Vol.2015-HPC-148 No.18
2015/3/2

FPGA タブレットによるモバイルアクセラレータ

塩谷丈史[†]　　成見哲[†,a)]

近年のプロセッサには動画デコードや暗号化回路など頻繁に利用される処理向けに専用回路がハードウェアで組み込まれていることも多い．しかし，すべての処理に対して専用回路を静的に用意することは現実的ではない．本研究では，モバイル端末での利用を前提として，FPGA を用いた Android タブレットを試作し，いくつかの数値演算専用アプリを実装した．Android アプリから FPGA 資源を Partial Reconfiguration（動的部分再構成）により利用するための API を実装し，CPU と演算性能を比較した．本研究で作成したアプリは，(1)アルゴン粒子の 2 次元分子動力学シミュレーション，(2)大きなデータに対する固定ビットパターンのマッチング，(3)外部 IO を使用した LED 発光回路の 3 つである．(1)の回路では CPU による処理速度に対して 340 倍，(2)の回路では 180 倍の高速化が行えた．また，(3)の回路により Android アプリから外付けハードウェアを簡単に操作できることが示せた．

1. はじめに

近年では，携帯端末の普及と利用方法の多様化により，アプリケーションから要求されるスペックや機能が増加している．これは専用の周辺回路やモジュール，チップを搭載することで問題はある程度解消されるが，携帯端末のプロセッサには電力や放熱の問題によりそれらの要求に応えるには限度がある．FPGA を用いてアプリケーション毎に回路を書き換えられれば，個別の専用回路が無くても専用回路並みの性能が期待できる．

PC を利用する場合には，PC-FPGA Hybrid Cluster [1] に見られるように CPU と FPGA を搭載したシステムを使用することで，アプリケーションに最適化された回路を使用した例がある．近年，ARM コアと FPGA を組み合わせた Programmable SoC と呼ばれる製品が出現しており，本研究ではこの方法を携帯端末に適用することを考えた．

本研究では，モバイル端末での利用を前提として，FPGA を用いた Android タブレットを試作し，いくつかの数値演算専用アプリを実装した(図 1)．Android から FPGA を利用する方法は Reconfigurable Android[2] で示されており，利用方法の改良と Zynq チップへの対応を行った．Java から FPGA を利用した例[3]はあるが，Linux で動作させている．本研究では，Android アプリから FPGA 資源を部分再構成[4]により利用するための API を実装し，CPU と演算性能を比較した．

