CYCLE-EFFICIENT LINEARY FEEDBACK SHIFT REGISTER IMPLEMENTATION ON WORD-BASED MICRO-ARCHITECTURE

Jui-Chieh Lin[1], *Sao-Jie Chen*[2], *and Yu Hen Hu*[1]

[1]University of Wisconsin, Dept. Electrical and Computer Engineering, Madison, WI 53706 USA
[2]Graduate Institute of Electronics Engineering, National Taiwan University

ABSTRACT

A novel algorithm transformation method, called term-preserving look-ahead transformation (TePLAT) is proposed to transform the bit-serial linear feedback shift register (LFSR) algorithm into a bit-parallel formulation which promises order of magnitudes improvement of execution speed compared to the traditional look-ahead algorithm transformation approach. TePLAT is applied to 26 commonly used LFSRs and tested on two popular word-based micro-processor development platforms: a Texas Instrument C6416 Code Composition Simulator and an ARM-9 Simulator. In all 26 cases, TePLAT transformed implementations consistently deliver much higher throughput than those implementations based on traditional look-ahead algorithm transformation.

Index Terms— Linear feedback shift register, iteration bound, vector processing, look ahead transformation, scrambler

1. INTRODUCTION

The linear feedback shift register (LFSR) algorithms have found wide applications in wireless communication, including scrambling, error correction coding, encryption, testing, and random number generation [4], [5], [6], [7]. Hence, increasing throughput of the LFSR implementation will have profound impact on accelerating the overall execution speed of many embedded applications, in particular, software defined radio (SDR) [1], [3].

In this work, cycle-efficient implementation of a total of 22 LFSRs [19], [20], [21] on a word-based micro-architecture platform is considered. The LFSR's output is determined solely by its initial condition without any external input. It is commonly specified by a generator polynomial over the Galois Field GF(2). The generator polynomial leads to a bit-serial algorithm formulation of the LFSR algorithm where each variable is a binary bit. An LFSR can also be modeled as a finite state machine with no external input.

Since LFSR is a bit-serial algorithm, it is inefficient implementing on a word based micro-architecture. To address this problem, one approach would be to implement a LFSR using special purpose hardware [2], [8], [9] which may interface with the host micro-processor via instruction set extensions or interrupt. However, dedicated hardware module is less flexible for applications such as a SDR.

A second approach seeks to reformulate the LFSR algorithm so that inherent bit-level parallelism afforded by a word-based micro-architecture may be fully exploited [11], [12], [13]. Since a word may be regarded as a vector of binary bits, traditional vectorized compilation techniques such as loop-unrolling [10] becomes a natural choice for this purpose. However, the recursive formulation of the LFSR algorithm imposes a fundamental limitation, known as the iteration bound [10] on the amount of parallelism that may be exploited using loop unrolling.

Fortunately, a look-ahead transformation (LAT) [10] promises to resolve this difficulty. By substituting the recursive expression of the next iteration into the present iteration, LAT yields new recursion formula that often has a smaller iteration bound. However, LAT comes with a side effect: It often introduces additional operations. In terms of LFSR, this implies the LAT-transformed LFSR formulation may contain many more terms [14] than the original LFSR. Since each term will require additional instruction to execute, the potential benefit of execution cycle reduction due to LAT and unrolling may often be compromised by these extra operations.

In [15], it has been shown that if the generator polynomial has the form of $X[n] = X[n-a] \oplus X[n-b]$, then a special look-ahead transformation may be applied that preserve the number of terms in the transformed formula and hence will not cause any overhead due to LAT. In this work, this preliminary result is generalized to *arbitrary* generator polynomials and is called the term-preserving look-ahead transformation (TePLAT). As promised by its name, when TePLAT is applied to a given LFSR generator polynomial, it is guaranteed that number of terms of the transformed generator polynomial will remain unchanged. This term-preserving property makes it feasible to apply TePLAT aggressively to achieve maximum throughput rate with respect to a particular micro-architecture.

