セレクタ論理に帰着させたアルファブレンド演算器を用いた画像間合成回路のFPGA実装

五十嵐 啓太† 柳澤 政生† 戸川 望†

† 早稲田大学大学院基幹理工学研究科情報理工・情報通信専攻

透明効果のある画像の合成を行う場合，画素毎に透過率による重み付き演算を行う．合成する画像の枚数が増えるほど演算が膨大になるため，より高速な処理が必要とされる．本稿では，セレクタ論理を利用した 2 枚の画像をアルファブレンドする手法を提案し，これを FPGA 上に実装し，その効果を確認する．セレクタ論理の出力値域は必ず 1 以下になるという特性を用いて，桁上げ処理時間を削減する．セレクタ論理を画像処理に適用させることで，アルファブレンド演算中の積項が減少し，面積と遅延が減少する．この回路を FPGA 上に実現した．画像の色情報と透過率を FPGA 上の BRAM に格納し，透過率を基にアルファブレンドを行い，合成後の色値を決定する．アルファブレンドされた色値は再び BRAM に格納され，全ての画素でアルファブレンド処理が終了次第，画像間合成処理は終了する．本実装では，RGB 計算を並列計算し，冗長な演算を削減した．アルファブレンド演算をビットレベル式変形することで得られた積項をそのまま加算する画像間合成回路と，アルファブレンド演算をビットレベル式変形しセレクタ論理によって積項を減少させた画像間合成回路を FPGA で実装比較した結果，最小動作クロック周期は 8.474ns から 6.523ns に削減され，約 23 %の削減が行われた．

FPGA Implementation and Evaluation of Image Synthesis Circuits Using Selector-logic-based Alpha Blending

Keita IGARASHI† Masao YANAGISAWA† Nozomu TOGAWA†

† Dept. of Computer Science and Communications Engineering, Waseda University

Alpha blending is one of image synthesis techniques, which synthesizes a new image by summing up weighted several input images and produces transparent effect. In this paper, we pick up an alpha blending method using selector logics and implement it on an FPGA board. By applying selector logics to the alpha blending operation, its total product terms are decreased and thus a circuit size and circuit delay are improved. In our implementation, original pixel values are stored into a memory on the FPGA board and then a new pixel value is synthesized based on input transmittance factor. We realize approximately 23% speed-up and approximately 8% area reduction using selector-logic based alpha blending.

1 はじめに

現代社会において身近な機器の多くがマルチメディア化しているという背景がある．例えば，スマートフォンなどの本来であれば通話機能を目的とした携帯電話は，通話機能だけではなく高性能なカメラ機能，指紋認証機能，音声認証機能など，様々な機能が要求されており，限られた面積のチップ中に多くの専用演算器を効率的に配置することが必要とされている [5].

上記のような専用演算の一種にアルファブレンド演算 [4] があり，画像の枚数分の計算が必要とされるため，膨大な計算量になる可能性がある．[2] では，アルファブレンド演算に対してセレクタ論理を適用させることで，セレクタ論理の出力値域は必ず 1 以下になることから演算中の積項が減少し，回路面積と遅延が減少することを示した．アルファブレンド演算は背景画像と前景画像を合成し 1 枚の画像にする画像間演算の一種であり，2 つの画像を透過率による重み付きの合成を行うことで，透明化の効果を得ることができる．アンチエイリアス処理やディゾルブ処理に用いられ，差積演算を含んでいるため，セレクタ論理へ帰着させることで画像処理回路の高速化・最適化が可能である [3]．図 1 にアルファブレンディングの概要を示す．アルファブレンド演算は，二種類の演算式が一般的に知られている．1 つはリアルタイム処理が求められるゲーム開発等で用いられる簡易演算式であり，もう一方は画像処理ソフトで用いられる精度の高い演算式である．本稿で扱うセレクタ論理に帰着させる演算式は後者のものとする．一方，ビットレベル式変形によって演算を高速化することができる場合がある．ビットレベル式変形は演算をビットレベルで表現し整理することで回路設計を最適化する手法である [6]．特に [3] ではセレクタ論理を利用してビットレベルの式を整理することで，演算途中の積項数を削減し画像補間を高速化している．

本稿では，セレクタ論理を利用した合成手法を，画像合成回路として FPGA 上で評価実験を行う．ここで，FPGA ボード上に実装した画像合成回路の処理手順を述べる．PC は画素数が等しい 2 枚の画像を FPGA ボード上の BRAM に格納し，FPGA 側は PC から送信される信号を基に格納された 2 枚の画像の合成が始まる．$(0, 0)$ 座標から順に各ピクセルに保持された透過率を用いて重み付きの合成を行う．合成された色値は再び BRAM に格納される．全画素の合成が終了次

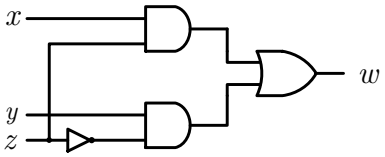

図 1　セレクタ回路.