図 1　試作した FPGA タブレット

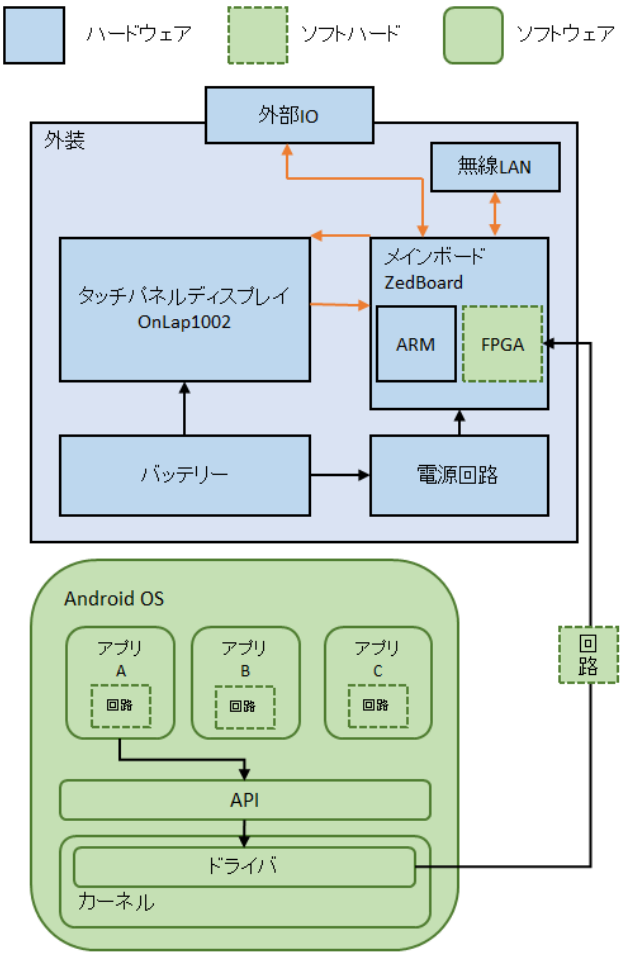

図 2　システムの構成

本研究で作成したアプリは，(1)アルゴン粒子の 2 次元分子動力学シミュレーション，(2)大きなデータに対する固定ビットパターンのマッチング，(3)外部 IO を使用した LED 発光回路の 3 つである．(1)の回路では CPU による処理速度に対して 340 倍，(2)の回路では 180 倍の高速化が行えた．また，(3)の回路により Android アプリから外付けハードウェアを簡単に操作できることが示せた．

†電気通信大学　情報理工学研究科
a)narumi@cs.uec.ac.jp

2. システム概要

本研究で開発したシステムは，デバイス，ソフトウェア，ソフトハードから成っている(図 2)．ソフトハードとは FPGA 等の再構成可能回路のことを意味する．

2.1 ハードウェア

ハードウェアは，制御部，電源部，表示部，入力部，無線部，外装部に分けられる．制御部は FPGA 及び ARM コアを搭載した ZedBoard[5]を使用し，Android OS が動作する．電源部はモバイルバッテリを電源とし 5V を表示部，入力部へ供給する．また，これを昇圧し，12V を制御部へ供給する．表示部は OS からの映像信号を 10 インチ液晶パネルで表示する．入力部は，タッチパネルからの信号を，制御部を介して OS へ送る．無線部は無線 LAN を使用して OS をネットワークへ接続する．外装部は，3D CAD を使用して，タブレット型にするためのフレームを設計し，3D プリンタによって出力した．フレームには前述の部品をすべて内蔵しており，デバイスを携帯することが可能である．3D プリンタには Replicator 2X[6] を使用した．

2.2 ソフトウェア

ソフトウェアは，OS 部，ドライバ部，API 部，アプリ部に分けられる．OS にはモバイル OS である Android を使用する．これはソースコードが容易に入手可能で改変もしやすいためである．Android は Zynq Android 4.1[7] を改変して使用した．改変した箇所は，いくつかの設定と画面出力，ドライバ関連である．

ドライバ部は，3 つのドライバを使用する．1 つ目は FPGA へ回路を書き込むドライバで，xilinx による xilinx_devcfg を使用し，メモリ確保の問題を回避するために改変を施した．2 つ目はアプリと回路でデータを転送するための DMA ドライバで，AXI DMA Engine の資料[8] を参考にして作成した．3 つ目はアプリからメモリを操作するためのドライバである．各種ドライバは連続でキャッシュレスなメモリ領域を

