2^{89} - 1$, after nesting the real sequence into a half-length complex sequence, ten million digit numbers can be tested using only a half-million point FFT. However, as complex multiplications are required, only three complex multipliers fit on the chip. Similarly for $2^{61} - 1$, only six complex multipliers can be implemented. A simple analysis shows that while a Galois Transform can reduce the FFT length by a factor of two, the limited multiplier count reduces throughput to less than an all-integer transform.

D. Algorithm Comparison

Other transform methods were briefly considered including the Winograd Transform [7]. The Winograd Transform, with its provable minimum of multiplications, places limits on the runs lengths – namely that the transform length be factorable into specific allowable lengths. These restrictions not only limit the

maximum run length but also prevent the FFT length from containing high powers of two as required by the IBDWT algorithm.

From a cycle count perspective the all-integer transform was selected for hardware implementation, achieving a 54% and 48% reduction in cycle count compared to the floating-point FFT and FGT, respectively. Although these results are solely cycle based, as all FFT designs require the construction of comparably sized large multipliers and each design would make use of the entire device, it is reasonable to assume that critical path timing would be comparable between designs.

III. Hardware Design

The computational building block of any FFT is a butterfly operation. Figure 1 shows the hardware design for an all-integer radix-2 butterfly. Sixteen of the 17x17 multipliers on the FPGA are combined and pipelined to form a single 64-bit multiplier. Delay registers are inserted in the top path of the butterfly to account for the six-cycle latency of the multiplier. Modular reductions occur after each operation; however, only the multiplication requires a complete reduction as the subsequent additions can at most add one bit to the result's size. This butterfly is fully pipelined, producing a result every clock cycle with a latency of ten cycles.

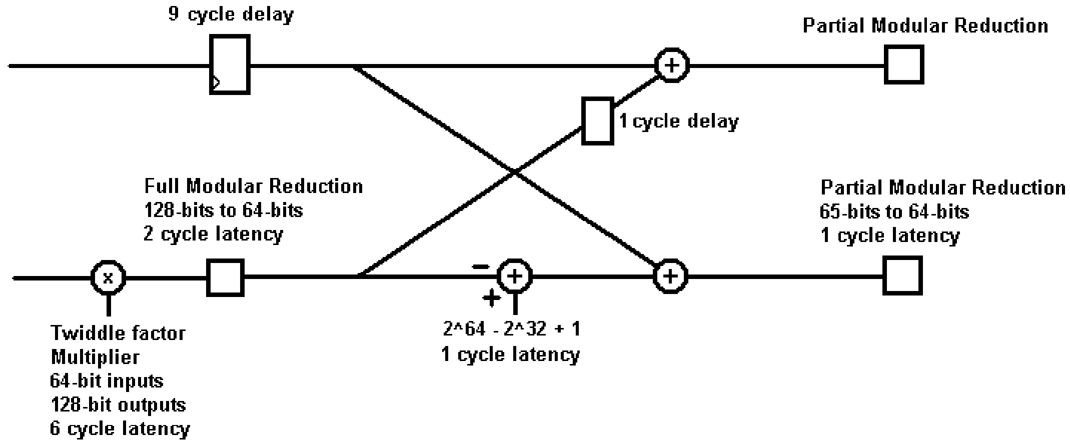

Figure 1. All-integer FFT Butterfly

Assuming 24 of these butterflies operating in parallel at an estimated speed of 80 MHz, the maximum throughput is 1.9 billion butterflies per second - requiring a throughput into the butterflies of 30 GB/second. Total throughput, considering input and output to memories, would be 60 GB/second. This greatly exceeds the I/O capabilities of the FPGA; therefore, an algorithmic enhancement is used to reduce the I/O throughput. The Four-Step FFT [8] converts a one-dimensional FFT into a two-dimensional problem. A length N FFT can be arranged in a matrix form with the product of the matrix dimensions equal to N. FFTs are then performed on each row and, after a matrix multiplication, on each column. This technique reduces a million-point FFT into one thousand identical thousand-point FFTs. The twiddle factors for these FFTs can completely fit inside the embedded Block RAMs within the FPGA. Furthermore, all of the intermediate values for these smaller FFTs can reside inside the dedicated RAM structures present in the FPGA.

