An update on Scalable Implementation of Primitives for Homomorphic EncRyption – FPGA implementation using Simulink

David Bruce Cousins, Kurt Rohloff, Chris Peikert, Rick Schantz
Raytheon BBN Technologies, Georgia Institute of Technology
{dcousins, krohloff, schantz}@bbn.com cpeikert@cc.gatech.edu

Abstract

Accellerating the development of a practical Fully Homomorphic Encryption (FHE) scheme is the goal of the DARPA PROCEED program. For the past year, this program has had as its focus the acceleration of various aspects of the FHE concept toward practical implementation and use. FHE would be a game-changing technology to enable secure, general computation on encrypted data, e.g., on untrusted off-site hardware. However, FHE will still require several orders of magnitude improvement in computation before it will be practical for widespread use.

Recent theoretical breakthroughs demonstrated the existence of FHE schemes [1, 2], and to date much progress has been made in both algorithmic and implementation improvements. Specifically our contribution to the Proceed program has been the development of FPGA based hardware primitives to accelerate the computation on encrypted data using FHE based on lattice techniques [3]. Our project, SIPHER, has been using a state of the art tool-chain developed by Mathworks to implement VHDL code for FPGA circuits directly from Simulink models. Our baseline Homomorphic Encryption prototypes are developed directly in Matlab using the fixed point toolbox to perform the required integer arithmetic. Constant improvements in algorithms require us to be able to quickly implement them in a high level language such as Matlab. We reported on our initial results at HPEC 2011 [4]. In the past year, increases in algorithm complexity have introduced several new design requirements for our FPGA implementation. This report presents new Simulink primitives that had to be developed to deal with these new requirements.

A review of Fully and Somewhat Homomorphic Encryption

Fully Homomorphic Encryption (FHE) holds the promise to securely run arbitrary computations over encrypted data on untrusted computation hosts [2]. The general FHE concept of operations is that sensitive data is encrypted with a public key, then sent to an untrusted computation host, which can perform arbitrary computations on the encrypted data without first needing to decrypt it. It has been shown to be theoretically possible to evaluate arbitrary programs using just two special purpose FHE operations, EvalAdd and EvalMult, which at the simplest level, roughly correspond to bitwise XOR and AND gates operating on encrypted bits. A sequence of these operations is run against the encrypted data, resulting in an encryption of the output of the original program run on the unencrypted data. This encrypted result can then be sent back to the original client, who decrypts the result using its secret key. The encrypted data is protected at all times with reasonable security guarantees based on computational hardness results.

A 'Fully' Homomorphic Encryption scheme allows and unlimited number of these Eval operations to be performed. All known FHE schemes are based on computationally hard stochastic lattice theory problems, which add some noise with each operation and require a very computationally expensive "recryption" operation that is periodically run on intermediate ciphertexts to keep the noise at a level that still permits decryption. A 'Somewhat' Homomorphic scheme, on the other hand, supports several (but not unlimited) EvalMult and EvalAdd operations while preserving the correctness of decryption. In other words, SHE can schemes support secure computation for only a small subset of programs. By focusing on an SHE scheme, we can direct our research towards the implementation of efficient hardware primitives, while the FHE community develops more efficient recryption algorithms.

Recent Developments in the SIPHER SHE Scheme

Our current SHE scheme relies on operations that are generally inefficient to implement on standard CPU architectures (i.e. modular arithmetic with a large modulus). The EvalAdd and EvalMult operations for example are element wise vector adds and multiplies taken modulo some particular prime integer q. These are trivial to express using Matlab: c = mod(a+b, q) and c = mod(a.*b, q).

For convenience most of the previously published SHE and FHE implementations have used standard tools such as the GNU Multiple Precision Arithmetic Library (GMP) [5], which enable researchers to code operations using very large integers. This limits their focus to operations on CPUs and does not allow them to take advantage of specialized parallel computation hardware like FPGAs which provide highly cost-effective parallelism. Our approach to developing the FPGA code for implementing EvalAdd and EvalMult is to develop arithmetic circuits that will achieve high throughput by using parallelism and pipelining on the FPGA.