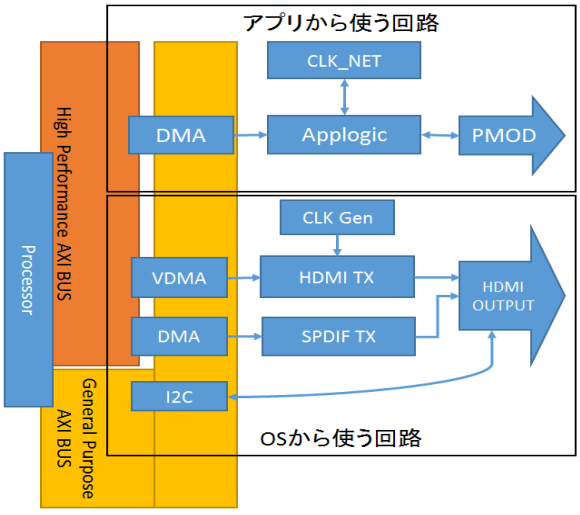

図 3　FPGA 上の回路

表 1 再構成領域で利用可能な資源

資源	個数
LUT	14400
FD_LD	28800
SLICEL	2100
SLICEM	1500
DSP48E1	120
RAMB18E1	60

必要とするため，OS から利用するメモリを制限することで OS 管理外のメモリを用意し，その領域を使用している．ZedBoard には 512MB のメモリが搭載されているが，本研究では 448MB に制限し，残りの 64MB を OS 非管理領域として使用している．この容量に設定した理由は，アドレスで 0x00000000～0x1BFFFFFF 及び，0x1C000000～0x1FFFFFFF と分割されてきりが良いためである．

API 部は，ドライバをアプリから利用するためのライブラリを用意する．ライブラリは C と Java を用いて記述され，Android NDK(Native Development Kit)と Android SDK によってコンパイルされる．C と Java での連帯には JNI(Java Native Interface)を用いる．C によるネイティブ部分でドライバを ioctl で呼び出すことで実現される．

アプリ部分は，実際のアプリケーションが記述され，API から FPGA へのアクセスを行うことができる．また，アプリは通常の Android アプリと同じように Android SDK を用いて Java の記述で作成可能であるが．書き込む回路データは PlanAhead を使用して作成する．回路データはアプリのリソースとして apk ファイルに内蔵され，アプリ起動時に API 経由で読み込まれ，FPGA へ書き込まれ，利用可能となる．部分再構成用の回路データの作成は xilinx wiki[9] を参考にして行った．

2.3 ソフトハード

ソフトハードは，FPGA 上の OS 使用部とアプリ使用部に分けられる(図 3)．OS 使用部には OS からの画面データを HDMI 信号へと変換し出力する回路及び，音声を出力するための回路がある．アプリ使用部には，データをやり取りするための DMA とグローバルクロックを使用するための回路，外部 IO を使用するための回路がある．さらに，アプリから回路を書き込む再構成領域がある(図 3 の Applogic)．再構成領域で利用可能な資源は表 1 の通りである．この領域を設定した理由は試行した中でもっとも資源を確保できる設定だったためである．

FPGA と ARM コアは汎用 AXI バスで接続されており，AXI DMA Engine などの AXI に対応したモジュールとの通信を行うことができる．FPGA とメモリコントローラは 4 チャネルの高速 AXI バスで接続されている．1 チャネルのバス幅は 64bit で，本研究では 100MHz で駆動させるので，最大転送速度は理論値で 800MB/s である．また，DMA Engine

と applogic は AXI Stream で接続される. このバスも幅 64bit, 100MHz 駆動で使用する. 本研究では, 画面表示, 音声出力, アプリ回路でそれぞれ高速 AXI バスを 1 チャネルずつ使用する.

PMOD とは，ZedBoard に実装された外部ピンのことである．PMOD は信号線 8 本と接地 2 本，電源 2 本がひとつのセットとなっている．本研究で使用する PMOD は 2 セットで，出力用と入力用としてアプリ回路に接続した.

3. アプリ

開発したシステム上で動作するアプリを 3 つ作成した．ここではそれぞれのアプリについて説明と設計，評価を行う.

3.1 2 次元分子動力学シミュレーション

アルゴン分子間の力の相互作用をそれぞれの分子で計算して分子の動きを表示するアプリである．表示の簡素化のため，分子の動き及び計算は 2 次元で行った．アプリを設計する上で必要な要素として，(1)各粒子のデータ，(2)GUI 上での表示，(3)描画と処理の仕方，(4)力の計算と適用方法の 4 つがある.