To simplify the hardware design the required two million-point FFT is converted into 512 FFTs, each of 4,096-points. Using the Four-Step FFT in this manner reduces the I/O requirements to a manageable 6.8 GB/sec. Four DDR SDRAMs can be connected to the device for a maximum memory throughput of 12.8 GB per sec, more than the required amount.

While the FPGA can hold 26 of these butterflies, to simplify I/O and control, only 24 of the butterflies are implemented with twelve in each of two 8-point FFTs. Figure 2, below, details the complete design. Two, 8-point FFT engines are implemented along with internal caches created from Block RAMs embedded in the FPGA. Each input to the FFT engines has a dedicated 512-word dual-port Reorder RAM. To compute the longest FFT used in the algorithm, a 4,096-point FFT, each input to the FFT computational engines must process 256 words per stage (as 256 words/input * 8 inputs/engine * 2 engines = 4,096 words). Therefore, 256 words in each input's Reorder RAM are used to store the output of the current stage's computation while the other 256 words supply the input words used in the current stage. This cache structure is duplicated to allow memory writes and reads to occur in parallel with computations. The previously completed FFT's results, occupying 256 words in each input's Reorder RAM, are written to memory while the next FFT's input data is read from memory. Upon completion of an FFT, these caches are swapped, presenting each FFT input with fresh data.

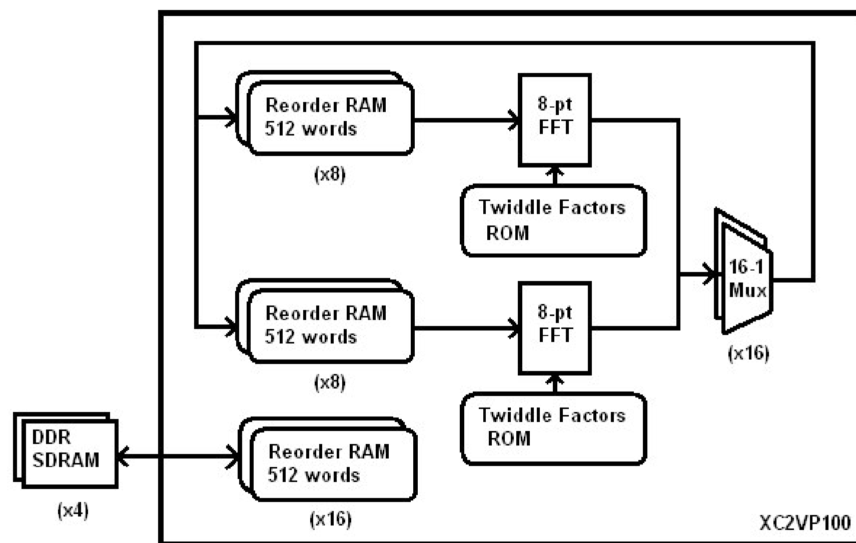

Figure 2. Hardware Design

IV. Performance Comparison

GIMPS maintains comprehensive performance benchmarks for a variety of processors [4]. These benchmarks are based on the amount of time needed for a processor to perform one squaring iteration of the Lucas-Lehmer test. A 3.4 GHz P4 can complete an iteration of a 40 million bit number in 60 milliseconds. As a complete Lucas-Lehmer test requires 40 million iterations, a Pentium can completely test a number for primality in roughly 28 days.

Initial implementation results for the computational engine indicate that the completed design can be run at an 80 MHz frequency in an XC2VP100-6ff1696. These timing numbers were obtained using the Xilinx ISE version 6.2i tools. The computational engine and associated RAM structures alone utilizes 70% of the soft logic and 86% of the dedicated multipliers in this device.

Performance estimates, shown in Table 2, include all of the steps needed to complete an iteration: creation of the weighted sequence, forward FFT, element-wise squaring, inverse FFT, weight removal, carry release, and final modular reduction. It should be noted that only the FFT steps were implemented. As expected, the majority of time is spent performing forward and inverse FFTs.