We initially develop prototype descriptions in Matlab that we re-implement in a stream-oriented hardware implementable manner in Simulink. The results of the implementations are compared to verify correctness. A conversion from Simulink to VHDL is done in a completely automated fashion using Mathwork's HDL coder. This tool chain provides us the means to develop our primitives, including cyclic VHDL based FPGA prototyping, much

Sponsored by Air Force Research Laboratory (AFRL) Contract No. FA8750-11-C-0098. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement "A"

faster than traditional methods. Some examples of efficiency are:

1. The Matlab and Simulink Models are driven with the same fixed point data variables, and generate the same format output, simplifying test and comparison
2. The bit width of the circuits is specified at compile time by specifying the bit width of the input data. The sizing of intermediate mathematical operations is done automatically by the fixed point toolbox. Thus many of the same models can be used for 8 bit or 64 bit inputs.
3. The resulting VHDL is vendor independent. This allows for rapid benchmarking on multiple architectures. However, hand optimization of VHDL may be required for optimum performance in order to take advantage of vendor specific IP.

Implementing fast modulo add, subtract and multiply in Simulink for HDL generation

Software implementations of modulus usually use some form of trial division to determine the remainder operation. Implementing modulus integers with large numbers of bits in an efficient manner requires the use of special numerical algorithms that have been developed, such as the Montgomery Reduction [6]. These algorithms avoid division by q, but rather scale the integers so that many of the divisions can be performed by a power of 2, requiring only simple bit shifts. Our SHE requires circuits for fast modulo addition and multiplication (to directly implement the EvalAdd and EvalMult mentioned above). In addition, our scheme relies heavily on the Chinese Remainder Transform (CRT), which can be implemented as an EvalMult, followed by an FFT [7] that uses modulo integer instead of complex arithmetic. The implementation of the FFT requires us to perform a standard radix 2 'Butterfly' operation, which uses one addition, one subtraction and one multiply, all modulo q. Thus we need to implement a modulo subtraction as well as addition.

Initially, our selection of lattice based HE led to looking at relatively modest sized modulus, on the order of twenty bits. An implementation of Montgomery Reduction based arithmetic would be relatively efficient, requiring hardware multipliers on the order of 40 bits. However, later research showed that for any reasonable security requirements our SHE scheme would need O(64) bits for our modulus. Our implementation of Montgomery arithmetic in Simulink required us to double our bit width to represent intermediate values represented in Montgomery form. We found that there is an intrinsic limitation of 128 bit width in Simulink even when using the fixed point toolbox. This meant that we could not compile our multipliers for bit widths on the order of 64 bits.

Additionally, our early arithmetic models were all designed for a single value of modulus q to be used for all operations. During the development of our SHE scheme we found that using multiple values of related moduli resulted in more efficient implementations. Thus our circuits would need to operate with multiple (but not unlimited) values of q. As a response to this we eliminated Montgomery arithmetic and take a simpler approach to modulo addition and subtraction.

Figure 1 shows the Matlab code and resulting Simulink block for performing a streaming EvalAdd when the inputs are constrained to be less than a given modulus q. The model can operate on one pair of inputs every clock cycle. The model shown does not have any additional pipeline registers for simplicity, but they can be added to the model in order to increase the maximum clock speed of the resulting VHDL, at a cost of additional pipeline stages. In our applications we expect to process streams of input on the order of several thousand entries, so this additional pipeline latency is trivial.

Figure 2 shows the Matlab and resulting Simlink block for modulo subtraction. The same comments about pipelining the circuit apply.

