一般社団法人 電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report

2 乗則を用いたデジタル乗算器アルゴリズムと FPGA 実装の検討

佐々木　秀†　　小林春夫‡

群馬大学理工学府　〒376-8515 群馬県桐生市天神町 1-5-1

E-mail:　†t15804040@gunma-u.ac.jp,　‡koba@gunma-u.ac.jp

あらまし　デジタル乗算器はデジタル計算機、DSP チップ等に広く用いられているが、直接的に実現すると全加算器の２次元配列になり、回路規模・消費電力・演算時間が比較的大きくなってしまう。そのため回路規模・消費電力の減少および高速化のために様々なアルゴリズムが提案され、それに基づきデジタル乗算器が設計・実現されてきている。この論文では入力 A，B に対し　その和と 2 乗から積 AB を計算する式を用いてデジタル乗算器を設計する方式を検討した。また、そこで A, B のそれぞれの上位ビット、下位ビットに分割して計算量を低減する方式 (Divide & Conquer 法) を用いる。提案アルゴリズムを FPGA で実現・検証をすべく、検討したアルゴリズムで 8bit×8bit （出力：16bit），16bitx16bit (出力 32bit)の乗算器を設計し RTL シミュレーションを行い、正しくデジタル乗算が行われていることを確認した。提案方式で内部をパイプライン構成にすれば高速化が可能であり、小規模・高速のデジタル乗算器を実現でき、一つのチップ内に多数配置でき大量のデータ処理が可能になる。

キーワード　デジタル乗算器，　２乗則，デジタル回路，　FPGA

Study of Digital Multiplier Algorithms Using a Square Law Its FPGA implementation

Shu Sasaki†　　Haruo Kobayashi‡

Gunma University　1-5-1 Tenjincho, Kiryu-shi, Gunma, 376-8515 Japan

E-mail:　†t15804040@gunma-u.ac.jp,　‡koba@gunma-u.ac.jp

Abstract　Digital multipliers are widely used in digital computers and DSP chips. However, if they are implemented directly, a two-dimensional array of full adders are required, which leads to large hardware and power as well as relatively long calculation time. For this reason, a variety of digital multiplier algorithms and circuits have been proposed for their reduction. In this paper, we consider to design a digital multiplier using the formula to calculate the product AB from the squares of inputs A, B and A+B, and their addition/subtraction. Furthermore, the divide & conquer technique which divides the long bit data into the upper and lower bits and calculates them separately for hardware and calculation reduction. The designed multipliers (8x8, 16x16) with the proposed algorithm have been validated with the RTL level simulation and FPGA implementation.

Keywords　Digital Multiplier, Square Law, FPGA, Digital Circuit

1. 研究背景と目的

1.1. 研究背景

デジタル乗算器はデジタル計算機、DSP チップ等に広く用いられている。2 進数を扱う乗算は、2 進数の加算を何回も繰り返すことで計算できるため、多くの計算量を必要とする。デジタル乗算器を直接的に実現すると全加算器の２次元配列になり、回路規模・消費電力・演算時間が大きくなってしまう問題がある。そのため回路量・消費電力の減少および高速化のために様々なアルゴリズムが提案され、それに基づきデジタル乗算器が設計・実現されてきた。

特にデジタル通信システムでは複雑・大量のデジタル演算をリアルタイムに行う必要があり、小規模なデジタル乗算器を実現できれば、それらを多数実装・並列演算できるので、そのアプリケーションにはきわめて有効である。

ここでは２つのデジタル入力 A，B に対し、その和と 2 乗から積 AB を計算する下記の式を用いてデジタル乗算器を実現する方式を検討する。

$$AB = \frac{1}{2}\{(A + B)^2 - A^2 - B^2\} \quad \cdots\cdots(1)$$

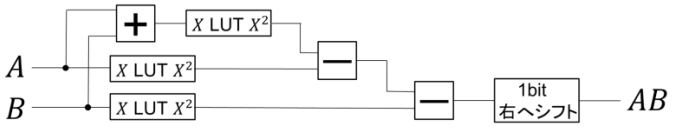
この 2 乗演算のためにルックアップテーブル (Look-Up Table: LUT) を用いる。また LUT と加算器、減算器のサイズを小さくするため A, B, A+B のそれぞれの上位ビット、下位ビットに分割して計算量を低減する方式 (Divide & Conquer 法) を用いる。