Table 2. Estimated Iteration Time Budget

Iteration Stage	Time (us)
Weighted sequence creation	250
Forward FFT	11,500
DFT coefficient squaring	250
Inverse FFT	11,500
Weight removal	250
Carry releasing	5,000
Mersenne mod reduction	5,000

An unexpected result from the iteration analysis was the large percentage of time (almost 30% of the total) spent performing the serial operations of carry releasing and Mersenne modular reduction. These activities are essentially equivalent to addition of two million-digit numbers and must be performed serially. The serial steps, computed at the FPGA's slower clock frequency, partially offset the performance gains of this implementation.

The design's estimated final iteration time of 34 milliseconds is 1.76 times faster than a Pentium 4. The complete design featuring a Virtex-II Pro 100 unfortunately would cost roughly ten times a Pentium's cost. While cluster implementations of the GIMPS algorithm exist that could be spread across multiple Pentiums in an effort match the hardware implementation's performance, they are not as efficient as the serial version. However, it is quite likely that a cluster of ten Pentiums would noticeably exceed the throughput of this design. Furthermore, the latency required for testing a given Mersenne number is not as important as the overall throughput of the GIMPS distributed computing effort. Thus ten individual Pentiums, operating on separate numbers, would exceed the throughput of this design by approximately six times for roughly the same cost.

While the cost comparison is perhaps unfair to the FPGA as the Pentium benefits from economies of scale not found in the programmable logic industry, the die size of the Virtex-II Pro 100, loosely extrapolated from the actual die size of a Virtex-II Pro 20 [9], is roughly four times that of a Pentium 4 implemented in a comparable 130 nm process. Thus, while this design produces a noticeable speed-up, the cost of this speed-up is difficult to justify.

V. Conclusion and Future Work

In conclusion, a stand-alone hardware implementation of the GIMPS algorithm has been designed capable of squaring 12-million digit numbers in under 34 milliseconds, a 1.76 times performance improvement compared to a fast Pentium. This design, with few modifications, can be used to multiply arbitrary large integers or convolve large sequences. While some avenues of improvement exist, the cost of this solution, in terms of dollars or silicon area, outweighs the benefit. However, when used as a benchmark the results indicate that FPGAs can now compete with Pentiums on a performance basis in applications that benefit from high-speed floating-point multiplication.

Avenues for improvement include algorithmic advances and design floor planning. Additionally, the Virginia Tech Configurable Computing Laboratory is in the process of constructing a cluster of FPGA boards connected via gigabit links. These boards feature the Virtex-II Pro 30, which are significantly less expensive than the Virtex-II Pro 100 device used in this project. Implementation of a parallelized GIMPS algorithm on this cluster should produce more attractive results.

Acknowledgements
The authors would like to thank Nick Craig-Wood for his significant contributions to the mathematical aspects of this paper. Without his assistance it is likely that no performance gain would have been achieved. Additionally, Semiconductor Insights (http://www.semiconductor.com) was gracious enough to supply die size information for Xilinx and Intel chips, allowing a more thorough comparison.

References

[1] GIMPS, “The Great Internet Mersenne Prime Search”, http://www.mersenne.org.
[2] E. Weisstein. "Mersenne Prime." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/MersennePrime.html.
[3] R. Crandall and Fagin, B. “Discrete Weighted Transforms and Large-Integer Arithmetic,” Math. Comp. 62 (1994), 305-324.
[4] GIMPS, “CPU Benchmarks,” http://www.mersenne.org/bench.htm.
[5] N. Craig-Wood, “Integer DWTs mod 2^64-2^32+1,” http://www.craig-wood.com/nick/armprime/math.html.
[6] N. Craig-Wood, personal correspondence.
[7] S. Winograd, "On Computing the Discrete Fourier Transform", Math. Comp., 32, pp.175-199, 1978.
[8] W. Gentleman, and Sande, G., ``Fast Fourier Transforms - For Fun and Profit", AFIPS Proceedings, vol. 29 (1966), p. 563 - 578.
[9] Semiconductor Insights, private correspondence, http://www.semiconductor.com.