表 1　セレクタ論理の真理値表.

入力			出力
x	y	z	w
X	Y	0	Y
X	Y	1	X

第，画像合成処理は終了する．本実装では，前景画像の透過率が 0 または 1 の場合，合成処理を省略し，RGB 計算を並列計算することで，演算の冗長性を削減している．

2　セレクタ論理を利用したアルファブレンド演算器設計

本章では，[3] にもとづきアルファブレンディングによる画像の合成手法を紹介し，そこにセレクタ論理を適用する．

2.1　セレクタ論理

セレクタ論理演算とは，2 入力の 1 ビット信号を 1 ビットの制御信号を用いて選択する演算である．入力信号を x，y，出力信号を w，制御信号を z とするとき，$w = (x \wedge z) \vee (y \wedge \overline{z})$ で表現される．アルファブレンディングをセレクタ論理を含む形式にビットレベル式変形を用いて変形することで，演算器をセレクタ論理に帰着させることができる．表 1 にセレクタ論理の真理値表，図 1 にセレクタ論理の回路図を示す．セレクタ論理の出力値域は必ず 1 以下になるという特性から，セレクタ論理に帰着させた部位に関しては桁上げが存在せず，帰着部位を事前演算することが可能である．即ち，セレクタ論理に演算を帰着させること，桁上げ処理時間が削減され，高速化を図ることが可能である．また，桁上げの回数が減少することで，加算器の個数も削減するため，回路面積の削減が可能である．

2.2　アルファブレンディングを用いた画像の合成処理

アルファブレンディングは背景画像と前景画像を合成し 1 枚の画像にする画像間演算であり，2 つの画像を透過率による重み付きの合成を行うことで，透明化の効果を得ることができる [1]．図 2 にアルファブレンディングの概要を示す．透過率は各画素毎に指定され，0 ～ 1 の小数で表現される．本稿で扱うアルファブレンディングの演算式は画像処理ソフトで使用されている式を用いる．背景画像の赤・緑・青の色値 R_b，G_b，B_b，透過率 $A_b (0 \leq A_b \leq 1)$ とし，前景画像について同様に R_f，G_f，B_f，A_f とする．合成画像について同様に R_s，G_s，B_s，A_s とするとき，次式でアルファブレンド演算式は表現される．

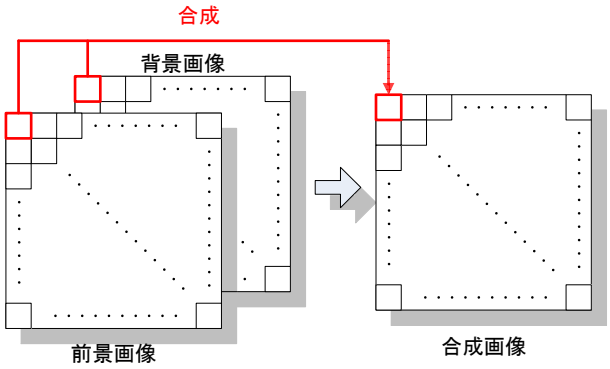

図 2　アルファブレンディングの概要.

$$A_s = \underline{(1 - A_b)A_f} + A_b \tag{1}$$

$$\begin{aligned} A_s R_s &= ((R_f A_f + (R_b A_b (1 - A_f)) \\ &= ((R_f A_f + (V_R (1 - A_f)) \\ &= (\underline{(R_f - V_R)A_f} + V_R) \end{aligned} \tag{2}$$