(1)各粒子のデータについて，各粒子が持つ情報は位置，速度，力の 3 つである．それぞれが(x, y)成分を持つので 6 つの変数が必要になるが，データを回路へ流す都合上，それぞれをひとつの配列に実装した．つまり，位置は

$$\mathrm{data}[\mathrm{i}\times 2] = \text{粒子}i\text{の x 座標}$$
$$\mathrm{data}[\mathrm{i}\times 2+1] = \text{粒子}i\text{の y 座標}$$
$$\mathrm{datav}[\mathrm{i}\times 2] = \text{粒子}i\text{の x 速度}$$
$$\mathrm{datav}[\mathrm{i}\times 2+1] = \text{粒子}i\text{の y 速度}$$
$$\mathrm{force}[\mathrm{i}\times 2] = \text{粒子}i\text{の受ける x 方向の力}$$
$$\mathrm{force}[\mathrm{i}\times 2+1] = \text{粒子}i\text{の受ける y 方向の力}$$

となる．これらはすべて 32 ビット浮動小数点数である．また，粒子データを BRAM へ格納する都合上 1,024 粒子を最大とした．これより大きい数の粒子数を扱う場合は BRAM を拡張する必要がある.

(2)GUI 上での表示は，各粒子の座標にスケール変数を掛けた座標に点を表示することで各粒子を表現する．また，各粒子の持つ速度を線で表すことで視覚的にわかりやすく工夫を行った.

(3)計算処理と描画処理ではこれを分けて処理する必要がある. それは, 計算処理と描画処理を逐次に実行した場合, 描画処理の時間だけ計算処理がとまってしまうことを意味しており，どれだけ計算処理が早くても 1 ステップの処理速度は描画処理に依存してしまうからである．二つの処理を分けて処理する方法として，マルチスレッディングを使用した．計算処理を別のスレッドで行うことで，描画処理の速度に依存せずにステップを進めることができる．計算処理が 1 ステップ終わるたびに描画可能フラグを立て，UI スレッドで行われている描画処理は処理可能なタイミング

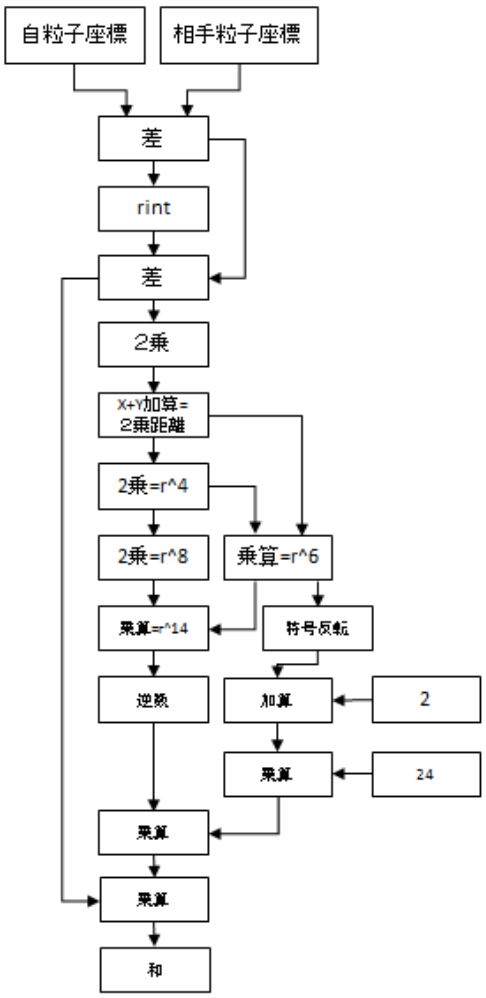

図 4　実装した MD パイプライン

で画面を描画する．これによって，描画処理は非同期処理となり，画面の更新タイミングにかかわらず計算処理は続行可能となる.