To evaluate the performance of the TePLAT algorithm, 25 LFSRs appeared in state of art communication standards have been implemented in two popular microprocessor development platforms: (i) Texas Instruments C6416 Code Composition Simulator [16] and (ii) the ARM-9 Simulator [17]. It is pleasantly confirmed that the speed up factors of TePLAT over existing methods range from 1.5 to 18 depending on the structure of the generator polynomials.

2. PRELIMINARIES

2.1 LFSR and Generator Polynomials

A LFSR can be specified by its *generator polynomial* over a Galois field GF(2):

$$P(x) = 1 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_n x^n \quad (1)$$

where "+" represents exclusive-or (XOR) operations. The length of the LFSR, n, is generally the degree of the generator polynomial. A LFSR output a periodic, pseudo-random sequence of length $2^n - 1$ when $P(x)$ is a *primitive polynomial*. A LFSR corresponds to

a *n*-state finite state machine (FSM) whose state diagram has two disjoint parts: one consists of a single state of all 0s, and the other is a ring of remaining states. The output of the LFSR is represented by the states in this ring.

2.2 Loop-Unrolling

Loop-unrolling (*a.k.a loop unfolding*) [10] consolidates loop bodies of consecutive iterations into a single iteration to expose inherent parallelism. However, loop unrolling cannot achieve arbitrary level of parallelism. An inter-iteration *data dependence* imposes an upper bound on how many times a loop can be unrolled to explore the inherent inter-iteration parallelism. Theoretically, this kind of inter-iteration dependence relation is characterized by a notion called *iteration bound* [10].

In [11] and [15], loop unrolling is applied to achieve 3-fold speed-up for the scrambler in Wifi. Due to the iteration bound, no further loop unrolling is applied. The iteration bound is also named as *first-block* and *truncated window* in [12] and [13] respectively.

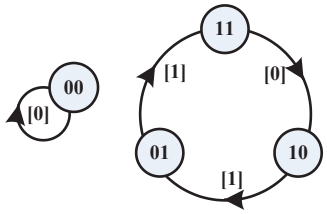

Fig. 1. State diagram of a LFSR in example 1

2.3 Look-Ahead Transformations

The iteration bound can be reduced using a *look-ahead transformation* (LAT) [10]. LAT can be best illustrated with an example. Consider a general LFSR equation ($m_k < m_{k+1}$):

$$x[n] = x[n-m_1] \oplus x[n-m_2] \oplus \cdots \oplus x[n-m_M] \quad (2)$$

The iteration bound in this case is τ_{XOR}/m_1 where τ_{XOR} is the execution time (clock cycles) per XOR operation. Now consider the same equation at an earlier iteration:

$$x[n-m_1] = x[n-2m_1] \oplus x[n-m_1-m_2] \oplus \ldots \oplus x[n-m_1-m_M] \quad (3)$$

Substituting (3) into (2), one has

$$x[n] = \{x[n-2m_1] \oplus x[n-m_2] \oplus \ldots \oplus x[n-m_M]\} \oplus \{x[n-m_1-m_2] \oplus \ldots \oplus x[n-m_1-m_M]\} \quad (4)$$

In (4), the new iteration bound is

$$\tau_{XOR}/\min.\{2m_1, m_2\} < \tau_{XOR}/m_1.$$

Thus, more unrolling may be applied to exploit more bit-level parallelism to speed-up execution.

However, LAT comes at a cost. Referring to (4), in general, it may contain more terms than the original recursion in (2). As such, although more bits may be processed per instruction, there will be more instructions to be executed due to the LAT induced overhead. Take scrambler for Wimax as an example, the original generator polynomial is $1+x^{14}+x^{15}$ and the LAT generator polynomial is $1+x^{15}+x^{28}+x^{29}$. The vectorized, unrolled implementation is:

X[n:n+13] = X[n–14:n-1] ^ X[n–15:n-2];

and the LAT version is:

X[n:n+14] = X[n–15:n-1] ^ X[n–28:n-14] ^ X[n–29:n-15];

The length of parallel bit vector increases negligibly from 14-bit to 15-bit and cannot compensate the cost of one additional term after applying LAT. The throughput reduced by 10% as shown in Table 1 and Table 2. Hence blindly applying LAT does not necessarily contribute to improving throughput rate of implementing the LFSR algorithm.