ここで $V_R = R_b A_b$ とする．式 (1) と式 (2) の下線部は差積演算であり，セレクタ論理に帰着させることが可能である [3]．

2.3　セレクタ論理型アルファブレンディング

n ビットの 2 進数で表記された整数 $a = [a_{n-1} \cdots a_1 a_0]$ を考える．a が 2 の補数表現による符号付き数であるときのビットレベル式表現を以下に示す．

$$\begin{aligned} a &= -a_{n-1}2^{n-1} + a_{n-2}2^{n-2} + \cdots + a_0 2^0 \\ &= -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \end{aligned} \tag{3}$$

式 (3) の先頭ビットは符号ビットである．2 の補数表現による $-a$ は a の各ビットを反転し，1 を加算することで表現できる．よって $-a$ は式 (4) で示すことができる．

$$-a = -\overline{a_{n-1}}2^{n-1} + \sum_{i=0}^{n-2} \overline{a_i} 2^i + 1 \tag{4}$$

ここで，差積演算は，a，b を整数，c を $0 \leq c \leq 1$ を満たす数とすると，ビットレベル式変形を行うことで，以下のように変形できる．

$$\begin{aligned} &(a - b)c = ac + b(-c) \\ &= \sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(a_i c_j + b_i \overline{c_j})2^{i+j} + a_{n-1}2^{n-1}\underline{\left(-\sum_{i=0}^{n-2} c_i 2^i\right)} \\ &+ c_{n-1}2^{n-1}\underline{\left(-\sum_{i=0}^{n-2} a_i 2^i\right)} + b_{n-1}2^{n-1}\underline{\left(-\sum_{i=0}^{n-2} \overline{c_i} 2^i\right)} \\ &+ \overline{c_{n-1}}2^{n-1}\underline{\left(-\sum_{i=0}^{n-2} b_i 2^i\right)} - b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \\ &+ (a_{n-1}c_{n-1} + b_{n-1}\overline{c_{n-1}})2^{2n-2} \end{aligned} \tag{5}$$

負項の部分積は符号ビットの処理が必要のため正の部分積に比べ余分な回路を必要とする．2 の補数表現で

表現すると，式（5）の下線部，すなわち負項は以下の等式が成り立つ．

$$-\sum_{i=0}^{n-2} c_i 2^i = -\left(-0\cdot 2^{n-1} + \sum_{i=0}^{n-2} c_i 2^i\right) = -1\cdot 2^{n-1} + \sum_{i=0}^{n-2} \overline{c_i} 2^i + 1 \quad (6)$$

同様に $-\sum_{i=0}^{n-2} a_i 2^i$，$-\sum_{i=0}^{n-2} b_i 2^i$，$-\sum_{i=0}^{n-2} \overline{c_i} 2^i$ についても変形し，式（5）に代入することで負項を削減した次式を得る．

$$\begin{aligned}
\text{式（5）} &= \sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(a_i c_j + b_i \overline{c_j})2^{i+j} \\
&+ \sum_{i=0}^{n-2} b_i 2^i \\
&+ \sum_{i=0}^{n-2}\{(b_{n-1}c_i + a_{n-1}\overline{c_i}) \\
&+ (\overline{a_i}c_{n-1} + \overline{b_i}\overline{c_{n-1}})\}2^{i+n-1} \\
&+ (a_{n-1}c_{n-1} + b_{n-1}\overline{c_{n-1}} \\
&- a_{n-1} - c_{n-1} - b_{n-1} - \overline{c_{n-1}})2^{2n-2} \\
&+ a_{n-1}2^{n-1} + (c_{n-1} + \overline{c_{n-1}})2^{n-1} \quad (7)
\end{aligned}$$

さらに式（7）の 2^{2n-2} の項に注目し，入力数を削減するために以下の式変形を行う．

$$\begin{aligned}
&(a_{n-1}c_{n-1} + b_{n-1}\overline{c_{n-1}} \\
&- a_{n-1} - c_{n-1} - b_{n-1} - \overline{c_{n-1}})2^{2n-2} \\
&= \{(a_{n-1}c_{n-1} - c_{n-1} - a_{n-1} + 1) - 1 \\
&+ (b_{n-1}\overline{c_{n-1}} - b_{n-1} - \overline{c_{n-1}} + 1) - 1\}2^{2n-2} \\
&= \{(\overline{a_{n-1}}\,\overline{c_{n-1}}) \\
&+ (\overline{b_{n-1}}c_{n-1}) - 2\}2^{2n-2} \quad (8)
\end{aligned}$$