Modulo multiplication is a much more complicated operation, even if the input multiplier and multiplicand are bounded by q. Furthermore, we determined in our earlier work that the VHDL code generated by Simulink for large multiplications is not automatically pipelined, so the resulting multiplies severely restrict the resulting clock rates of the circuits. To address these two constraints, we adopted a recently developed interleaved modular multiplication based on a generalized Barrett reduction [8]. This multiplier has the following properties:

1) Long words of bit length L can be represented by n

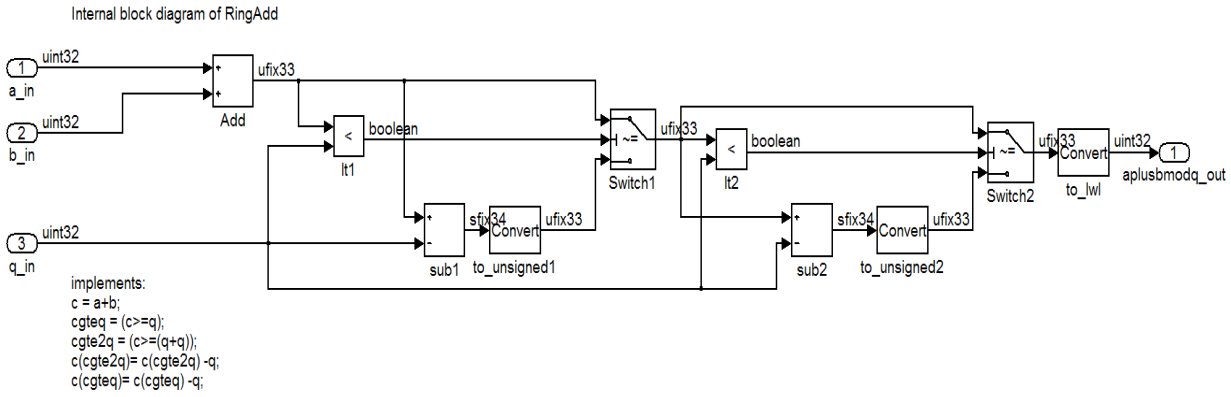

Figure 1: Internal Structure of Simulink HDL ready Modulo Add primitive.

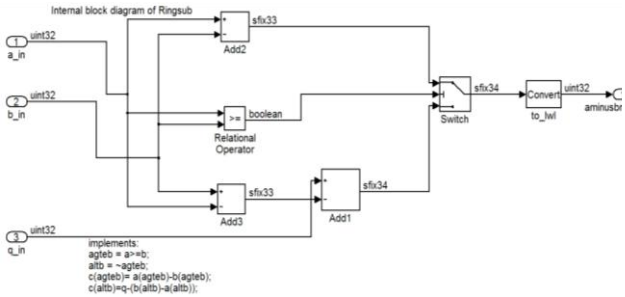

Figure 2: Internal Structure of Simulink HDL ready Modulo Subtract primitive.

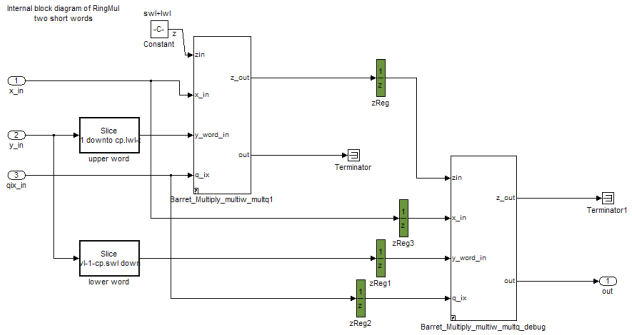

Figure 3: Top level structure of Simulink HDL ready two stage Barrett Modulo Multiply primitive.

smaller words of bit length S (i.e. four 16 bit words to represent a 64 bit modulus).

2) The multiplication is performed in n stages, where each stage performs one modulo multiplication that is L+S bits long. The stage can be pipelined to perform one modulo multiply per clock cycle.

3) Each stage has a Barrett modulus performed on the partial product, which reduces overall bit growth of the partial products to L+S. Each stage requires 3 multiplies, and all divisions required by the Barrett algorithm are implemented as simple bit shifts.

4) One circuit can support multiple moduli. All parameters that are specific to a given modulus can be stored in a table and indexed.