3. TERM-PRESERVING LOOK-AHEAD TRANSFORMATION

In previous section, it is shown that while LAT promises to reduce iteration bound, it also introduces computation overhead that threatens to nullify any potential performance gain. In this section, an alternative look-ahead transformation for the LFSR algorithm will be developed.

Property 1. *Term Preserving Property* – Denote

$$Q(x) = \sum_{m=0}^{2K} q_m x^m = \left[P(x)\right]^2 = \sum_{k=0}^{K} p_k x^{2k} \quad (5)$$

then

$$q_m = \begin{cases} p_k & m = 2k; \\ 0 & \text{otherwise.} \end{cases} \quad (6)$$

In other words, although Q(x) is a polynomial of twice the order of P(x), both of them have the same number of terms.

Definition 1. *Term-Preserving Look-Ahead Transform* (TePLAT) – A TePLAT of a LFSR with a generator polynomial P(x) is a LFSR with a generator polynomial $Q(x) = [P(x)]^2$.

Example 1. Consider the LFSR with recursive equation:

$$X[n] = X[n-1] \oplus X[n-2]. \quad (7)$$

After applying the TePLAT once, the transformed recursive equation becomes:

$$X[n] = X[n-2] \oplus X[n-4]. \quad (8)$$

These two LFSRs are depicted in Fig. 2. The original iteration bound in (7) is τ_{XOR}, and that of the transformed LFSR in (8) is $\tau_{XOR}/2$. Hence, the throughput may be doubled after applying loop unrolling to eq. (8).

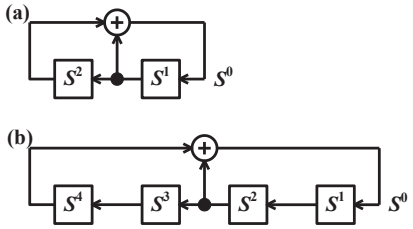

Fig. 2. (a) Original LFSR. (b) LFSR after TePLAT.

3.1 Equivalence of Transformed LFSR

A LFSR generates a periodic binary sequence. Each period of this sequence, denoted by $S(P(x))$ is dictated by the corresponding generator polynomial $P(x)$. Traditional LAT can be interpreted as converting the LFSR with a generator polynomial $P(x)$ into a new LFSR with another generator polynomial

$$Q(x) = P(x) \cdot (1+x^{k^*}) \quad (9)$$

where $k^* = min.\ \{k;\ 1 \le k \le K,\ p_k = 1\}$. Then, according to a theorem presented in [18], the following property holds:

Property 2. **(Theorem 6.53 in [18])** *Let $S(P(x))$, and $S(Q(x))$ be the linear recurrent sequences generated using generator polynomials $P(x)$, and $Q(x)$ respectively, and $g(x)$ is another polynomial in GF(2), then*

$$S(P(x)) \subset S(Q(x)) \text{ if and only if } Q(x) = P(x) \cdot g(x) \quad (9)$$

In other words, the binary sequence of the original LFSR is a subset of all the binary sequences that may be generated by a LAT transformed LFSR.

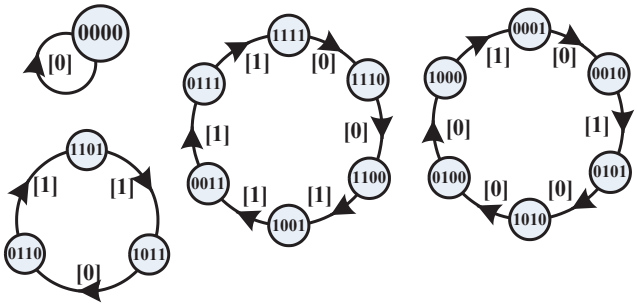

Fig. 3. State diagram for degree-1 TePLAT LFSR

To illustrate, consider the state diagram of the LFSR specified in (7) as shown in Fig. 1. It contains two independent cycles. The maximal length sequence that may be generated by this LFSR is 011 which may be obtained if the LFSR is initialized to 01, 10, or 11. Otherwise, the LFSR will be trapped into the unused state of 00.