ここで，n ビットの 2 進数で表記された色値 R_b，R_f と透過率 A_b，A_f と整数 V_R を，それぞれビットレベル式表現したときの i ビット目を r_{bi}，r_{fi}，α_{bi}，α_{fi}，v_{ri} とすると，式（8），式（7）より，式（1）と式（2）は次式に変換出来る．

$$\begin{aligned}
A_s &= \sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(\underline{\alpha_{fj} + \alpha_{bi}\overline{\alpha_{fj}}})2^{i+j} \\
&+ \sum_{i=0}^{n-2}\alpha_b 2^{i+1} \\
&+ \sum_{i=0}^{n-2}\big[\{\underline{(\alpha_b)_{n-1}\alpha_{fi} + \overline{\alpha_{fi}}}\} \\
&+ \{\underline{(\alpha_f)_{n-1} + \overline{\alpha_{bi}}\,\overline{(\alpha_f)_{n-1}}}\}\big]2^{i+n-1} \\
&+ \big[\underline{\{\overline{(\alpha_f)_{n-1}}\} + \{\overline{(\alpha_b)_{n-1}}(\alpha_f)_{n-1}\}}\big]2^{2n-2} \\
&- \{(\alpha_b)_{n-1} - 1\}2^{n-1} - 2^{2n-1} \quad (9)
\end{aligned}$$

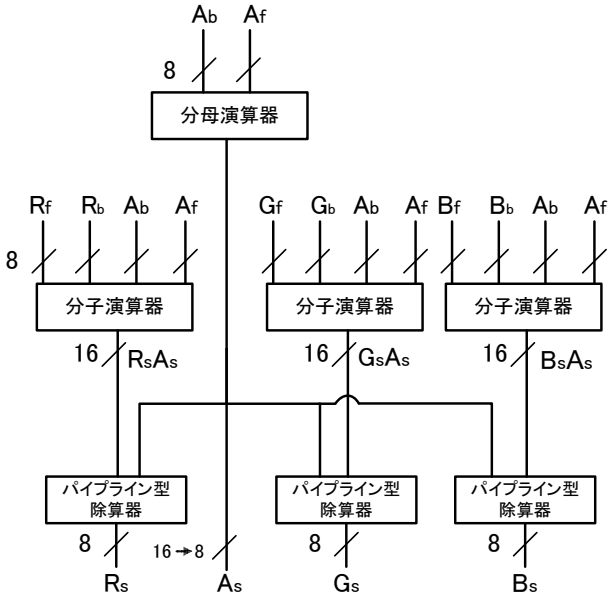

図 3　アルファブレンド演算器の構成．

$$\begin{aligned}
A_s R_s &= \sum_{i=0}^{n-2}\sum_{j=0}^{n-2}(\underline{r_{si}\alpha_{fj} + v_{ri}\overline{\alpha_{fj}}})2^{i+j} \\
&+ \sum_{i=0}^{n-2} v_{ri}2^{i+1} \\
&+ \sum_{i=0}^{n-2}\big[\{\underline{(v_r)_{n-1}\alpha_{fi} + (r_s)_{n-1}\overline{\alpha_{fi}}}\} \\
&+ \{\underline{\overline{r_{si}}(\alpha_f)_{n-1} + \overline{v_{ri}}\,\overline{(\alpha_f)_{n-1}}}\}\big]2^{i+n-1} \\
&+ \big[\underline{\{\overline{(r_s)_{n-1}}\,\overline{(\alpha_f)_{n-1}}\} + \{\overline{(v_r)_{n-1}}(\alpha_f)_{n-1}\}}\big]2^{2n-2} \\
&+ \{(r_s)_{n-1} - (v_r)_{n-1} + 1\}2^{n-1} - 2^{2n-1} \quad (10)
\end{aligned}$$

式（10）の初めの下線部 $r_{si}\alpha_{fj} + v_{ri}\overline{\alpha_{fj}}$ は桁上げを行わず，値は 0 と 1 になることから，この算術値を論理値と同値とみなすことが可能である．そのため，$r_{si}\alpha_{fj} + v_{ri}\overline{\alpha_{fj}} = (r_{si} \wedge \alpha_{fj}) \vee (v_{ri} \wedge \overline{\alpha_{fj}})$ とみなすことができ，この部分をセレクタ論理に帰着できる．式（9），式（10）において，その他の下線部においても同様に考えることができるため，下線部の演算に関しては桁上げが行われないことが約束される．G_s と B_s も同様に計算できる．