Figure 3 shows the structure of our resulting multiplier for a two stage operation (i.e. L = 2S). Figure 4 shows the model for a single stage in the pipeline. All stages use the same model. Again, internal pipelining in the stage is not shown.

Implementing fast CRT in Simulink for HDL generation

As mentioned earlier, our scheme uses the CRT, which relies heavily on modulo arithmetic. We have developed a Simulink model for performing a fast CRT, based on the primitives discussed above. We implemented one of the standard pipeline decimation in frequency FFT architectures, known as the Radix 2, Multiplath Delay Commutator [7]. The fundamental structure of the model is identical for a complex version that computes the standard FFT, and the modulo arithmetic version that computes the FFT portion of our CRT. The only difference is in the Simulink Model that implements the radix 2 butterfly.

Figure 3 shows the structure of this pipelined CRT. The design trades off area for processing speed. For an N point transform, log2(N) radix 2 Butterflies are required (though the last butterfly does not require multiplies). Additionally, 3/2N-2 delay elements are required. The data needs to be presented to the circuit in two parallel streams, and the resulting output is in bit reverse order.

We are currently in the process of analyzing the performance of this circuit, and determining the size CRT operation that can be fit into our candidate FPGA architecture. Our analysis has shown that for high security applications we may need to perform CRT operations on vectors of up to 2^{14} in length. For such large vector sizes, an alternative design approach may be necessary in order to fit the circuit within the FGPA.

Interim Results

Our presentation will include examples of our primitives coded in Matlab and Simulink and examples of VHDL code generated by the HDL coder. We will also be able to show timing results from Modelsim based simulations of the resulting code., as well as actual timings using a Virtex 6 on the Xilinx ML605 evaluation board

References

[1] C. Gentry and S. Halevi. Implementing Gentry's Fully-Homomorphic encryption scheme. In Kenneth Paterson, editor, Advances in Cryptology – EUROCRYPT 2011, volume 6632 of Lecture Notes in Computer Science, chapter 9, pages 129–148. Springer, 2011.

[2] D. Micciancio. A first glimpse of cryptography's Holy Grail. Comm. ACM 53, 3 (March 2010), 96-96.

[3] V. Lyubashevsky, C. Peikert, and O. Regev. "On ideal lattices and learning with errors over rings". In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, *volume*

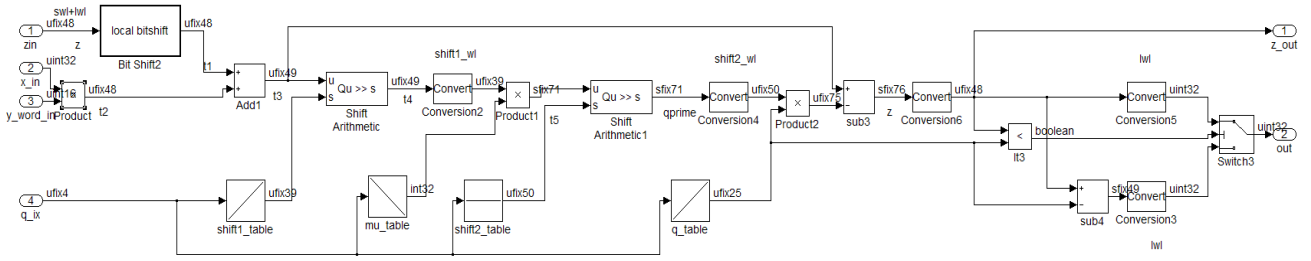

Figure 4: Internal structure of Barrett Modulo Multiply stage

1–23. Springer Berlin / Heidelberg, Berlin.

[4] D. Cousins, K. Rohloff, C. Peikert, R. Schantz "Scalable Implementation of Primitives for Homomorphic EncRyption – FPGA implementation using Simulink" 2011 High Performance Extreme Computing Workshop Sep 21-22 2011, Lexington MA

[5] http://gmplib.org/ last accessed May 14, 2012.

