LETTER

Pipelined Squarer for Unsigned Integers of Up to 12 Bits

Seongjin CHOI[†], ***Nonmember and*** **Hyeong-Cheol OH**[†a)], ***Member***

SUMMARY This paper proposes and analyzes a pipelining scheme for a hardware squarer that can square unsigned integers of up to 12 bits. Each stage is designed and adjusted such that stage delays are well balanced and that the critical path delay of the design does not exceed the reference value which is set up based on the analysis. The resultant design has the critical path delay of approximately 3.5 times a full-adder delay. In an implementation using an Intel Stratix V FPGA, the design operates at approximately 23% higher frequency than the comparable pipelined squarer provided in the Intel library.

key words: *squarer, high-speed pipelining*

1. Introduction

Squaring is a widely used operation in various applications. The L_2-norm requires squaring operations in its calculation and is a typical measure for optimization, identification, and clustering applications. Moreover, squaring is intensively used in various signal processing problems, such as subband vector quantization, image compression, and pattern recognition. Given the intensive usage of squaring and its application in portable devices, the optimized design of squarers has attracted the interest of many researchers [1]–[7].

Most optimization schemes exploit the *folding* and *merging* techniques, which use $x_i + x_i = 2x_i$ and $x_i + x_i x_{i-1} = 2x_i x_{i-1} + x_i \overline{x_{i-1}}$, respectively [2]. Figure 1 describes the use of the optimization techniques in a 4-bit unsigned integer squarer. Trials have been performed for further improvements of squarer performance. However, trials, such as in [3], that extend the techniques in Fig. 1 do not significantly improve the squarer performance, for the bit widths and the implementation conditions considered in this paper. The high-radix techniques [4], [5] or signed-digit techniques [6] can be beneficial but require extra circuits to recode operands or convert representations. The designs with reduced precision, such as in [2], are also interesting but are not considered in the present study.

For many applications, including multimedia and portable applications, square operations are performed on short words with 8 or 12 bits [7]. In this work, we investigate a pipelining scheme for a hardware squarer that can process a 12-bit squaring operation in every cycle with a latency of four cycles. Using our analysis results, we adjust the delay of each stage such that the stage delays of the resultant design are well balanced. The proposed squarer can calculate the square of an 8-bit unsigned integer with reduced latency by bypassing a pipeline stage.

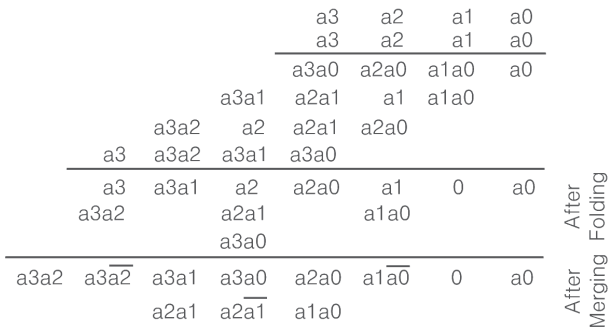

Fig. 1 Folding and merging techniques [2], for squaring a 4-bit unsigned integer.

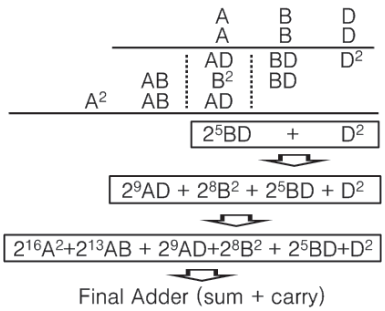

Fig. 2 Proposed pipelining scheme for a 12-bit squarer. *A*, *B*, and *D* are 4-bit unsigned integers.

Although this paper presents an FPGA implementation, the pipelined design proposed in this work is not restricted to FPGA designs. FPGAs are popularly adopted in various applications such as cloud servers and IoT platforms [8]; hence studies on FPGA-specific implementation, such as [9], are interesting. However, this study focuses on the design at the abstraction level of carry-save adder (CSA) trees [10] and relies on the technology mapper of the FPGA design tool, so that the design can be adopted in other implementation technologies.

2. Proposed Squarer

Figure 2 presents the proposed pipelining scheme for a 12-bit squarer. We aim to calculate the square of a 12-bit un-

Manuscript received October 14, 2017.
Manuscript revised November 26, 2017.
Manuscript publicized December 6, 2017.