(4)力の計算を行う上で周期境界条件を設ける．この条件は，実際にデータとしてある範囲が周期的に無限に現れるということを前提とするものである．この条件下では，ある粒子へ相互作用を及ぼす粒子は周期的に現れるエリアの中から一番近いものとなる．また，力の計算は各粒子について，ほかの粒子から受ける力をすべて足し合わせることで行う．力の計算に用いる数式は式(1)のようになる.

$$\frac{d}{dr}\phi = 4(12r^{-14} - 6r^{-8}) = 24r^{-14}(2 - r^{6}) \quad (1)$$

を実際には計算している.
また，時間積分にはベルレ法[10] を用いる．ベルレ法による時間積分は式(2)によって行う. a_nはnステップ目の粒子に働く加速度であり，質量を 1 と仮定しているので力と同じである.

$$\begin{cases} \mathrm{v}_n = v_{n-1} + \frac{1}{2}(a_n + a_{n-1})\Delta t \\ x_n = x_{n-1} + v_{n-1}\Delta t + \frac{1}{2}a_{n-1}(\Delta t)^2 \end{cases} \quad (2)$$

表 2　力の計算にかかった時間

粒子数	FPGA(ms)	CPU(ms)	加速率
32	0.098	1.838	18.76
64	0.114	7.500	65.79
128	0.246	30.336	123.32
256	0.514	121.627	236.63
512	1.568	491.026	313.15
1024	5.701	1968.180	345.23

表 3　MD 回路の規模

資源	個数	使用率
LUT	7102	50%
FD_LD	4504	16%
SLICEL	1021	49%
SLICEM	755	51%
DSP48E1	24	20%
RAMB18E1	4	7%

表 4　ネイティブ実行での計測結果

粒子数	FPGA(ms)	CPU(ms)	加速率
32	0.058	1.031	17.82
64	0.077	4.434	57.82
128	0.149	18.454	123.75
256	0.429	75.761	176.65
512	1.506	305.182	202.89
1024	5.581	1229.164	220.25
1024 (4 本)	2.853	1229.164	430.83

回路での力の計算はパイプラインで行う(図 4)．このパイプラインによって前述の力の計算式を実装している．パイプラインでは浮動小数点数による計算を行う．パイプライン内で使用するそれぞれの計算器で必要なクロックは，加算器が 2 クロック，乗算器が 1 クロックである．ただし，FPGA に組み込まれているプリミティブな DSP のビット幅に合わせる都合上，25 ビット浮動小数点数として計算している．25 ビットの内訳は符号 1 ビット，指数部 8 ビット，仮数部 16 ビットである．また，パイプラインの本数は回路動作周波数の都合で 2 本とした．パイプライン中での除算は一度逆数に変換し，その後乗算を行うことで実現している．逆数の計算にはあらかじめ計算した値を表として保持しておき，この表を用いたニュートン法による計算で実現した．表のサイズはインデックス 4bit，値 14bit である．また，ニュートン法の適用回数を 1 回とし，逆数の計算式を式(3)のようにした．A は $1/A$ となる求めたい逆数の分母で，x_0は表から得られた初期値である．

$$x = 2x_0 - Ax_0^2 \qquad (3)$$

求めた力の総和は，通常の浮動小数点数の加算器では 2 クロックかかってしまい，累算することができない．そのため，累算器のみ固定小数点数に変換して加算を行った．

パイプラインへデータを流すためのステップは以下のように行う．

1．DMA エンジンからのデータを BRAM に格納する．
2．i = 0に初期化する．
3．それぞれのパイプラインにi + p番目の粒子データを配布する．(p はパイプラインの番号，p ≥ 0)
4．すべてのパイプラインに 0 から N までの粒子データを流す，(N は粒子数)
5．すべてのパイプラインがビジー状態でなくなるまで待つ．
6．それぞれのパイプラインから計算結果を回収して BRAM のi + p番目へ格納する．
7．i = i + Np を行い，i < Nならばステップ 3 へ, そうでなければ次のステップへ．(Np はパイプラインの数)
8．結果の BRAM から DMA エンジンへ結果を送る．
9．1 へ戻り待機する．

GUI の設計として，制御用のボタン，パラメータ用のテキストボックス，粒子を表示するエリア，情報を表示するエリアを配置した．

アプリ上で力の計算を CPU と FPGA による処理の二通りの方法で 100 回行い計測時間を平均した(表 2)．結果から FPGA による計算は CPU による計算よりも約 345 倍高速であることがわかった．処理回路の規模は表 3 のようになった．