提案アルゴリズムを FPGA で実現・検証をすべく、検討したアルゴリズムで入力 8bit×8bit（出力 16bit)、および入力 16bit x 16bit (出力 32bit) の乗算器を設計しレジスタ・トランスファ・レベル（Register Transfer Level: RTL）シミュレーションを行い、正しくデジタル乗算が行われていることを確認する。そして FPGA で実装し、実際に動作可能かどうかを検証する。この提案方式を実現した回路を内部でパイプライン構成にすれば回路全体の高速化が可能であり、小規模・高速のデジタル乗算器を、一つのチップ内に多数配置して大量のデータ処理を行うことが期待できる。なお、ここでは簡単のため A, B は正の数の場合のみを扱っている。

2. 検討する計算アルゴリズム

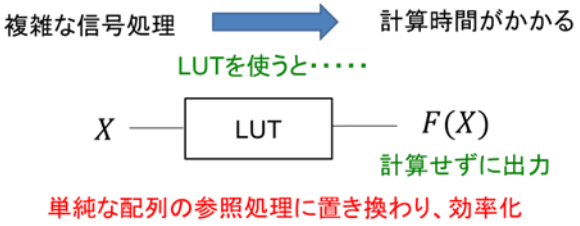
2.1. 2乗則アルゴリズム

検討したアルゴリズムは式 (1) の 2 乗則を用いる。1/2 は右シフト演算（実際には結線変更のみでよい)、二乗の計算部分はルックアップテーブル (Look-Up Table: LUT) を用いることで回路規模削減・低消費電力化・計算時間短縮を実現する。図 1 に式(1)を実現する回路構成を示す。



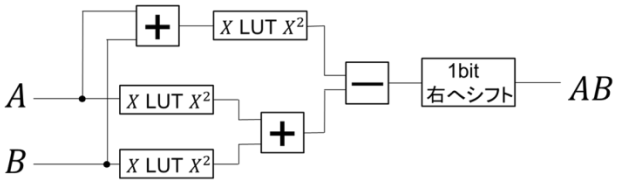
【図 1：式(1) を実現する回路構成】

LUT はメモリ(RAM または ROM)である。入力はメモリの”アドレス”であり、出力はメモリの”データ”である。メモリ内に計算データを記憶しておくことで、LUT で所望の計算結果を得ることができる。

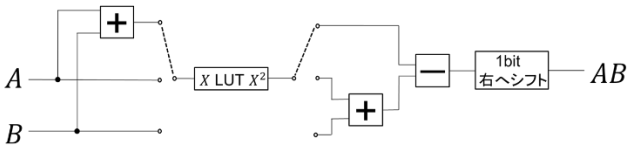


【図 2：ルックアップテーブル(LUT)回路】

図 1 では 3 つの LUT を用いている。しかし演算時間のタイミングを考慮していないため、図 3 のような回路構成にすることで演算時間のバランスを考慮した回路も考えられる。他にも、1 つの LUT を$A^2, B^2, (A+B)^2$ の計算を行うために逐次的に使うこともできる（図 4 参照)。演算時間は約 3 倍と大きくなってしまうが回路量を減らすことができる。この際 3 つの加減算器も 1 つにすることができ、全体として 3 分の 1 の回路量でよい。



【図 3：演算時間のバランスを考慮した回路】



【図 4：LUT を逐次使用した回路】

2.2. 対数を用いるアルゴリズムとの比較

デジタル乗算を容易に計算する方法の一つとして、対数を用いる演算する手法がある。2 つのデータ A, B に対して AB の乗算を、加算器と LUT を用いて次のように計算する。

① A, B の対数 log A, log B を
対数データの LUT を用いて得る。

② log A + log B (= log AB) を
加算器で得る。

③ log AB から AB を
指数データの LUT を用いて得る。

しかし高精度で対数、指数を得るためには LUT のビット数を大きくしなければならず、回路規模が大きくなってしまい、本研究の目的に沿わないため今回は検討対象外とした。

2.3. 次ステップの検討アルゴリズム

次の 2 乗則アルゴリズムも有力である。

$$AB = \frac{1}{4}\{(A+B)^2 - (A-B)^2\} \quad \cdots\cdots\cdots (2)$$

加減算回数は 3 回と同じであるが、LUT 参照処理は 2 回になる。ここでは簡単のため A, B が正の数の場合のみを扱っているので 式(1)を検討した。次のステップとして式(2)を検討する予定である。

2.4. Divide & Conquer について

ここでは二乗計算を担う LUT の回路量低減について検討する。

適当な入力値 A のビット数 n を大きくすると出力A^2の 計算精度を保つためには LUT のサイズを（n^2に比例して）大きくしなければならない。そこで n を上位

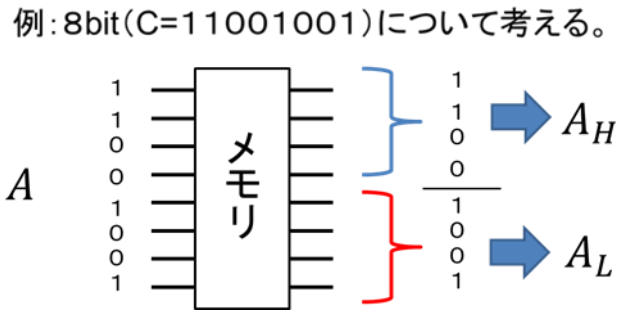
ビットと下位ビットに分割して計算する方式（Divide & Conquer 法）を考える。

たとえば図 5 のように入力 A が 8 ビットで、その上位 4bit をA_H, 下位 4 ビットをA_L とする。ここでA^2の計算は次のようになる。

$A^2 = A_H^2$ (8bit 左シフト)$+2A_H A_L$ (4bit 左シフト) $+A_L^2$ ……(3)

式(3) を用いた入力 A (8bit) の 2 乗A^2 (16bit) 回路の計算回路を図 3 に示す。8 ビットの入力信号 A を上部 4 ビットと下部 4 ビットに分割してビット数を減らしてから計算し、シフト演算によって桁を揃える。

ここでシフト演算は回路上では配線の配置を換えるだけで実現できるため余計な回路を必要しない。



【図 5：データ A の上位、下位ビット分割】

分割したビット列はさらに分割することができる。この方式を用いることで、演算に必要なビット数を分割して小さなビット数の LUT を用いて演算できるようになる。

図 5 を参照し、8bit で$A = 11001001 = (201)_{10}$について考える。まず 10 進数を用いて理論計算を行う。

上部（A_H）4bit、下部（A_L）4bit に分割すると、

$A_H = 1100\ (12)_{10}$、

$A_L = 1001\ (9)_{10}$

となり、これを使って演算を行っていく。

まず分割したA_H、A_Lを 2 乗する。

$A_H^2 = 12^2 = 144$

$A_L^2 = 81$

次にA_H^2を 8bit 左にシフトして桁を合わせる。

A_H^2 (8bit 左) $= 12^2 \times 256 = 36864$

次に分割したA_H、A_Lを足し、2 乗する

$A_H + A_L = 12 + 9 = 21$

$(A_H + A_L)^2 = 21^2 = 441$

そして求めた値を回路に基づいて減算していく。

$(A_H + A_L)^2 - A_H^2 = 441 - 144 = 297$

$(A_H + A_L)^2 - A_H^2 - A_L^2 = 297 - 81 = 216$

$\{(A_H + A_L)^2 - A_H^2 - A_L^2\}$ (4bit 左)

$= 216 \times 16 = 3456$

最後に A の 2 乗と、分割して求めた解が合っているかを確かめる。

$(A_H)^2$ (8bit 左) $+ (A_H + A_L)^2$ (4bit 左) $+ (A_L)^2$

$= 36864 + 3456 + 81$

$= 40401 = 201^2 = A^2$

このように、分割して求めた解とA^2の値が一致したので、この回路の正当性が示せた。

次に、2 進数において上記と同様の手順で理論計算を行う。上部（A_H）4bit、下部（A_L）4bit に分割する。

$A_H = 1100\ (12)_{10}$、

$A_L = 1001\ (9)_{10}$

さらに演算を進める。

$A_H^2 = 10010000\ (144)_{10}$

A_H^2 (8bit 左) $= 1001000000000000\ (36864)_{10}$

$A_L^2 = 1010001\ (81)_{10}$

$A_H + A_L = 00010101\ (21)_{10}$

$(A_H + A_L)^2 = 110111001\ (441)_{10}$

$(A_H + A_L)^2 - A_H^2 = 100101001\ (297)_{10}$

$(A_H + A_L)^2 - A_H^2 - A_L^2 = 11011000\ (216)_{10}$

$\{(A_H + A_L)^2 - A_H^2 - A_L^2\}$ (4bit 左)

$= 110110000000\ \ (3456)_{10}$

$(A_H)^2$ (8bit 左) $+ (A_H + A_L)^2$ (4bit 左) $+ (A_L)^2$

$= 1001000000000000\ (36864)_{10}$

$+ 110110000000\ (7056)_{10}$

$+ 1010001\ (81)_{10}$

$= 1001110111010001\ (40401)_{10} = A^2$

以上より 10 進数の場合と同じ出力結果を得られることが確認できた。

2.5. 問題の分割（Divide & Conquer 法）による演算速度の向上

大きなビット数を持つ LUT を分割することで回路面積を小さくすることが出来ると同時に、演算速度も大幅に縮小できる。

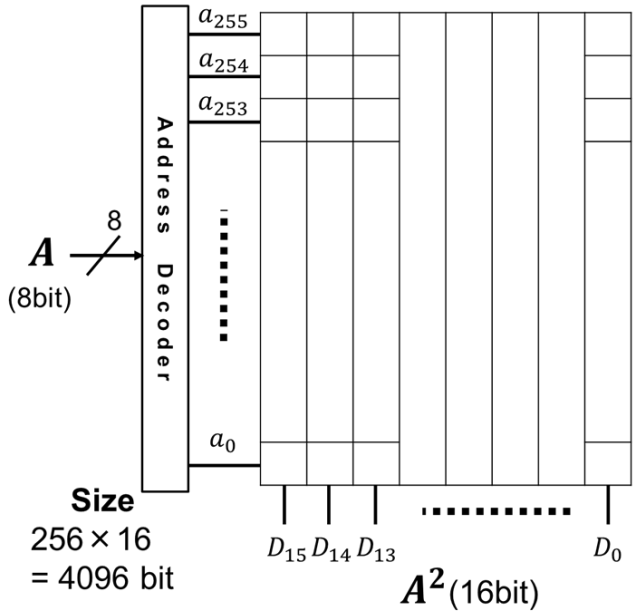
加算器はビット数が大きくなるほど消費電力や計算時間を大きくなってしまう[3]。LUT でも同様であり、データを分割することによって、演算処理をする加算器側のビット数が減り、演算速度が向上できる。また、より大きなビット数から分割することによって、その向上性能も大きくなることが分かる。

演算の際に生じる遅延時間もビット数が小さいほど少ない。これによって遅延による誤差も減少させることができる。

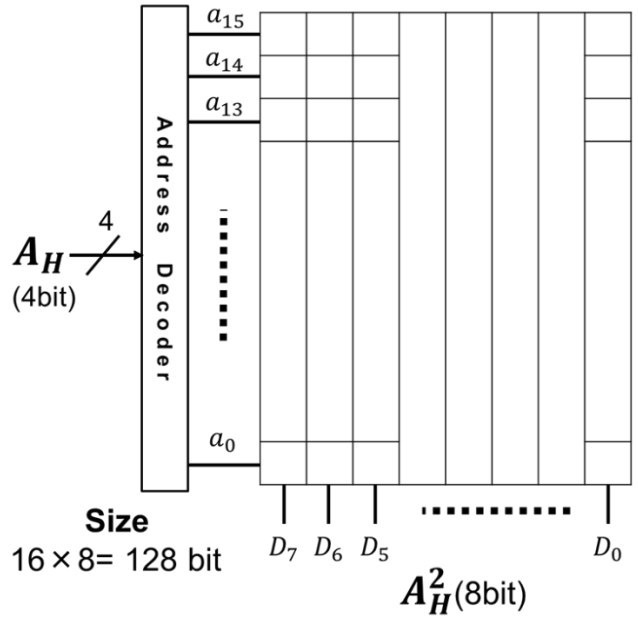
ビット数が半分になると、加算器は面積、演算速度、消費電力も半分になるため、分割して計算することで性能が向上する。ここでは簡単のため、4bit まで分割して理論計算と回路構成を行った。

図 6 のように、8bit 用の LUT では 4096bit が必要になるが、4bit 用の LUT では必要な bit 数は 128bit とな

っており、大幅な回路量の減少が見込める。



【図 6-1. 8bit で扱う LUT の簡略図】



【図 6-2. 4bit で扱う LUT の簡略図】

3. シミュレーションによる検討

検討アルゴリズム・回路構成の正当性、つまり検討した内容が実装に反映させることが出来るかどうかを検証するため「Verilog HDL」を用いて回路シミュレーションを行った。

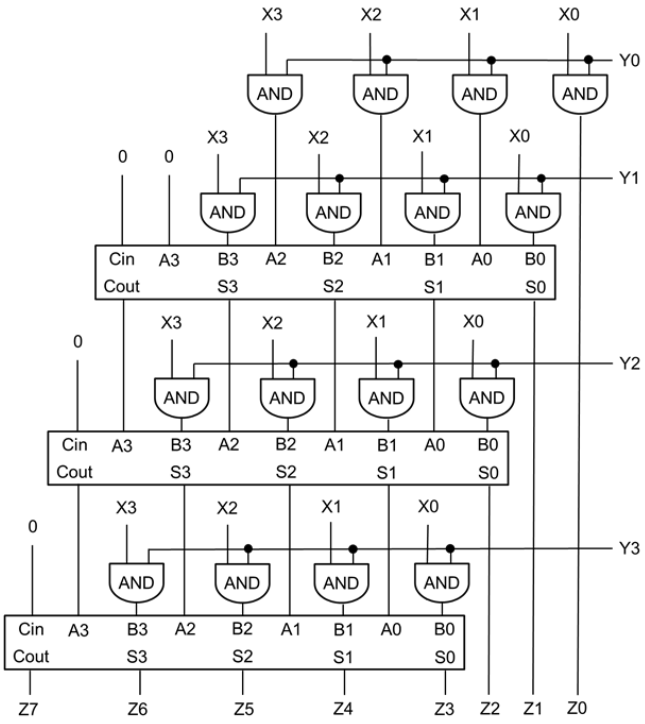
具体的には、シミュレーションソフト上に回路構成を実現し、二つの入力値を変えて演算させて結果を出力し、結果が正しいかどうかをチェックした。入力が 4bit×4bit で出力が 8bit の場合は、数字の組み合わせが 16×16 = 256 通り、入力が 8bit×8bit で出力が 16bit の場合、 256×256 = 65536 通り、入力が 16bit×16bit で出力が 32bit の場合は 65536×65536 = 4294967296 通りある。それらすべての数値で提案アルゴリズムが正しく乗算を行っていることを確認した。

シミュレーション結果を図 12 に記載する。このプログラムを使用すれば提案アルゴリズムの FPGA 実現ができる。今回は【Spartan3E】を用いることで 4bit×4bit の回路を実装し、その動作を確認した。

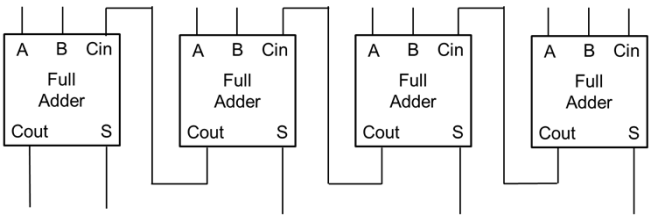
4. 従来乗算器との比較

計算アルゴリズムを用いて乗算器の回路設計を行い、シミュレーション、FPGA 実装を経て正しく動作することを確認した。

ここで、従来の乗算器との比較を行う。最初に述べたとおり、直接的なデジタル乗算器は加算器を連続的に組み合わせて構成されている（図 6-1,6-2 を参照）。

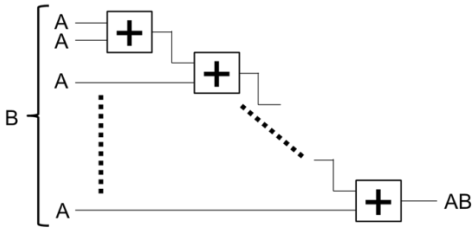


【図 7-1. 従来の乗算器の回路図（4bit)】

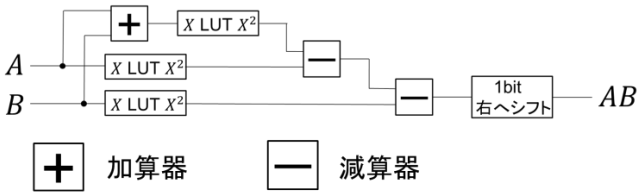


【図 7-2. 全加算器の回路図】

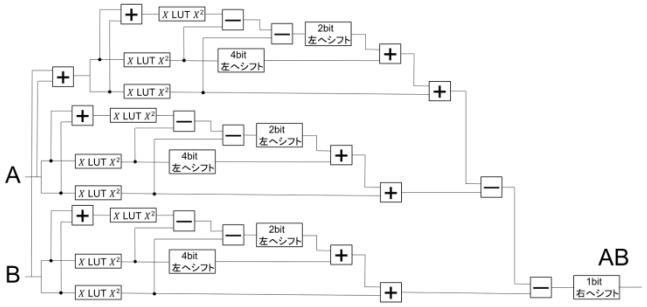
この回路と、理論を元に構成したデジタル乗算回路と RTL レベルで比較する。（図 8,9,10 を参照）



【図 8：従来の乗算器の回路図】



【図 9：構成した回路図（divide&conquer 無し）】



【図 10：構成した回路図（divide&conquer 有り）】

5. まとめと今後の課題

二乗則の計算アルゴリズムを用いた回路構成と、そのハードウェア実現を検討し、FPGA 実装のための RTL レベルシミュレーションで動作を確認した。今後は具体的には次を重点的に検証していく。

・従来法との回路量の定量的な比較
・計算精度と演算器・LUT のビット数の明確化
・実装 FPGA 動作クロック周波数）と
　計算速度の明確化
・マイナスの数値の場合の計算。

デジタル乗算器はすべて 2 進数で表現されており、マイナスがつくと 2 の補数によってマイナスを表現する。ビット分割の際にはその配慮が必要であるが、今後はそれに対応していく。