The state diagram of the TePLAT transformed LFSR is shown in Fig. 3. Note that there are now four independent cycles in this state diagram. Moreover, the set of binary sequences of the original LFSR, {0} and {011}, are a proper subset of the binary sequences generated by the TePLAT transformed LFSR, namely, {0}, {011}, {001111}, and {010001}. The sequence sets in Fig. 3 can be obtained by exhaust all possibilities of initial vectors {0000} to {1111}. Interested reader may refer to Example 6.18 and Theorem 6.63 in [18] for the method and the characteristics of TePLAT sequence sets. Note these diagrams are only for illustration; real implementation does not require exhausting the states.

In order for the TePLAT transformed LFSR to generate an identical binary sequence as the original LFSR, the initial state must be set to one of the three states {1101, 1011, 0110}. Note that these three feasible initial states are direct concatenation of successive output bits of the original LFSR. This example further points to a method to initialize the TePLAT transformed LFSR to ensure it to generate an identical binary sequence as original LFSR.

3.2 Complexity Analysis and Execution Overhead

We categorize the coeds into three parts: *data alignment*, *iteration overhead* and *arithmetic operations*. Fig. 4 shows the histogram of performing the LFSR—$P(x) = 1 + x^3 + x^{16}$ on TI-C6416. The horizontal axis represents the logarithm of look-ahead factor $\log_2 F$ such that $Q(x) = [P(x)]^F$. The vertical axis depicts the number of cycle required for generating 768 bits. The curve appears to be an inverse power of look-ahead factor for $F < 2^4$. The cycle number drops below the inverse power curve and reaches the bottom when both terms in $P(x) = 1 + x^3 + x^{16}$ align at word boundaries for $F=2^5$. However, the performance enhancement is compensated due to the longer chains induced by TePLAT.

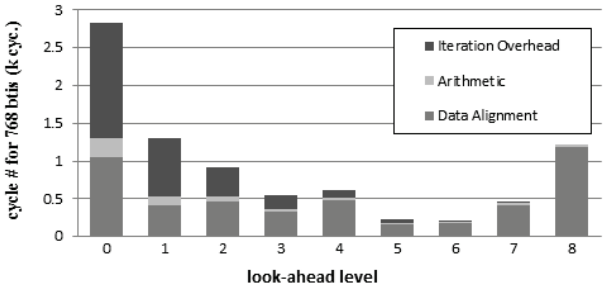

Fig. 4. TePLAT of LFSR:$P(x) = 1 + x^3 + x^{16}$ cycle histogram.

5 SIMULATION RESULTS

5.1 Experimental Setup

We believe that the cycle-accurate simulator can profile convincible outcome for demonstrating our algorithm. Therefore, we adopt Texas Instruments© Inc. Code Composer Studio (CCS) and Advanced RISC Machines© Ltd. Instruction Set Simulator (ARMulator). In this work, we build an in-house source-to-source compiler that generates LFSR codes with TePLAT factors ranging from 2^0 to 2^8. We then run the generated codes on the corresponding simulators and determine the best TePLAT factor for the LFSR. We call the procedure *exhaustive method*, and the best performance look-ahead transformed LFSR found based on the experiments is termed as "*best*" in the following results.

5.2 Throughput Performance Evaluation

We chose two popular and representative processors for mobile devices, TI-C6416 digital signal processor and ARM-926 general purpose processor. We put the results into two separate tables, where significant improvement can be observed. Comparisons of the optimization techniques are provided in Figs. 5 and 6.