CPU と FPGA の最大性能を見るために，CPU で，Java を使わず C によるネイティブプログラムで計算した場合と FPGA で部分再構成を使わなかった場合についても比較した(表 4). FPGA 全体を使用して回路を構成することができ，パイプライン 4 本が実装できた．ネイティブプログラムでは FPGA 回路への DMA にメモリコピーを必要としないため若干早くなっている．CPU での実行ではネイティブは Java に比べて 1.6 倍ほど高速であることがわかる．結果として FPGA の CPU に対する高速化率は 220 倍にとどまった．また, パイプライン 4 本での動作では 1,024 粒子時に約 430 倍の高速化ができた．

3.2 固定ビットパターンマッチング

固定ビットパターンが対象のデータ中にいくつ存在するかを探索しカウントするアプリである．パターンの比較にはビット単位での操作が必要であり CPU の苦手とする処理のひとつと考えられる．アプリを設計する上で必要な要素として，(1)探索対象のデータ，(2)探索ビットパターンおよびパターン長，(3)探索処理の 3 つがある．

(1) の探索対象のデータは int 型配列で要素数 1,048,576(32Mbit)として実装した．要素数をこの値よりも大きくしたところ制限値を超えてしまう旨のエラーが実行時に発生したためこの値を設定した．また，このデータをアプリ GUI 上で表示する場合に，テキストでは長さ制限によ

表 5　マッチング回路の規模

資源	個数	使用率
LUT	1617	12%
FD_LD	293	2%
SLICEL	236	12%
SLICEM	169	12%
DSP48E1	0	0%
RAMB18E1	0	0%

表 6　アプリ上での処理速度

	CPU	FPGA
処理時間(ms)	2528.15	14.08
相対時間	179.56	1

表 7　ネイティブでの処理速度

	CPU	FPGA
処理時間(ms)	650.03	5.38
相対時間	120.77	1

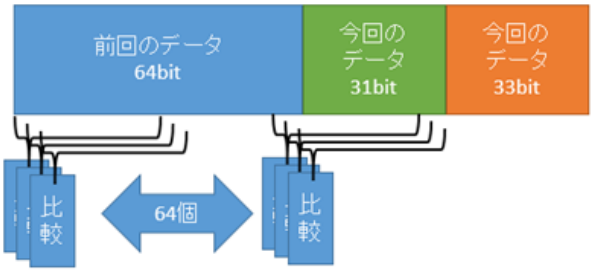

図 5　マッチング回路でのデータ構造

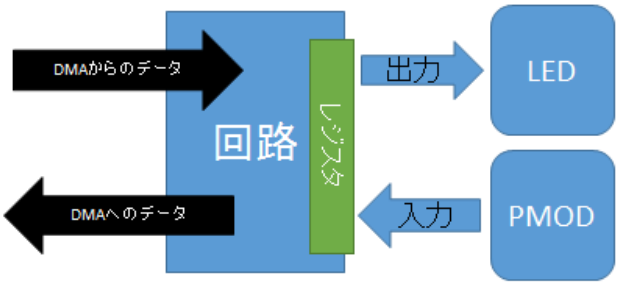

図 6　外部 IO 操作回路

り表示できなかったため，1,024×1,024 ピクセルのビットマップとしてデータを描画して GUI 上に表示した．

(2)の探索ビットパターンには int 型の変数として実装した．また，パターン長も int 型である．パターンデータは GUI 上のテキストボックスから 16 進数の形式で入力し，これを int 型へパースしてパターン長分のみを探索ビットパターンから使用する．