2.4　アルファブレンド演算器の設計

設計したアルファブレンディング回路の構成を図 3 に示す．本稿では，RGB を並列にアルファブレンド演算することを考える．A_s は RGB 共通のため，分母演算器を RGB で共通させる．合成後の色値 R_s を求めるためには式（2）から式（1）を除算する必要がある．合成結果の透過率 A_s の値域は $0 \le A_s \le 1$ であるため，式（1）を計算する回路の出力 16 ビットは小数部位 16 ビットをもつ．一方，式（2）を計算する回路の出力は整数部位 8 ビット，小数部位 8 ビットの計 16 ビットで出力される．ここで，両回路における出力の固定小数点位置を揃えるために，入力値である透過値 A を 255 で除算せず，透過値のまま計算に組み込む

と，式（1）は次式のように表現される．

$$式(1) = 255\alpha_s + A_d(1-\alpha_s) \tag{11}$$

これにより，分母計算回路の出力 16 ビットは整数部位 8 ビット，小数部位 8 ビットとなるため，除算回路はシフト演算を行わずとも 16 ビット整数同士の除算を行っていることになる．除算回路の出力結果である 32 ビットから取り出すビット幅を変更するだけで済むため，計算結果は変わらず透過率 A_b を計算するための除算コストを削減できる．ここで，1/255 は循環 2 進小数であるため演算を簡単化することを考える．1/255 のビット列は以下の式で表現される．

$$\begin{aligned} 1/255 &= [0.0000000100000001\cdots]b \\ &= [0.\dot{0}0000000\dot{1}]b \end{aligned} \tag{12}$$

1/255 は式（12）の下線部が示すように循環小数であるため，近似的に 1/256 とすることで固定小数点位置のシフト演算に置き換えることが可能である．本稿で扱う透過率 A の算出は全てこの手法を用いる．また，アルファブレンディング処理において画像データは透過率が 0 または 1 ではない限り，アルファブレンド演算器へ画像データが連続的に入力される．そのため，連続したデータ処理と相性の良いパイプライン型除算器を用いる．

3 FPGA 実装を考慮した画像合成回路の設計

PPM 形式の画像を合成する場合，各ピクセルの色データを RGB に分解し，それぞれにおいて合成処理を行う．つまり，1 ピクセルにつき，Red と Green と Brue で 3 回ずつアルファブレンディングを行う必要があるため，総ピクセル数の 3 倍の処理時間が必要になる．また，前景画像の透過率が 0 または 1 の場合，式（1），式（2）よりアルファブレンド演算を行う必要がないことが分かる．即ち，前景画像の透過率が 0 の場合，不透明となり出力画素は前景画像の対応した画素値になる．前景画像の透過率が 1 の場合，透明となり出力画素は背景画像の対応した画素値になる．本章では，RGB 演算を並列化させ，前景画像の透過率を場合分けすることにより演算の冗長性を削除した画像合成回路を提案する．

3.1 実験環境

FPGA は Xilinx 社が提供している Kintex7 FPGA TB-7V-2000T-LSI 評価ボードを用いる．上記 FPGA ボードの全体図を図 4（a），動作中の様子を図 4（b）に示す．PC と FPGA のデータ通信は USB を用いる．Cypress 社が提供している CyAPI を用いて，FX3 USB コントローラとデータのやり取りを行う．FPGA のバス規格は AXI（Advanced eXtensible Interface），開発ツールは ISE Project Navigator 14.2，論理合成は XST（Xilinx Synthesis Technology），BRAM は FPGA ボード上に搭載されている DDR3 SDRAM を用いる．PC 上で C++ を用いて記述したアルファブレンディング処理の入力データは PPM 形式の 45×39 画素の画像と，アスキーコードで各画素の透過値が記述されたアルファチャンネルを取り扱う．PPM 画像