Throughput numbers are given for all the LFSRs. The conventional LFSRs are similar to [12][13] that applied loop-unrolling (LU) technique. The *best* look-ahead transformed LFSR's improvement depends on the LFSR generator polynomial and the processor architecture. Our experimental results show that the best LFSR can usually be found by TePLAT factor ranging from 2^0 to 2^8. The *best* look-ahead transformed LFSRs can perform at most 18X to 50% faster. A star (*) is used to indicate the

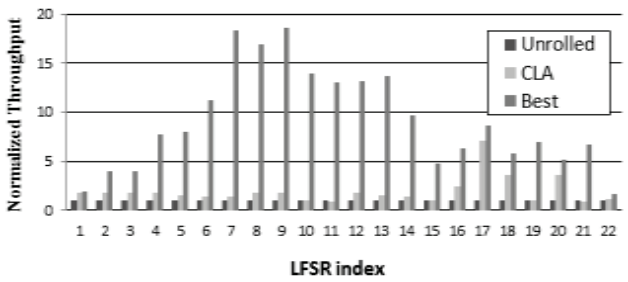

Fig. 5. Comparison of algorithm on TI C6416 architecture (normalized to unrolled version).

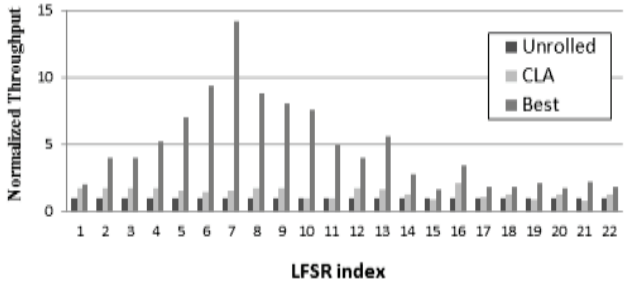

Fig. 6. Comparison of algorithm on ARM926 architecture (normalized to loop-unrolled version).

maximum-level TePLAT is applied. In [8], the bit manipulation unit (BMU) hardware was proposed and implemented on XLINX VirtexII. The throughput of Wifi scrambler in [8] was 0.6 bit/cycle. However, our work can achieve 0.7 bit/cycle on ARM and 2.9bit/cycle on TI. The authors also note that some LFSRs were designed for efficient implementation such as Grain stream cipher [23]. Our TePLAT method has negligible improvement with such LFSR.

6 CONCLUSION AND FUTURE WORKS

In this paper, we proposed a term preserving look-ahead transformation (TePLAT) to achieve bit-level parallelism in LFSR. Compared to the conventional look-ahead transformation (LAT), this method does not introduce additional terms in the formula and therefore owns the same complexity as the original LFSR formula. The method is used to implement arbitrary-input LFSR and scrambler on word–based processors. The correctness on the implemented LFSR with arbitrary number of input and its look-ahead factor is proved. The paper also provided abundant simulations results on 25 popular LFSRs and scramblers in wide-spread communication standards. This method adjusts parameter based on the platform and performs well on ARM and TI platforms, that is, this algorithm transformation method will be applicable to various platforms.

11. REFERENCES

[1] J. Mitola III, Cognitive Radio Architecture, John Wiley & Sons, Inc., 2006.

[2] J. Glossner, D. Iancu, J. Lu, E. Hokenek, and M. Moudgill, "A software-defined communications baseband design," in IEEE Communication Magazine, vol. 41, no. 1, pp. 120–128, Jan. 2003.

[3] H. Lee and T. Mudge, "A Dual-Processor Solution for the MAC Layer of a Software Defined Radio Terminal," in Proc. of the 2005 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASE 2005), no. 7, pp. 257-265, Sept. 2005.

[4] IEEE Std. 802.11–2007, Part 11: "Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," IEEE Std. 802.11, 2007.

[5] G. C. Ahlquist, M. Rice, and B. Nelson, "Error Control Coding in Software Radios: An FPGA Approach," in Proc. of the IEEE Personal Communications, vol. 6, no. 4, pp. 35-39, Aug. 1999.