(3)の探索処理はデータの先頭から 32 ビットを 1 ビットずつずらして比較する方法を用いた．

ビットパターンの比較処理は DMA から送られてくるデータに対してストリーミングの形で処理を行う．DMA は 64 ビット単位でデータを送ってくるので，ひとつのデータブロックに対して 64 個の比較回路を並列に適用する．回路内部で保持するデータの構造は図 5 のようになっている．パターンデータは 32 ビットなので，前回送られてきたデータ 64 ビットと今回送られてきたデータのうち前半 31 ビットを用いて 95 ビットのデータとして比較に用いる．各比較回路は一致していれば 1，していなければ 0 を出力し，この出力を足し合わせることで一致数とする．この一致数はカウントのためのレジスタに足されて最終的なカウント値となる．DMA からのデータをすべて処理した後，カウントの値をメモリへ書き戻して回路は初期状態へ戻る．設計した回路の規模は表 5 のようになった．

GUI の設計としては，パターン長のスライダー，パターンのテキストボックス，CPU 実行用および FPGA 実行用のボタン，結果表示用のラベルを配置した．

アプリ上での処理時間をそれぞれ 100 回測定して平均を出した(表 6)．FPGA での処理は CPU に比べて約 180 倍高速であることがわかった．

同様にして，C によるネイティブプログラムでの計測も行った(表 7)．こちらでは FPGA での処理は CPU に比べて約 120 倍高速であることがわかった．FPGA の回路は表 6 と表 7 で同じであるが，アプリ上での実行とネイティブでの実行で，FPGA での処理時間が 2.6 倍違う結果となった．これはアプリから DMA を使用する場合に一度メモリ内部でコピーを行っており，データ量が大きいため時間がかかっていたものと考えられる．

3.3 外部 IO 操作

デバイス外部にある PMOD を使用した出力をソフトウェアから操作するアプリである．CPU は即時性が求められる処理に不向きであるため，外部との通信には専用の回路やプロセッサを用いたほうが安定すると考えられる．出力先には LED を取り付けて視覚的にわかりやすいようにした．アプリに必要な要素としては外部への出力をコントロールするための GUI のみであるので，トグルボタンを横に 8 つ並べた．動作はトグルボタンの状態変化をイベントで検知し，回路へ現在の状態情報を送信して出力を変化させるというものである．

回路では，出力と入力の 2 つの動作を行う(図 6)が，アプリで実装したのは出力のみである．出力の動作は，DMA から送られてきたデータを出力レジスタへ格納し待機状態へ戻るというものである．入力の動作は DMA からのデータ要求があった際に PMOD からの入力を送信し，待機状態へ戻るというものである．

実行した結果，実際にアプリから外部 IO が操作できていることを確認できた．

4. ソフトウェア無線への応用

ソフトウェア無線キット AD-FMCOMMS2-EBZ[11]が市販されており，ZedBoard に接続することが可能である．この製品はアンテナと AD コンバータ AD9361 を搭載しており，70MHz～6.0GHz の周波数帯に対して同調することができる．また，チャンネル幅は<200KHz～66MHz まで利用可能である. ZedBoard との接続は FMC コネクタを介して行う．

このデバイスで使用可能なデザインは ANALOG DEVICES[12] から入手可能である．本研究では cf_ad9361_zed を使用した．しかし，インプリメントに必要な IP コア(Intellectual Property Core)が不完全であり，そのままではインプリメントができなかった．問題となったのは，ad_dds_1 および ad_dcfilter_1 のモジュールである．ここで，ad_dds_1 については付属の ad_dds_1.xco ファイルを IP コアの設定ファイルとして読み込むことで生成できた．しかし，付属の ad_dcfilter_1.xco ファイルについては IP が Virtex6 向けのものであり，ZedBoard に搭載された Zynq7020 の Artix7 シリーズには対応していなかった．そのため，DSP48E 用の ad_dcfilter を用意し使用することで解決した．また，サンプルには chipscope 関連のモジュールが組み込まれており，これが原因のエラーが発生してしまうので chipscope も削除した．以上の作業により cf_ad9361_zed のインプリメントに成功した．