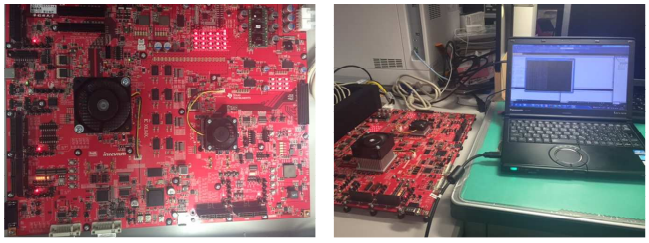

図 4　Kintex7 FPGA TB-7V-2000T-LSI 評価ボード．

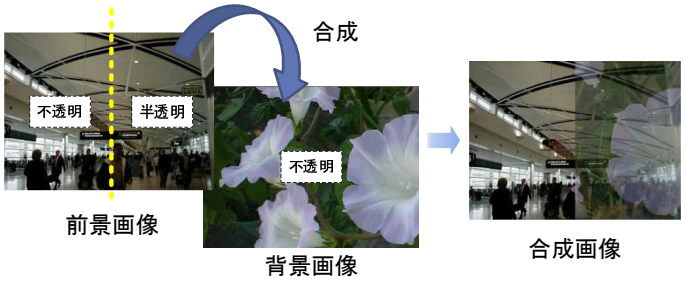

図 5　入力画像と出力画像．

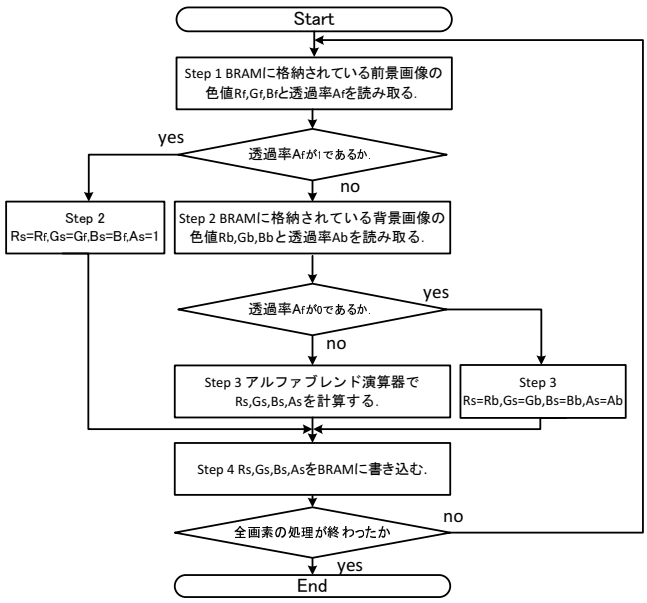

図 6　画像合成回路のフローチャート．

は P3 形式，RGB がアスキーコードで記述されているものを使用した．入力した画像を BRAM に一旦保存し，PC から画像処理回路のスタート信号を受信することで，画像間合成が始まる．画像の透過値は PC からの入力を基に決定する．以下の実験では，背景画像の透過率を全て 1，前景画像の左半分を透過率 1，右半分を透過率 0.5 とした．画像のアルファブレンド処理が終了した後，PC が BRAM に格納されている合成済みの画像データを読み込む．FPGA から読み込まれた画像は PC 上で PPM ファイルとして出力される．図 5 に，本稿で扱う入力画像と出力画像を示す．

3.2 FPGA 実装を考慮した画像の合成処理

PC から全画像データを BRAM へ転送後の画像合成処理の流れを図 6 に示す．合成処理は $(0,0)$ 番地から y 軸方向へ順に計算していき，y が画像の端まで到達したら，y をリセットし x に 1 を加算する．これを全

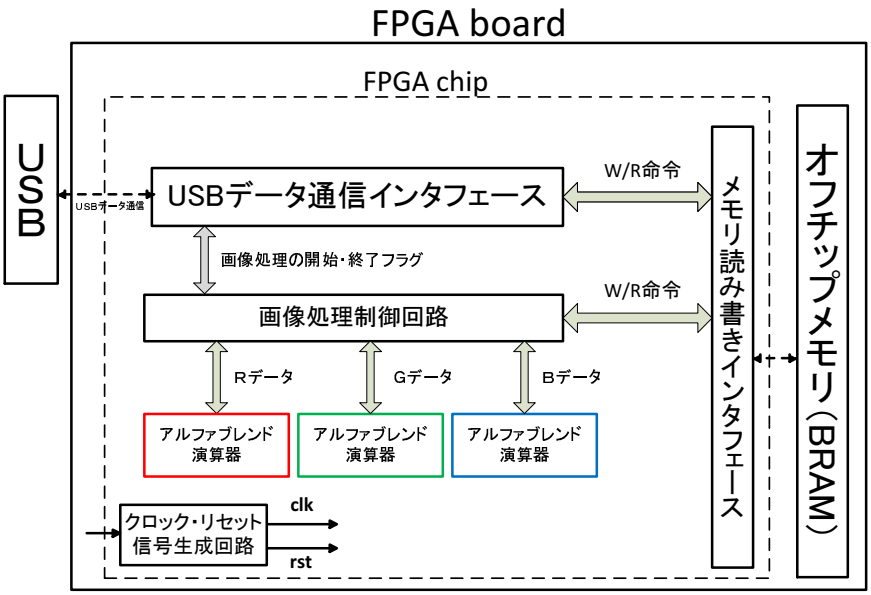

図 7　画像合成回路全体の構成.

画素の処理が終了するまで繰り返す．現在処理を行っている座標を (m,n) としたとき，各ステップにおける処理を以下に示す．

Step 1

BRAM の 1 アドレスには 32 ビットの情報が格納可能であり，色値 RGB は 0~255 の 256 階色で表現され，それぞれ 8 ビット，透過率 A は 8 ビットで表現されるため，1 アドレス毎に 1 画素の情報が格納されている．前景画像の透過率によって，アルファブレンディングを行うかどうかを判定するため，前景画像の座標 (m,n) における画素データを読み込む．

Step 2

前景画像の透過率が 1 の場合，座標 (m,n) の色値を $R_s = R_f, G_s = G_f, B_s = B_f, A_s = 1$ として，$STEP4$ へ移行する．一方，前景画像の透過率が 1 ではない場合，透過率が 0 であるかに関係なく背景画像のデータを用いるため，背景画像の座標 (m,n) の画像データを読み込む．その後，$STEP3$ へ移行する．

Step 3

前景画像の透過率が 0 の場合，座標 (m,n) の色値を $R_s = R_b, G_s = G_b, B_s = B_b, A_s = 0$ として，$STEP4$ へ移行する．一方，前景画像の透過率が 1 ではない場合，R_s, G_s, B_s, A_s のそれぞれの値をアルファブレンド演算器により決定する．その後，$STEP4$ へ移行する．

Step 4

決定した R_s, G_s, B_s, A_s のそれぞれの値を BRAM へ書き込む．BRAM に格納する色値は 24 ビット左シフトした R，16 ビット左シフトした G，8 ビット左シフトした B，A を連結した 32 ビットのデータとして 1 つのアドレスに格納する．

3.3　実装した回路の構成

回路全体の構成を図 7 に示す．FPGA ボード上に実装した回路は大別すると，USB データ通信インタフェース回路，画像処理制御回路，アルファブレンド演算器，メモリ読み書きインタフェース回路，オフチップメモリの 5 つから構成される．アルファブレンド演算器は 2.4 節で，画像処理制御回路は 3.2 節で説明したため，その他の上記主要回路について説明を行う．

3.3.1　USB データ通信インタフェース回路

USB データ通信インタフェース回路は，PC からの入力として 0x00~0x54 番地のレジスタ読み書きを受け付ける（ただし，BRAM 読み書き用の特別なアドレスには読み書き不可）．画像処理制御回路が動作している間は，PC からの入力を受け付けず，メモリ読み書きインタフェース回路から BRAM に読み書きを行うフラグとデータ，アドレスを受け付ける．

3.3.2　メモリ読み書きインタフェース回路

メモリ読み書きインタフェース回路中に BRAM 読み書き用の特別な意味を持つレジスタがあり，各レジスタはアドレス 0x00~0x54 をもつ．特定のアドレスのレジスタにデータをセットすることにより，BRAM への読み書きを制御する．以下の手順でレジスタに書き込みを行うことで，BRAM へデータを書き込む．

1. 0x08 番地のレジスタに 1 を格納する．
2. 0x17 番地のレジスタに書き込みたいアドレス番地を格納する．
3. 0x32 番地のレジスタに 1 を格納する．
4. 0x36 番地のレジスタに書き込みたいデータを格納する．
5. 0x48 番地のレジスタに 3843 を格納する．
6. 0x08 番地のレジスタに 0 を格納する．

以下の手順でレジスタに読み書きを行うことで，BRAM からデータを読み込む．

表 2　画像合成回路全体の論理合成結果.

	最大動作周波数 (MHz)	最小動作クロック周期 (ns)	LUT 数	スライス数
IS w/o SL	118.004	8.474	3293	1061
IS w/ SL	153.298	6.523	3069	1019

1. 0x08 番地のレジスタに 1 を格納する.
2. 0x56 番地のレジスタに読み込みたいアドレス番地を格納する.
3. 0x72 番地のレジスタに 1 を格納する.
4. 0x76 番地のレジスタからデータを読み込む命令を発行することで，BRAM から直接データを読み込む.
5. 0x84 番地に 1 を格納する.
6. 0x08 番地に 0 を格納する.

3.3.3 オフチップメモリ（BRAM）

オフチップの BRAM は，AXI_BRAM_32X4096 を用いる．メモリアクセス時のアドレス管理について説明する．PC から画像データを転送する際は，アドレス 0 番地から逐次的に格納していく．格納されるアドレスを ADD，座標を (x, y)，対象画像の x 軸の画素数を MAX_x とすると，1 枚目の画像データが格納される ADD は次式で示される．

$$\mathrm{ADD} = (y \times \mathrm{MAX_x} + x)$$

2 枚目の画像は ADD + 1 のアドレスから同様に格納される．全ての合成処理が終了し，BRAM へ色情報を格納する場合，使用されているアドレス ADD_{max} は，画素数 $MAX_x \times MAX_y$ の画像データを扱うとすると次式で表される．

$$\mathrm{ADD_{max}} = (\mathrm{MAX_y} \times \mathrm{MAX_x} + \mathrm{MAX_x}) \times 2$$

上式より，合成処理の完了した色情報は $\mathrm{ADD_{max}} + 1$ から順に格納することで，PC から逐次的に読み込み画像の座標情報を崩すことなく PC 上で画像を再構築可能である.

3.4 実験結果

セレクタ論理に適用させたアルファブレンド演算器を用いた画像合成回路を IS w/ SL，セレクタ論理を適用させないアルファブレンド演算器を用いた画像合成回路を IS w/o SL としたとき，これらを配置配線した結果を表 2 に示す．最大動作周波数は IS w/o SL が 118.004MHz，IS w/ SL が 153.298MHz，最小クロック周期は IS w/o SL が 8.474ns，IS w/ SL が 6.523ns であった．また，LUT 数は IS w/o SL が 3293 個，IS w/ SL が 3069 個であった.

本稿で実装した画像合成回路にセレクタ論理を適用させることで，およそ 23.1% の最大動作周波数の向上と，およそ 6.8% の LUT 数の削減を確認できた．PC から入力された画像を FPGA 上でアルファブレンド演算器を用いた画像合成を行った結果，PC 上で合成された画像を視認することができた.

4　おわりに

本稿では，PPM 形式画像を取り扱うアルファブレンド演算器を用いた画像合成回路を FPGA に実装し，その性能を評価した．FPGA に実装した結果，正しい画像を PC 上で視認することができ，画像合成回路中のアルファブレンド演算器をセレクタ論理に帰着させることで，回路処理時間の高速化，回路面積の削減を実現することができた.

謝辞

本研究の一部は総務省 SCOPE の支援による.

参考文献

[1] R. C. Gonzalez and R. E. Woods, *Digital Image Processing, 2nd Edition*, Prentice-Hall, 2002.
[2] 五十嵐啓太, 柳澤政生, 戸川望, “FPGA を対象としたセレクタ論理型アルファブレンディング回路の実装と評価, ” 電子情報通信学会総合大会, pp. 87, 2015.
[3] K. Igarashi, M. Yanagisawa, and N. Togawa, “FPGA implementation and evaluation of image scaling circuits using selector-logic-based bi-linear interpolation,” SASIMI2015, pp. 159–160, 2015.
[4] M. Maule, J. L. D. Comba, R. Torchelsen, and R. Bastos, “Transparency and anti-aliasing techniques for real-time renderig,” in *Proc. of the IEEE Graphics, Patterns and Images Tutorials*, pp. 50–59, 2012.
[5] T. Ma, M. Hempel, and H. Sharif, “A survey of energy-efficient compression and communication techniques for multimedia in resource constrained systems,” in *Proc. of the IEEE Trans. Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 963–972, 2013.
[6] M. J. Schulte, L. Marquette, and S. Krithivasan, E. G. Walters III, and J. Glossner, “Combined multiplication and sum of squares units,” in *Proc. of the IEEE International Conference on Application Specific Systems, Architectures, and Processors*, pp. 204–214, 2003.