[6] V. Sriram and D. Kearney, "An FPGA Implementation of a Parallelized MT19937 Uniform Random Number Generator," EURASIP Journal on Embedded Systems, vol. 2009, no. 7, pp. 1-6, 2009.

[7] K. K. Saluja and C.-F. See, "An Efficient Signature Computation Method," IEEE Design and Test of Computers, vol. 9, no. 4, pp. 22-26, Oct. 1992.

[8] S. H. Jeong, M. H. Sunwoo, and S. K. Oh, "Bit Manipulation Accelerator for Communication Systems Digital Signal Processor," in Journal on Applied Signal Processing (EURASIP), vol. 2005, no. 16, pp. 2655–2663, 2005.

[9] M. Wei, M. Snir, J. Torrellas, and R. B. Tremaine, "A Near-Memory Processor for Vector, Streaming and Bit Manipulation Workloads," in Proc. of the Second Watson Conference on Interaction between Architecture, Circuits, and Compilers (P=AC2), Sept. 2005.

[10] K. K. Parhi, VLSI Digital Signal Processing Systems Design and Implementation, John Wiley and Sons, Inc., 1999.

[11] Y. Tang, L. Qian, and Y. Wang, "Optimized Software Implementation of a Full–Rate IEEE 802.11a Compliant Digital Baseband Transmitter on a Digital Signal Processor," in Proc. of the IEEE Global Communication (GLOBECOM 2005), vol. 4, pp. 2194–2198, Nov. 2005.

[12] S. Chowdhury and S. Maitra, "Efficient Software Implementation of Linear Feedback Shift Registers," in Proc. of the International Conference on Cryptology in India (INDOCRYPT 2001), Lecture Notes in Computer Science (LCNS), Springer-Verlag, vol. 2247, pp. 297-307, Dec. 2001.

[13] C. Lauradoux, "From Hardware to Software Synthesis of Linear Feedback Shift Registers," in Proc. of the 21st IEEE International Parallel and Distributed Processing Symp. (IPDPS 2007), pp. 453-460, Mar. 2007.

[14] S. Sriram and V. Sundararajan," Efficient Pseudo-Noise Sequence Generation for Spread-Spectrum Applications" Workshop on Signal Processing Systems (SIPS 2002), pp. 80-86, Oct. 2002.

[15] J. Lin, M. Fan-Chiang., M. Hsieh, S. Mao, S. Chen, and Y. Hu, "Cycle Efficient Scrambler Implementation for Software Defined Radio," in Proc. of the International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2010), pp. 1586-1589, Mar. 2010.

[16] Texas Instruments, "Code Composer Studio Development Tools v3.3 Getting Started Guide," SPRU509H, May 2008.

[17] ARM Limited, "Realview ARMulator ISS User Guide v1.4.3," ARM DUI 0207D, Mar. 2007.

[18] R. Lidl and H. Niederreiter, Introduction to Finite Fields and Their Applications, revised ed., Cambridge Univ. Press, 1994.

[19] R. S. Katti, X. Ruan, and H. Khattri, "Multiple-Output Low-Power Linear Feedback Shift Register Design," IEEE Transaction on Circuits and System I: Regular Papers, vol.53, no.7, pp. 1487-1495, Jul. 2006.

[20] IEEE Std. 802.16e-2005, Amendment to IEEE Standard for Local and Metropolitan Area Networks - Part 16: "Air Interface for Fixed Broadband Wireless Access Systems - Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands," IEEE Std. 8021.16, Feb. 2006.

[21] 3GPP TS 36.212 v8.4.0, "Multiplexing and Channel Coding," Sept. 2008.

[22] F. Didier and L. Yann, "Finding Low-Weight Polynomial Multiples Using Discrete Logarithm," IEEE International Symposium on Information Theory (ISIS 2007), pp. 1036-1040, Jun. 2007.

[23] M. Hell, T. Johansson and W. Meier, "Grain – A Stream Cipher for Constrained Environment," International Journal of Wireless and Mobile Computing, vol. 2, no. 1, pp. 86-93, May 2007.