[†]The authors are with College of Sci. & Tech., Korea Univ., Sejong 30019, Korea.
a) E-mail: ohyeong@korea.ac.kr

DOI: 10.1587/transinf.2017EDL8229

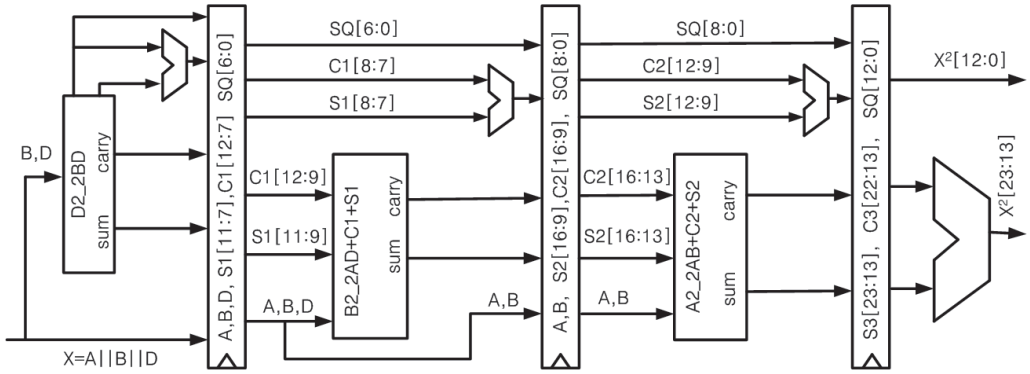

Fig. 3 Architecture of the proposed 12-bit squarer. A, B, and D are 4-bit unsigned integers. The adder in the final stage is an 11-bit adder of which the 23rd bit of $C3$ input is tied to 0 (ground). Those FFs shown in Table 2 can be removed.

signed integer $X = A||B||D$, where A, B, and D are 4-bit unsigned integers, and "||" denotes concatenation. The partial product matrix (PPM) is divided into three parts, as shown by the dotted lines, which constitute the three stages of the four-stage pipeline. Each of the three stages adds the square of A, B, or D to the shifted result of the multiplication of two of A, B, and D. The second and the third stages additionally accumulate the output of the previous stage. Each of the three boxes in Fig. 2 calculates the value written in the box in terms of the two intermediate forms, sum and carry vectors [11]. The squares of A, B, and D are implemented similarly to the "after-merging" PPM shown in Fig. 1. The fourth stage adds up the sum and carry vectors from the third stage to obtain the final square result.

In the front three stages, the maximum time delay depends on the maximum number of bit rows in the PPM to be handled in each stage. Claim 1 analyzes the maximum number of bit rows in the PPMs to be accumulated by a CSA tree in a squarer that is designed in the manner described in Fig. 2:

Claim 1: Consider a squarer for unsigned integers of n bits, where $n = 4k$ for some positive integer $k \geq 2$, designed in the way described in Fig. 2. Let $N_{b,max}$ be the maximum number of bits in any column of the PPM to be accumulated in any stage. Then, $N_{b,max} \leq 4 \times \lfloor \frac{k}{2} \rfloor + 2 \times (k \mod 2) + 2$.

Proof: The operand can be written as $X = X_1||X_2||\cdots||X_k$, where X_i is a 4-bit number for $i = 1, \cdots, k$. Then, when we partition the PPM of X^2 in the form of Fig. 2, the largest number of rows occurs in the part of PPM below X_1, which is denoted as PPM_{X_1} below. When k is even, PPM_{X_1} consists of $2X_1X_k + 2X_2X_{k-1} + \cdots + 2X_{k/2}X_{(k/2)+1}$. Since the number of bits in each column of PPM of X_iX_j $(i \neq j)$ is less than or equal to 4 and at most two bits from the previous stage can be added to each column. Thus, $N_{b,max} \leq 4 \times \frac{k}{2} + 2$. When k is odd, PPM_{X_1} consists of $2X_1X_k + 2X_2X_{k-1} + \cdots + 2X_{(k-1)/2}X_{((k-1)/2)+2} + (X_{((k-1)/2)+1})^2$. The number of bits in each column of PPM of $(X_i)^2$ is less than or equal to 2. □

For the 12-bit squarer, by Claim 1, any column of the PPM contains at most eight bits, which requires a CSA tree

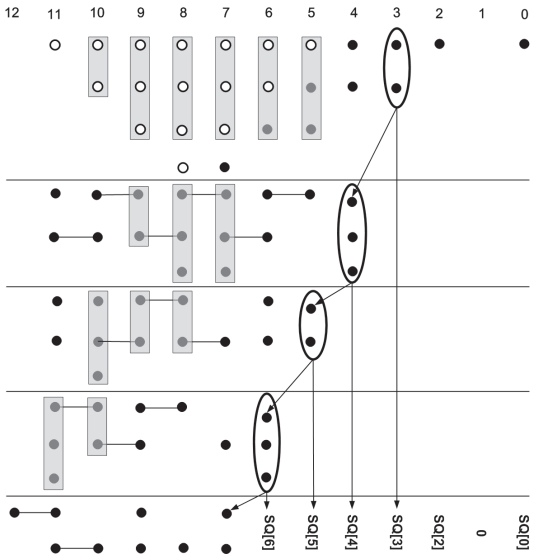

Fig. 4 Organization of the first stage. Unfilled circles represent the PPM of BD whereas the filled circles in the topmost four rows represent the PPM of D^2. Each shaded rectangle represents a full or a half adder in the CSA tree. Each pair of outputs from an adder is tied together with a line segment. Each oval represents a full or a half adder of the ripple-carry adder. Arrows represent the locations of the carry and sum bits.

with up to four levels according to [10]. Using this result, we confine the number of levels of the CSA tree in each stage to less than or equal to four. We evaluate the delay of a half adder as one half of the delay of a full adder (Δ_{FA}) in the evaluation. We also distribute the burden of the final adder in the fourth stage into the front three stages; otherwise, the critical path would occur in the fourth stage.

Figure 3 shows the architecture of the proposed squarer. Block `D2_2BD` consists of the gates for generating the PPM of $2^5BD + D^2$ (Fig. 2), and a CSA tree for accumulating the PPM. Figure 4 describes the organization of the first stage. The low seven bits of the final square value are constructed in a ripple-carry fashion as shown in Fig. 4. The sum and carry vectors, which are $S1$ and $C1$, respectively, and the obtained part of the final square value `SQ[6:0]` are stored together with the input into the pipeline

register (Fig. 3). The maximum time delay in the first stage is $3.5\Delta_{FA}$.

In the second stage, the most significant three and four bits of $S1$ and $C1$, respectively, are merged and accumulated with the PPM of $2^9AD + 2^8B^2$ by the CSA tree in the block `B2_2AD+C1+S1`. Figure 5 describes the organization of the second stage. We adjust the CSA trees in the first stage to generate the "always-0" bits in $C1$ such that the number of levels of the CSA tree is reduced in the second stage. Although the implementation in Fig. 5 has a column with seven bits, the PPM is sparse such that the PPM can be accumulated by a CSA tree with three levels. The sum and carry vectors that are generated in the second stage are $S2$ and $C2$, respectively. The maximum time delay in Fig. 5 is $3\Delta_{FA}$. Only two bits of the final square value are generated in a ripple-carry fashion. Our trials for generating additional bits of the final square value will increase the maximum time delay.

In the third stage, the most significant four bits of $S2$ and $C2$ are merged and accumulated with the PPM of $2^{16}A^2 + 2^{13}AB$ by the CSA tree with three levels in the block `A2_2AB+C2+S2`. In addition, four more bits of the final square value, `SQ[12:9]`, are calculated in a ripple-carry way with the maximum time delay of $3.5\Delta_{FA}$. The final addition of the two results, $S3$ and $C3$, is performed by an adder in the fourth stage to produce the remaining 11 bits of the final square value. In the design described above, the critical path delay is $3.5\Delta_{FA}$ as long as the maximum time delay in the final adder does not exceed $3.5\Delta_{FA}$.

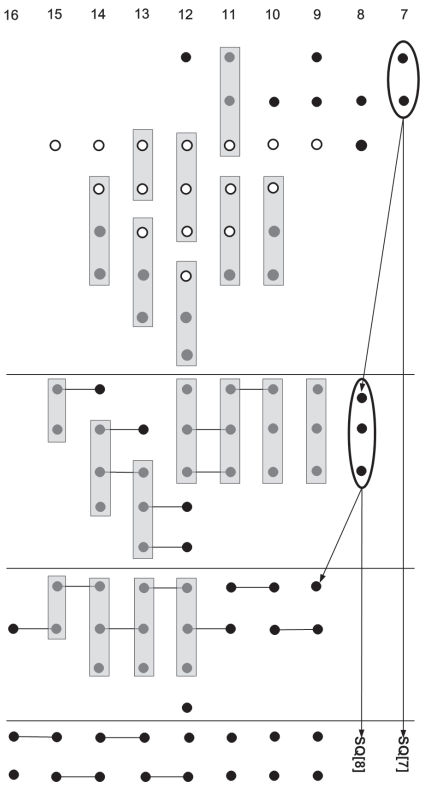

Fig. 5 Organization of the second stage. Topmost two rows are from the first stage. The 3rd to 8th rows show the PPM of $2^9AD + 2^8B^2$. Merging is not applied to the most significant bit of B^2.

3. Evaluation

This section presents the results of an FPGA implementation in evaluating the performance of the proposed squarer. For comparison, the following four squarers are considered:

1) SquarerA is the multiplier that uses the dedicated DSP hardware in the FPGA and is provided in the library of parameterized modules (LPM) of Quartus Prime 17.0 [12]. The operating frequency of this squarer cannot be increased by adding the pipeline stages. (Non-pipelined)
2) SquarerB is the multiplier that is provided as `LPM_MULT` in the LPM of Quartus Prime 17.0 [12]. (4-stage pipelined)
3) SquarerC is the squarer obtained from the `LPM_MULT` core in the LPM of Quartus Prime 17.0 [12], with the configuration option of "squaring operation". Three-stage pipeline outperforms four-stage pipeline for this squarer. (3-stage pipelined)
4) SquarerD is the squarer shown in Fig. 6. The PPMs are accumulated in a similar way to the accumulation in the proposed squarer. However, the final addition is separated as an independent stage. (4-stage pipelined)

The proposed squarer and the four squarers listed above have been modeled in Verilog and implemented with Quartus Prime 17.0, targeting an Intel Stratix FPGA, 5SGSMD3E1H29C1 [13]. Table 1 summarizes the implementation results. Multipliers are generally used for computing squares. SquarerA uses the dedicated multiplier circuit in the FPGA and operates fast, but its operation frequency cannot be increased further.

The remaining squarers are implemented using logic elements and flip-flops (FFs). No DSP block nor blockRAM

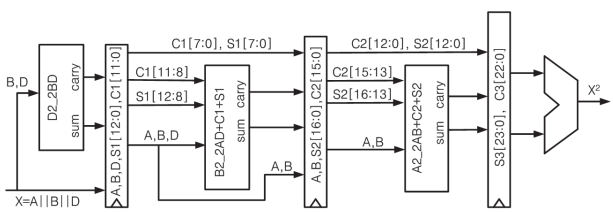

Fig. 6 Architecture of SquarerD.

Table 1 Comparison with prior works. The maximum operation frequencies shown are estimated with the Slow 900 mV 85°C model.

	Logic utilization [ALMs]	Registers [FFs]	Max. Op. freq. [MHz]
SquarerA [12]	18 (+ 1DSP)	35	751
SquarerB [12]	103	204	822
SquarerC [12]	62	116	820
SquarerD	82	134	914
Proposed	79	123	1010

Table 2 Removable FFs in pipeline registers. D[0] can be merged with SQ[0]. The others always contain 0.

Register	Removable FFs
A,B,D,S1[11:7],C1[12:7],SQ[6:0]	D[0],C1[8,10],SQ[1]
A,B,S2[16:9],C2[16:9],SQ[8:0]	SQ[1]
S3[23:13],C3[22,13],SQ[12:0]	S3[21],C3[14,16],SQ[1]

is used. Using a four-stage pipeline, the multiplier SquarerB can achieve a speedup over SquarerA at a significant expense of hardware. SquarerC exploits the identicalness of two inputs of squaring operations and can thus greatly reduce the hardware cost while keeping the operation frequency nearly the same. However, the maximum operation frequency of SquarerC is achieved with three pipeline stages and cannot be increased further even with four pipeline stages.

A multiplier is commonly pipelined by dividing its operation functionally into the generation of PPM, the accumulation of PPM, and the final addition. Then some parts of final addition can be moved into the front stages, as in [14], and the process of retiming may follow. SquarerD divides its operation by the amount of computation and can increase the operation frequency by approximately 11% higher than that of SquarerC. The number of FFs used in SquarerC in Table 1 is different from that in Fig. 6 because there are removable FFs in the figure as in the proposed squarer. SquarerD adopts an optimized CSA tree but naively places the final ripple-carry adder in the final stage, which makes the critical path in the squarer.

By distributing the final adder over front pipeline stages quantitatively, the proposed squarer costs less than SquarerD but operates approximately 23% faster than SquarerC. The gate usages are almost the same: the proposed squarer uses 54 FAs and 20 HAs, whereas SquarerD uses 54 FAs and 19 HAs. The reduction in hardware cost is due to the reduction in the usage of FFs. The number of FFs used in Fig. 3 is 97. However, nine FFs are removable in the proposed design (Table 2). The proposed squarer uses additional 35 FFs for I/O registers as other squarers, because bit 1 is always zero in any square result. As the final adder in the fourth stage, an 11-bit ripple-carry adder is adopted, which operates faster than the CSA trees in the other stages when they are implemented in the FPGA used in this evaluation. If the fourth stage is slower than any of the other stages in other implementation technologies, then other fast adder architectures must be considered at the cost of hardware.

4. Conclusion

This paper has proposed a pipelined hardware squarer that can square unsigned integers of up to 12 bits. The pipeline stages are balanced such that the squarer operates faster than prior comparable designs. In our evaluation using an FPGA, the proposed squarer outperforms the pipelined squarer provided in the LPM of Intel Quartus Prime 17.0.

The proposed pipelining scheme can be extended to the 16-bit design, which can perform a 16-, a 12-, or two 8-bit squaring operations in every cycle; the last two of which are popular data sizes in video applications. For squaring a 16-bit unsigned integer, at most 10 bits are needed to be accumulated in each column by Claim 1. We can reduce the maximum number of bits over all columns to 9 by adjusting the carry vector from the previous stage and, thus, accumulate the PPM by a CSA tree with four levels. The actual implementation results will be dealt with in future works.

Although the number of bits estimated by Claim 1 grows slowly as the operand size increases, the columns of PPMs should be divided into multiple pipeline stages for larger operand sizes, which will also be dealt with in future works.

Acknowledgments

This work has been supported by a Korea University Grant.

References

[1] S.L. Chiu, “Fuzzy model identification based on cluster estimation,” J. Intelligent and Fuzzy Syst., vol.2, pp.267–278, 1994. DOI: 10.3233/IFS-1994-2306

[2] K.-J. Cho, “Efficient unsigned squarer design techniques,” IEICE Electron. Express, vol.9, no.6, pp.422–428, March 2012. DOI: 10.1587/elex:9.422

[3] S. Bui and J.E. Stein Jr., “Additional Optimizations for Parallel Squarer Units,” Proc. IEEE Int. Symp. Circuits and Syst., pp.361–364, July 2014. DOI: 10.1109/ISCAS.2014.6865140

[4] A.G.M. Strollo and D.D. Caro, “Booth folding encoding for high performance square circuits,” IEEE Trans. Circuits and Syst. II, vol.50, no.5, pp.250–254, May 2003. DOI: 10.1109/TCSII.2003.810574

[5] D.W. Matula, “Higher radix squaring operations employing left-to-right dual recoding,” Proc. 19th IEEE Symp. Computer Arith., pp.40–47, June 2009. DOI: 10.1109/ARITH.2009.34

[6] Y. Tanaka and S. Wei, “Efficient squaring circuit using canonical signed-digit number representation,” IEICE Electron. Express, vol.11, no.2, p.20130955, Jan. 2014. DOI: 10.1587/elex.11.20130955

[7] J.-T. Yoo, K.F. Smith, and G. Gopalakrishnan, “A fast Parallel squarer based on divide-and-conquer,” IEEE J. Solid-State Circuits, vol.32, no.6, pp.909–912, June 1997. DOI: 10.1109/4.585298

[8] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh, and D. Marr, “Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC,” Int. Conf. Field-Programmable Technology, pp.77–84, Dec. 2016. DOI: 10.1109/FPT.2016.7929192

[9] H. Parandeh-Afshar, P. Brisk, and P. Ienne, “Efficient synthesis of compressor trees on FPGAs,” Proc. ASP-DAC, pp.138–143, 2008. DOI: 10.1109/ASPDAC.2008.4483927

[10] L. Dadda, “Some schemes for parallel multipliers,” Alta Frequenza, vol.34, pp.349–356, 1965.

[11] I. Koren, Computer Arithmetic Algorithms, 2nd ed., A K Peters, MA, 2001.

[12] Altera Corp., “Integer Arithmetic IP Cores User Guide,” UG-01063, https://www.altera.com, accessed 2017.

[13] Altera Corp., “Stratix V Device Overview,” SV51001, https://www.altera.com, accessed 2017.

[14] Q. Li, G. Liang, and A. Bermak, “A High-speed 32-bit Signed/Unsigned Pipelined Multiplier,” Fifth Int. Symp. Electron. Design, Test & App., pp.207–211, 2010. DOI: 10.1109/DELTA.2010.10