[6] P. L. Montgomery "Modular Multiplication Without Trial Division", *Mathematics of Computation* Vol. 44, No. 170 (Apr., 1985), pp. 519-521, American Mathematical Society.

[7] L. R. Rabiner and B. Gold. *Theory and Application of Digital Signal Processing.* Prentice-Hall, Inc., 1975.

[8] M. Knezevic, F. Vercauteren, and I. Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods", *IEEE Transactions on Computers*, Vol. 59, No. 12, Dec 2010.

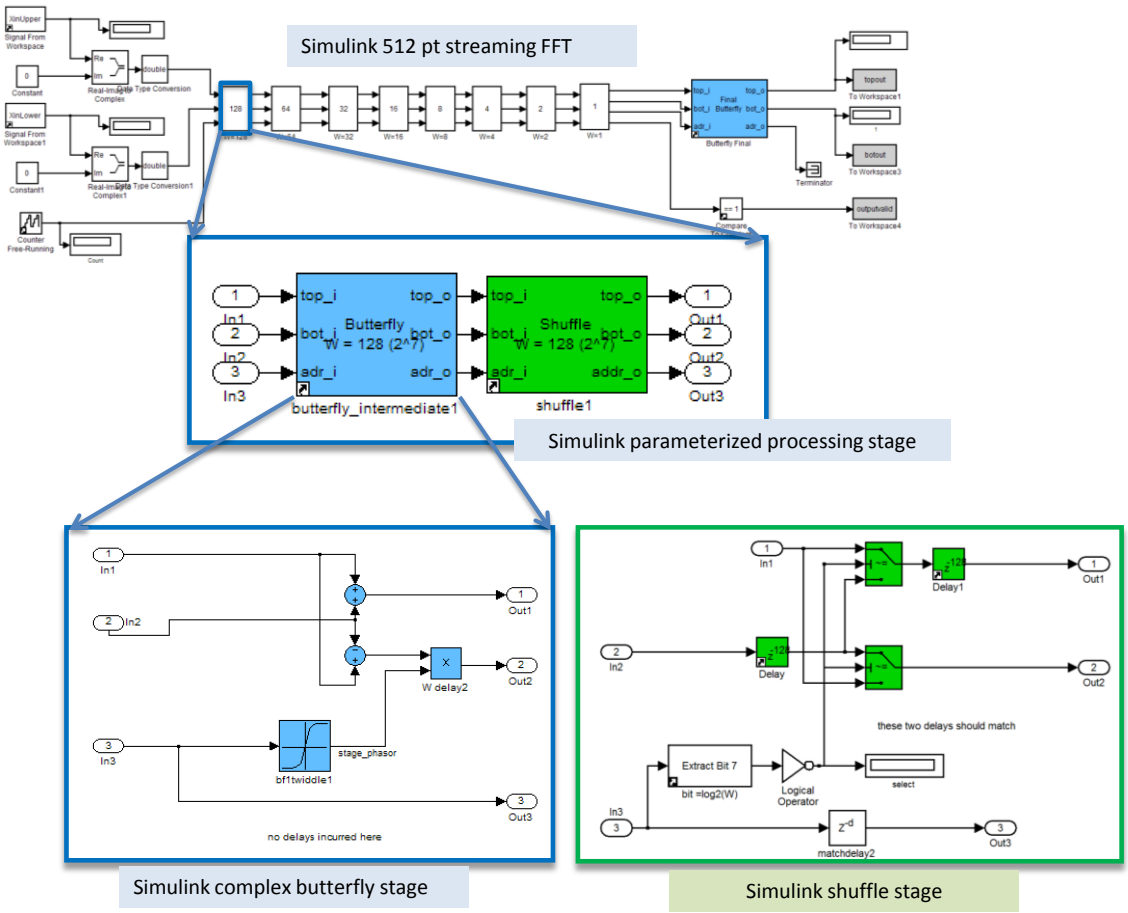

Figure 5: Simulink Pipeline FFT Structure