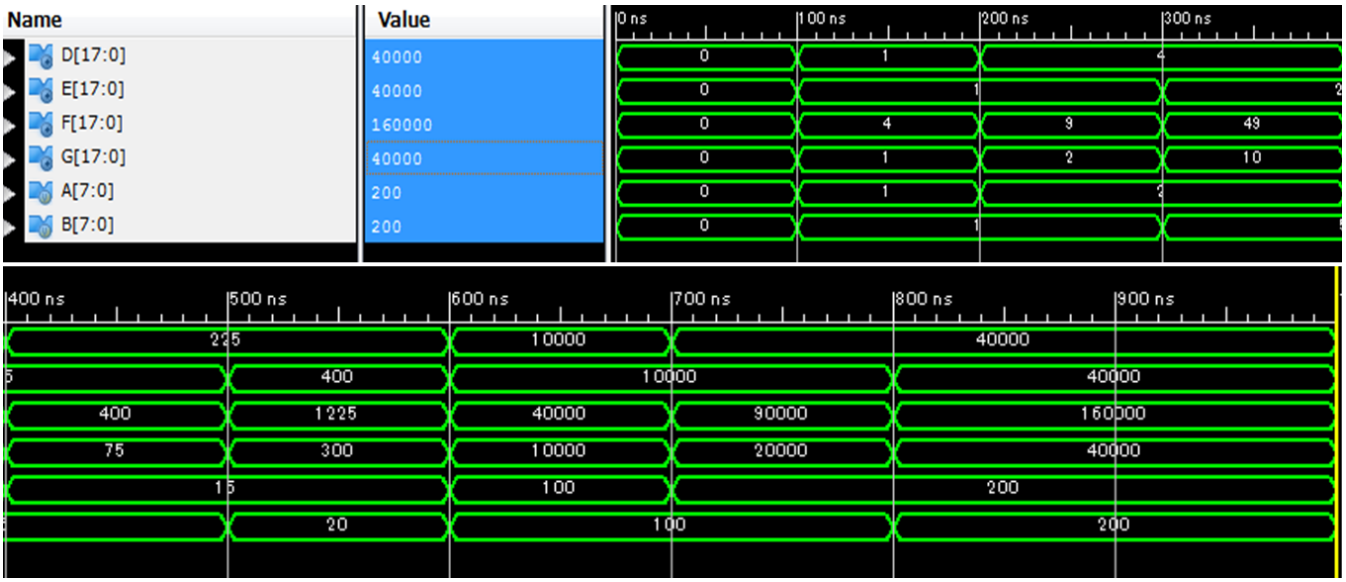
・式(2) のアルゴリズムの適用

謝辞： FPGA 実装にアドバイスをいただきました弓仲康史先生、王俊善さん、李从兵さんに感謝いたします。

【参考文献】

[1] A. V. Oppenheim, and R. W. Shafer, Digital Signal Processing, Printice-Hall (1975)
[2] L. Cohen, Time-Frequency Analysis, Prentice Hall
[3] A Technical Tutorial on Digital Signal Synthesis, Analog Devices, Inc. (1999).
[4] N. Weste, D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Addison Wesley (2010)
[5] 佐々木秀、小林春夫「短時間スペクトラム解析の計算アーキテクチャの検討」第 5 回電気学会東京支部栃木・群馬支所合同研究発表会　ETT-15-69, ETG-15-69 宇都宮　(2015 年 3 月)
[6] 佐々木秀、小林春夫「2 乗則を用いたデジタル乗算器アルゴリズムの検討」第 38 回多値論理フォーラム、北海道大学　(2015 年 9 月)

付録　提案手法を用いたデジタル乗算器の RTL シミュレーション



【図 11：8bit×8bit のシミュレーション】

Devide&Conquer 方式を用いた 8bit×8bit の回路プログラムを入力し、入力値 A，B の値を 100ns、または 200ns 毎に変化させ、その区間における演算結果を波形上に表示した。

演算結果の確認として

D（18bit）$= A^2$

E（18bit）$= B^2$

F（18bit）$= (A + B)^2$

を表示しており、

これらを用いて次式を演算した。

$$G\text{（18bit）} = \frac{1}{2}\{(A+B)^2 - A^2 - B^2\}$$

図 11 中の D，E，F，G それぞれには処理することのできる最大 bit 数を示している。この bit 数は入力の bit 数によって変化し、入力値の bit 数の倍以上の容量が必要になる。

Value における数値はシミュレーションの最終結果を表示しており、今回のシミュレーション結果での最終値は次のようになる。

A = 200

B = 200

D = 40000

E = 40000

F = 160000

G = 40000

このため、Value ではその値が表示されている。また、これらの計算はすべて 2 進数で行っているが、わかりやすくするため結果を 10 進数で表示した。

図 11 中に示したとおり、それぞれの入力値 A，B に対して出力 G は A×B の値を導いている。これで検討したアルゴリズムが回路上に反映できることが示せた。