ソフトウェア無線を本研究で開発したデバイスへ装備することの意義と可能性を考える．アプリ回路からソフトウェア無線を利用可能となることで，アプリはアンテナと AD チップがサポートする任意の周波数において無線機能を利用可能となる．これによってアプリが実現できる機能は拡大し，様々な場所への応用が期待できる．また，アプリのみならず，本体の通信に利用する可能性も考えられる．昨今では様々な移動体通信網が発達しており，すべてのプロトコルに対して対応するには多くのコストがかかってしまう．そこで，必要なプロトコルに対応した無線モジュールを動的に再構成して利用することでコストを削減できるだけでなく，未知のプロトコルにも対応できる可能性がある．

5. まとめと今後の課題

本研究では，モバイルでの利用を考え，Android OS の動く FPGA タブレットを開発した．FPGA と CPU をひとつのチップにまとめたヘテロジニアスなデバイス上で OS を動作させ，アプリから FPGA の利用を可能にするためのドライバ及び API を整備・実装した．また，このプラットホーム上で実際にアプリを作成して性能を評価し，高速化が可能であることを示した．

さらに，タブレット形状にするために 3D プリンタによるフレームの出力や無線化，バッテリ駆動化を行った．携帯端末としての FPGA の利用可能性を示すことができた．実際に組み込むことはできなかったが，ソフトウェア無線による機能拡張の可能性について言及した．

ただし，いくつか問題点が残っている．Android OS の移植が不完全であるため，利用できる機能に制限がある点や，カーネルにおいて無線化を実現したために，Android OS から無線 LAN の設定ができないなどの問題がある．また，バッテリ駆動化に使用したモバイルバッテリは通電していない状態でもコネクタがつながっていれば昇圧回路に電流が流れ，電力を消費してしまう問題がある．現状では，アプリから回路へのデータ転送にはメモリ内でのデータコピーが発生してしまうため，性能の低下につながっており，アプリから直接メモリを操作するための解決策が必要である．アプリから使用する回路の設計はアプリ開発者が行う必要があり，これには HDL による記述が必須であるため，開発者の負担が大きい．このためこの負担を軽減する解決策が期待される．

参考文献

1) 尾崎亮，上嶋明，小畑正貴，”PC-FPGA 複合クラスタにおける部分再構成とその応用”，CPSY，コンピュータシステム, Vol.111, No.398,Pp.147-152, 2012-01-18
2) 小池恵介，太田涼，大島浩太，藤波香織，郡信幸，竹本正志，中條拓伯，“Android における Java アプリケーションの FPGA アクセラレーション”，情報処理学会論文誌, Vol.53, No.12, 2740-2751 .Dec. 2012.
3) Philippe Faes, Mark Chiristiaens and Dirk Stroobandt, “Interfacing Java with Reconfigurable Hardware”, Proceedings of the 15th ProRISC Workshop, Pp.385-390, 2004.
4) Partial Reconfiguration User Guide, Xilinx, http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_5/ug702.pdf.
5) ZedBoard, http://zedboard.org/product/zedboard.
6) Replicator 2X, Maker Bot, http://store.makerbot.com/replicator2x.
7) Zynq Android 4.1, Available from, https://github.com/cambridgehackers/zynq-android4/wiki.
8) LogiCORE IP AXI DMA v6.03a, http://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v6_03_a/pg021_axi_dma.pdf
9) Zynq 7000 Partial Reconfiguration Reference Design, Xilinx Wiki, http://www.wiki.xilinx.com/Zynq+7000+Partial+Reconfiguration+Reference+Design.
10) Loup Verlet, "Computer ‘Experiments’ on Classical Fluids. I. Thermodynamical Properties of Lennard-Jones Molecules”, Phys. Rev. Vol.159, Pp.98-103, July 1967.
11) AD-FMCOMMS2-EBZ: AD9361 RF Hardware evaluation Board, http://www.analog.com/jp/evaluation/eval-ad-fmcomms2/eb.html.
12) AD-FMCOMMS2-EBZ USER GUIDE, http://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz