平成29年度　**修　士　論　文**

修士論文題目

小規模開発下における
ZYNQ の開発環境の提案

指導教員　木村　誠聡　教授

神奈川工科大学　情報工学専攻

学籍番号　1685008

学生氏名　岡村　拓弥

提出日　平成30年　1月　27日　指導教員　印

受理日　平成30年　1月　29日　情報工学専攻主任　印

目次

1. まえがき

身の回りに存在する組み込みシステムは，一般的に CPU の命令によって周辺機器を制御し，様々な動作を実現している．しかしながら,組み込みシステムで使われる汎用性のある CPU は高速で処理が必要な信号処理や映像処理，数値計算処理等に対して適用することが難しく，専用の IC 又は LSI 等に処理を置き換えることによって処理速度を確保することが出来る．その一方で LSI，ASIC や ASSP などのロジックデバイスの設計・製造コストは高騰を続け，1 つのチップの開発に数億円以上の費用がかかる [1]，製品が市場に出るまでの時間（Time to Market）が長くなる，修正が効かない等の問題が種々存在する. その代替として FPGA と言われる実装状態で内部のロジックが書き換え可能なデバイスが注目されている．FPGA は LSI や ASIC 等に比べて容易に回路の修正を行うことが出来るため，専用回路の動作テスト，開発テスト等といったシミュレーションにもよく利用され，実装からテスト，修正と言った工程を FPGA で代用することが出来ることから, 開発コストの削減, 開発期間の短縮に貢献している．

前述の様に FPGA は論理回路で構成されるため，高速処理には向いているものの，複雑な条件分岐や反復処理などの制御が含まれる場合，論理回路のみで構成するのは回路規模が大きくなるため適切でない．よって複雑な制御が必要な場合，ソフトウェア処理も必要である．FPGA だけでは現状必要とされる様々な複雑な処理に対して，処理が追いつかない場合がある．そこで現在の組み込みシステムでは CPU と FPGA を組み合わせたシステムが必要であると考える．これにより CPU を組み込むことにより, 処理内容に汎用性を持たせることが可能であり, 幅広い要求に対応することが可能である. 例えば, 現在 FPGA と CPU が混在したものとして NOIS プロセッサーが搭載されている Altera 社の FPGA が存在し，同様に Xilinx 社では PowerPC が搭載された FPGA が存在する．これらは前述の様に論理回路を構成し，かつその論理回路を搭載している CPU で制御可能となっているため，高速動作かつ複雑な制御の 2 適性を有している.しかしながら，この CPU 搭載 FPGA を使うために，FPGA の回路設計技術と CPU によるソフトウェア技術の両方が必要であり，更に CPU から適切な回路を制御することも必要である．これらのことを初心者が学ぶには現時点では FPGA 設計,ソフトウェア開発等のツールが必要であり，適切な初心者用の学習ツールは存在しない．よって CPU が搭載された FPGA の動作がまず理解できる初心者用のツールが望まれる．

FPGA とは回路をプログラムによって何度も書き換え可能なデバイスであり, 回路であるため並列に実行することができる．並列で動作するため，速度が必要な部分だけを並列化することができ，効率的に組み上げることが可能である．FPGA は画像・映像処理や科学技術計算などに使われており，組み込みの分野で幅広く利用されている. そのため世間的に FPGA 技術者の需要が高まっている．また FPGA のみならず，条件によって処理内容を変更できる CPU の存在が必要である．しかしながら，ソフトウェア技術者が並列処理を感覚的に扱うのが困難であり，FPGA への導入に際し技術的な支援が望まれる. そこで FPGA と CPU を組み合わせた SoC（system on a chip）というものに着目する．しかしながら，既存の SoC の開発には CPU と FPGA の知識が必要であること，開発環境の切り替えが必要であり，同時に開発ができないこと，また既存の開発環境は大規模なものであり，開発に複雑な手順を要する．よって環境の構築に手間が掛かり，またツールを使いこなすまでに時間がかかるということもあり，現状初心者にとって着手しづらいものとなっている．そこで本論文では FPGA と CPU を簡易的に扱える実験環境，または教育用システムとして初心者用システムを検討し，統合開発環境を提案する．

本システムは FPGA 初心者，またはソフトウェアに触れたことのある技術者を対象とし，並列処理

を感覚的に扱えることを目指し，小規模開発および実験ができる環境を構築する．提案する初心者用システムの解決するべき点として，開発環境を切り替えることなく，CPU と FPGA の実験が可能であること．また，初心者がツールの扱いに困らないことである．以上の点を解決する方法としてインタプリタを利用した統合開発環境を提案する．

検討するシステムは，インタプリタを利用することでインターフェイスを統一し，CPU 側と FPGA 側の開発環境を切り替えることなく利用することができ，機能を制限することにより実行手順を簡略化し，複雑な手順を必要とせず，プログラミング初心者でも本ツールを扱うことができるものを考える．開発環境を SD カードに構築することで，PC 側に環境を用意する必要がなくなり，どの PC でも利用する事ができる．またファイルシステムを実装し，インタプリタで記述したコードをテキスト形式で SD カードに保存することにより，後に外部から編集できるものとする．

本システムの開発環境を実装する実機として，今後機能を追加することを考え，利用できるインターフェイスが豊富であることが望ましい．その中で，比較的安価で容易に入手できるものとして Xilinx 社が提供している SoC である zynq の zybo ボードを今回使用する．

本論文の構成は以下のとおりである．まず，2 章で FPGA と CPU，ハードウェアとソフトウェアの役割と違いについて，また SoC の特徴について述べ，現状抱える問題点について明らかにし，その問題点についての解決策を検討し提案する．3 章では提案したシステムの詳しい仕様や主な機能について説明する．4 章で実装したプログラムと回路について詳しく説明する．また本システムの使い方と手順について詳しく説明し，開発環境と開発機器について詳しく説明する．5 章で提案した本システムを実装し，行った検証についての目的や方法，結果と検証から発覚した今後の課題や問題点について述べる．最後に 6 章で本論文をまとめる．

2. CPU と FPGA の特徴と問題点

2.1. CPU と FPGA の特徴

CPU は 1 つの命令を小さな機能に分散させて処理をしており，複雑な処理を単純な処理の組み合わせにより実現している．組み込みシステムで利用されている小規模マイコンは，専用回路が小さく命令の種類が少ないのが特徴である[2]．専用回路の種類が少なく 1 つの命令を何度も使い回すため，多くの命令サイクルを必要とし，そのため処理速度が遅くなる．また処理速度を向上させるために高性能なマイコンを選択した場合，専用回路の種類と数は多くなるがその分チップサイズが大きくなるため単価が高騰してしまう．またチップサイズが大きくなるほど全ての回路を使いきれず，利用しない回路が無駄になってしまう．そこで特定のアプリケーション向けに必要な機能だけを搭載した ASIC や LSI 等と言った専用回路が存在する．ソフトウェアは処理の順番をプログラムによって指定するのに対し，ハードウェアは電気信号で実行され専用回路で組まれた順番に沿って処理するため，順番を指定する必要がなく複数の回路を同時に実行することができ，CPU より効率的に動作することが出来る．しかしながら，一般的に生産されている ASIC 等の専用回路の種類は膨大であり，また製造中止になる等のリスクがあり安定的に利用できるとは言えない．また ASIC 等の専用回路は自由な構成で設計することが出来る．そのため速度が必要な回路だけを実装する等無駄な回路を作ることなく効率的に組み上げることが出来るが，開発する場合近年プロセスの微細化に伴ってレイアウトやマスクの作成，冶具等の NRE コスト等で億単位の開発コストかかるといった問題点がある．また ASIC は後から回路を変更することができないため，設計ミスが見つかった場合修正することが不可能である．そこで近年 FPGA が注目されている．

FPGA(Field Programmable Gate Array)とは 1985 年に xilinx 社が初めて発表し商用化した，プログラムによって何度も書き換え可能なハードウェアデバイスである[2]．FPGA は回路であり電気信号によって即時実行され，並列で複数の回路を同時に実行することができる．そのため並列処理を得意とし，並列処理が必要とされる画像・映像処理やシミュレーション等に利用されている．FPGA はロジックセル，乗算器，RAM 等で構成されている．またルックアップテーブル(LUT)とフリップフロップを組み合わせて基本セルとしており，これらを組み合わせて自由に回路を構成することが出来，専用の回路を組むため無駄な回路に容量を取られることなく，効率的にリソースを利用できる．また直接回路を書き換えることが出来るため，ASIC や LSI などと言った専用回路の開発で必要とするコストがかからず，開発コストを大幅に抑えることが出来る．

2.2. CPU と FPGA の役割と違い

CPU は逐次処理であり，基本的には 1 つずつ順番に命令を実行する．CPU の中には複数種類の微細化された命令が実装されており，その命令の組み合わせによって処理を行っている．命令の組み合わせはプログラムによって変更されるため柔軟性があり，汎用性のあるシステムに利用される．一方 FPGA は反復処理や並列処理と言った処理が得意であり，CPU と違い同時に複数の処理を実行することが出来る．この性質を利用し汎用 IC では実現できない事，かなり複雑になってしまう場合等に FPGA に置き換え処理の効率化や処理速度の向上を図る．また年々FPGA の性能が向上し，低コスト FPGA でもある程度の規模の専用回路の実現が可能になっている．そのことから ASIC や ASSP 等の LSI から FPGA への乗り換えが増えてきており，FPGA の生産数量が増え，FPGA の単価が安くなってきている．また FPGA の特徴として回路の書き換えが可能であり，製品の量産後に回路の問題が顕在化しても

後から回路の変更が可能である.また回路規模によって FPGA は多くの種類が存在する．当然ながら従来の ASIC や ASSP 等の回路設計と FPGA の回路設計は基本的には同じベースであるものの，FPGA は手軽に扱える点，後から回路が修正できる点，多くの種類が存在する点等のメリットがあるため，ASIC や ASSP 等により幅広く扱われることになり,その為に FPGA を搭載する製品も多くなっている．これに伴い図 2.1 に示す様に開発環境(EDA)の売上も伸びている [3]．以上により FPGA が開発できる技術者及びその環境の搭載が求められていると考えられる．しかしながら，FPGA のみならず状況によって処理内容を変更できる CPU の存在も不可欠であると考える．この理由として，論理回路のみで構成される FPGA は高速で動作することは可能であるものの，分岐を使う複雑な制御には適さない．しかしながら，複数の回路を状況に応じて変更する場合が生じた時，CPU によるソフトウェア制御は柔軟に対応が可能となる．そこで本論文では，これらの点を踏まえ CPU と FPGA が混在した SoC に注目し，xilinx 社が提供している SoC のひとつである zynq に着目する．

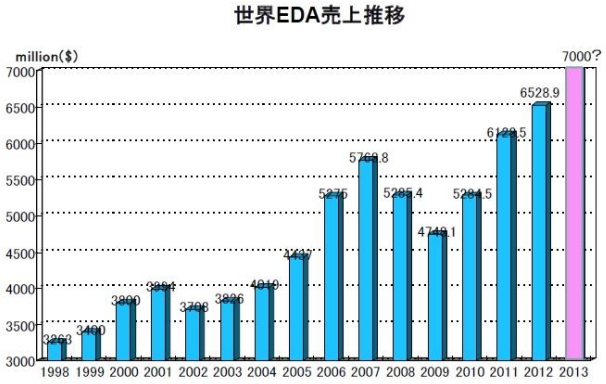


図 2.1　世界開発環境売上推移

2.3. SoC の問題点と解決策

SoC（system on a chip）とは FPGA と CPU を組み合わせたデバイスであり，FPGA と CPU それぞれ別々に開発を行い後に統合することが開発環境によってサポートされている．これによりソフトウェアとハードウェアの同時開発が容易になる．また 1 つのチップに収めたことにより，装置の小型化や製造コストの低減，配線の省略による高速化や部品点数の削減などのメリットがあげられる．しかしながら，SoC を開発する上での問題点として図 2.2 に示す様に CPU と FPGA の特性を理解する必要があり両方の知識が必要になること，FPGA と CPU の両方の開発環境が必要となるため大規模で複雑な開発環境になる等といった問題点がある．また図 2.3 に示す様にソフトウェアには順次処理が基本であり，処理が同時に進むことは考えられていない．しかしながら,ハードウェアは異なる処理が同時に進む場合が往々にして存在するため，ソフトウェアしか携わっていない技術者にとって複数の処理が同時に進むことは中々理解し難いものがある．よって並列処理という感覚が扱い辛いため，FPGA 初心者やソフトウェア技術者にとって着手しづらいという状況にある．そこで，並列処理を感覚的に扱うことの難しさ，また開発環境構築の手間，複雑な開発工程等といった問題点を解決するために，技術的な支援が必要であると考え，FPGA 導入に際しての初心者用システムを検討する．また本研究で利用するデバイスとして，zynq の zybo ボードを使用して開発を行う．

検討するシステムとして，着手・操作が簡単であること，並列処理を感覚的に扱えるようになること

を目的とし，FPGA 初心者に対して教育用もしくは個人での小規模開発用としての利用を考え，教育用または実験用として初心者用システムを検討し，統合開発環境を提案する．

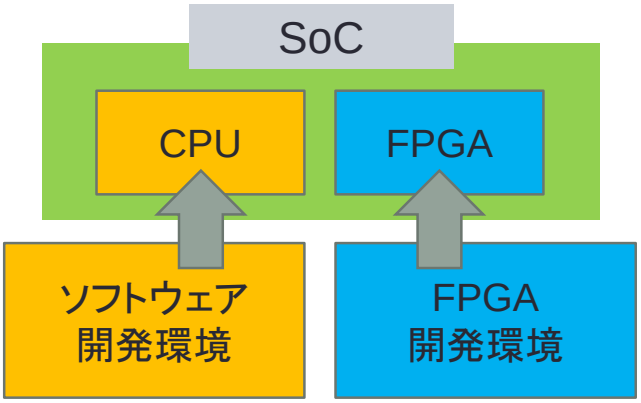
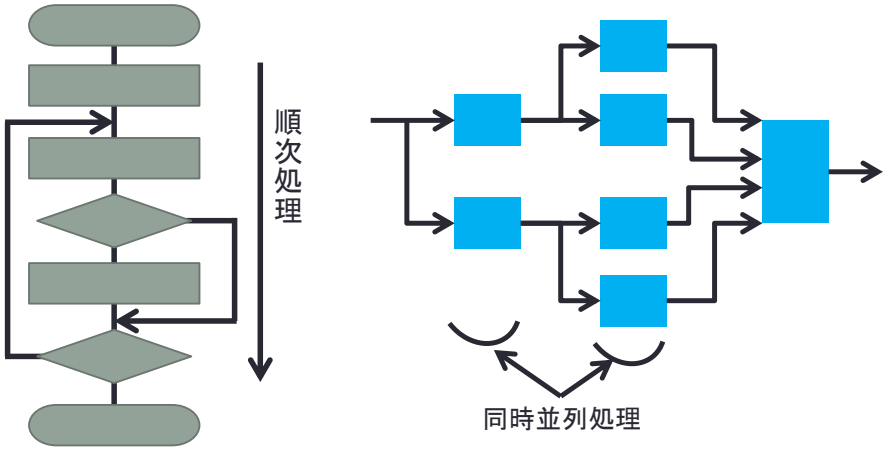


図 2.2　SoC 開発環境



(a)ソフトウェアによる構成例　　(b)　FPGA による回路構成例

図 2.3　ソフトウェアとハードウェアの違い

3. 提案する統合開発環境

本章は提案した統合開発環境の目的，システム概要，システム構成，仕様について詳しく説明する．

3.1. 提案する統合開発環境の目的

本システムは FPGA の初心者またはソフトウェアのプログラミングの経験がある人，またはない人を対象として，教育用または個人で FPGA を扱う際の導入として利用する事を想定し，手軽に利用できることにより導入しやすく，FPGA を感覚的に扱えるようになる事を目指す．また感覚に慣れるには試行錯誤を繰り返す必要があり，既存の FPGA の開発は，論理合成，配置配線，実装，ダウンロードと言った手順を踏みコーディングから実行まで時間がかかってしまう．そこで事前に回路を用意しコンパイル時間を省略することで，即座に利用者の考えを反映させることが出来る．

前章で上げた SoC の開発環境が複雑で大規模なものになると言った問題点を，統合開発環境として解決する．これはインタプリタを利用しインターフェイスを統一することで，開発環境の切り替えを必要としないものである．また単純なコマンドで利用できるインタプリタを用意し，操作に手間取ることなく利用できるものとする．また SD カードの中に開発環境を構築することによって，即座に開発環境の導入を済ませることが出来る．また既存の開発環境と提案する開発環境の比較を表 3.1 に示す．

表 3.1　開発環境の比較

開発環境	容量	開発言語	環境の切り替え
SDSoC	約 25G	別々	必要ない
HLS	約 20G	統一	必要
提案	なし	統一	必要ない

SDSoC と HLS は xilinx 社が提供している zynq の開発環境である．SDSoC は統合開発環境ではあるが，FPGA と CPU でそれぞれ別の開発言語を使用する．また，HLS は FPGA の回路を C 言語/C++言語で書けるようにした高位合成ツールである．そのためこのソフトでは FPGA 側の開発しかできず，CPU 側の開発をするためにはソフトウェアの切り替えが必要である．そこで提案する開発環境は PC 側ではなく SD カードに開発環境を構築するため，PC 側の容量は必要とせずインタプリタでインターフェイスを統一し CPU と FPGA の開発を同時に行うため，開発環境を切り替えることなく FPGA と CPU の両方の開発を同時に行うことができる．

3.2. システム概要

提案するシステムは，FPGA 初心者に向けた簡易的な実験および教育用として利用することを想定したシステムである．環境構築における手間を省き簡単に実験環境を整えることが出来る．

本システムは提案する開発環境を SD カードの中に導入し，PC と接続して電源を入れるだけで実験が可能である．このとき PC は端末として利用するだけなので，teraterm がインストールしてある PC ならどれでもよい．図 3.1 にシステム概要図を示す．

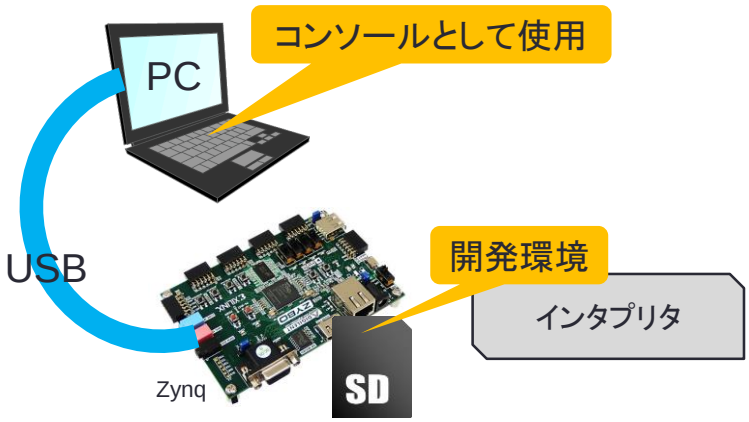


図 3.1　システム概要図

3.3. システム構成

本システムは microSD カードに開発環境を構築し，PC と zynq を USB で接続する．その後 PC からの入力でインタプリタにコマンドを送り CPU を制御する．そこからファイルシステムや FPGA を制御するシステムとなっている．図 3.2 に本研究のシステム構成図を示す．

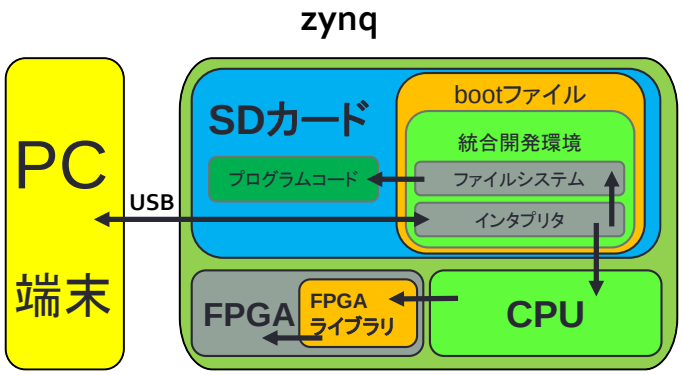


図 3.2　システム構成図

zynq は SD カード内の Boot ファイルから起動することができるため，Boot ファイルの中に開発環境を構築し，ファイルシステムとインタプリタを実装する．インタプリタは端末として使用する PC から，利用者がコマンドを入力することによって実行する．

本システムは端末からの命令をインタプリタが解釈し zynq を制御する．入力する命令によって，ファイルシステムにアクセスし SD カード内のテキストファイルの読み込みおよび書き込みができ，FPGA 回路のアドレスを指定することによって回路を切り替えることができる．また四則演算や　また回路は FPGA のコンパイル時間を省略するために複数実装しており，FPGA ライブラリから選択して実行して

切り替える．また既存の開発環境との工程差を図 3.3 に示す．

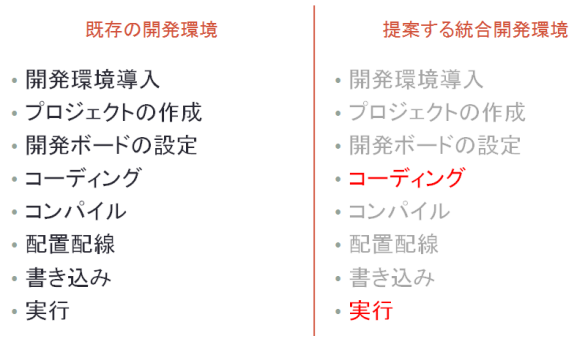


図 3.3　開発工程の差

既存の開発環境では開発に至るまでの手順として，開発環境の導入，プロジェクトの作成，開発ボードの設定，コーディング，コンパイル，配置配線，書き込み，実行というステップを踏む．これに対し，提案するシステムはコーディングして実行という手順で実行できる．

FPGA のコンパイル処理は，FPGA 開発の中でもかなりの時間を占める．そこで，あらかじめ回路を用意しておくことでコンパイル時間を省略し手軽に実行することができる．

また本システムの仕様上，実装してある回路でしか実験できないこと，また VHDL や VerilogHDL のコードを記述しての開発を行わないため FPGA の大規模開発には向かない．

3.4. 仕様

本節では提案する統合開発環境の仕様について詳しく説明する. 本研究では Xilinx 社が提供している SoC のひとつである Zynq の評価ボードの zybo を使用する. 理由として手軽で安価に入手できること, 入出力インターフェイスが豊富であること, 追加デバイスを必要とすることなく多くの実験ができることを基準に選定した. 入出力インターフェイスは以下の通りである.

リスト 3.1　zybo 入出力インターフェイス

- microSD カードスロット
- HDMI
- Ethernet
- VGA
- USB
- microUSB
- スライドスイッチ
- ボタン
- LED
- GPIO
- ライン入力
- マイク入力
- ヘッドホン出力

本システムで使うボードを図 3.4 に示す.
また利用する各部品を表 3.2 に示す. またに各部品に関して説明する.

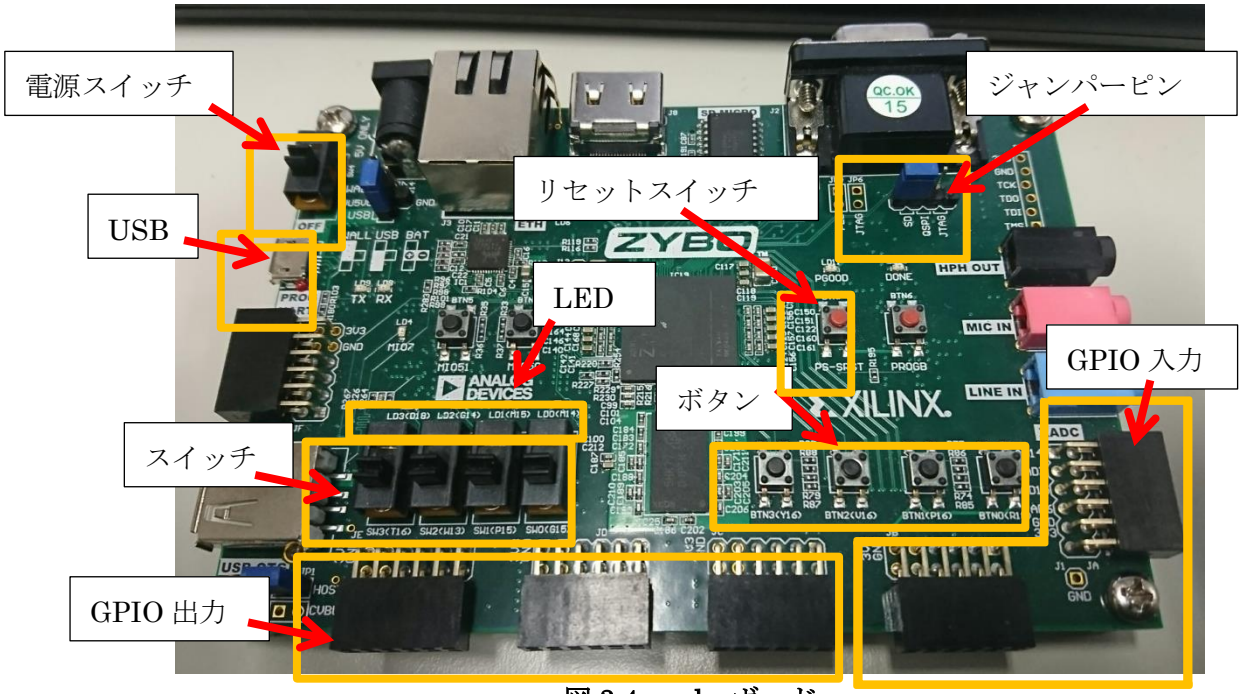


図 3.4　zybo ボード

表 3.2　各部品の説明

電源スイッチ	電源の ON・OFF が出来る
microUSB	PC と接続するためのインターフェイス．USB 給電で動く．
LED	4 つの LED
スイッチ	4 つのスライドスイッチ．
GPIO 出力	8bit の出力端子
GPIO 入力	8bit の入力端子
ボタン	4 つのプッシュスイッチ．
リセットスイッチ	システムを再起動する際に使用する
ジャンパーピン	zybo を起動する際にどこから読み込むかを設定する

・電源スイッチ

zybo ボードの電源スイッチ．PC と繋いでから電源を入れる．

・microUSB

USB バスパワーで駆動するため，PC と接続する場合別電源を必要としない．

・LED

4bit の LED．出力先として用意．

・スイッチ

４つのスライドスイッチ．入力用の回路として実装．

・GPIO 出力

8bit の GPIO 出力端子．3.3v の電源と GND の信号を各 2 本ずつ備える．

・GPIO 入力

8bit の GPIO 入力端子．3.3v の電源と GND の信号を各 2 本ずつ備える．

・ボタン

４つのプッシュスイッチ．入力用の回路として実装．

・リセットスイッチ

システムを再起動したい場合に使用する．

・ジャンパーピン

本システムを起動する場合，ジャンパーピンを SD に合わせる．

本システムを起動した際の動作手順は以下のようになっている．

リスト 3.2　起動時の動作フロー

１．Boot ファイルの起動．
２．統合開発環境の起動．
３．ファイルシステムにより SD カード内のテキストファイルを読み込む．
４．インタプリタコマンドの一覧を表示させる
５．読み込んだテキストファイルのコードをリスト化して表示させる．
６．インタプリタのコマンド入力による実行．

3.4.1. インタプリタコマンド

本節ではインタプリタコマンドの入力方法と各コマンドの仕様を詳しく説明する．

各実行画面は[4.実装内容]へ記述する．

本システムを起動すると SD カードの boot ファイルが読み込まれその中の開発環境が起動する．その際に SD カード内のテキストファイルが読み込まれ，記述されている開発コードが表示される．またテキストファイルがない場合自動的に生成される．また起動時には次の文字が表示される．

リスト 3.3　起動時コンソール画面

```
Script Start
 --- --- Interpreter Command List --- ---


A       program Add
I       program Insert
E       program 1 line Edit
del     program 1 line Delete
lst     program List
run     program Run
sav     program Save
lod     program Load
pls     program list in SD card
pco     program command list
ico     interpreter command list
fco     FPGA circuit list

※プログラムリスト


command ->
```

※ここで表示されるプログラムリストは，利用者が実際に記述したコードが表示される．

command ->の後に続けてインタプリタコマンドを入力して実行する．

全ての入力に対し，全角文字は使えないものとする．入力中の文字はコンソール上に表示されないため注意が必要．本システムで利用できるインタプリタコマンド一覧を以下に示す．

またプログラムを編集するたびに SD カードに保存される．

表 3.3　インタプリタコマンド一覧

	コマンド	機能
プログラム追加	A	1 番最後の行へプログラムを追加する
プログラム挿入	I	指定した行にプログラムを追加する
プログラム編集	E	指定した行のプログラムを編集する

プログラム削除	del	指定した行のプログラムを削除する
プログラムコード表示	lst	編集中のプログラムのコードを一覧にして表示する
プログラム実行	run	編集中のプログラムを実行する
プログラム保存	sav	編集中のプログラムを SD カードへ別名で保存する
プログラム読み込み	lod	指定したプログラムを SD カードから読み込む
プログラム一覧表示	pls	SD カード内のプログラムをリスト化して表示する
インタプリタコマンド一覧	ico	インタプリタコマンドを一覧にして表示する
回路コマンド一覧	fco	回路コマンドを一覧にして表示する
プログラムコマンド一覧	pco	プログラムコマンドを一覧にして表示する

プログラムの実行にはプログラムを追加・編集しただけでは実行されず，[run]コマンドを実行する必要がある．< program command >はプログラムコマンドのことを指し詳細は[3.4.2 プログラムコマンド]で説明する．

・インタプリタコマンド詳細

プログラム追加	A	1 番最後の行へプログラムを追加する

編集中のプログラムの最後の行へプログラムを追加する．
コマンドは大文字の「A」とする．小文字の「a」ではエラーとなる．

```
command  ->  A
< program command >
```

プログラム挿入	I	指定した行にプログラムを追加する

指定した行以下のプログラムを 1 行下へずらす．その後指定した行へプログラムを挿入する．
コマンドは大文字の「I」とする．小文字の「i」ではエラーとなる．

```
command -> I
line number -> <挿入したい行を入力>
input command -> < program command >
```

プログラム編集	E	指定した行のプログラムを編集する

指定した行のプログラムを編集する．
コマンドは大文字の「E」とする．小文字の「e」ではエラーとなる．
Line number で指定した行のプログラムを書き換える．書き換える際に書き換える前のプログラムを表示させる．

```
command -> E
line number -> <編集したい行を入力>
修正前のプログラム  -> < program command >
```

プログラム削除	del	指定した行のプログラムを削除する

指定した行のプログラムを削除する．その後その行以下のプログラムを 1 行上へ動かす．

コマンドは小文字で「del」とする．

```
command -> del
line number -> <削除したい行を入力>
```

プログラムコード表示	lst	編集中のプログラムのコードを表示する

編集中のプログラムのコードを一覧にして表示する．

コマンドは小文字の「lst」とする．

```
command -> lst
```

プログラム実行	run	編集中のプログラムを実行する

編集中のプログラムを実行する．

コマンドは小文字の「run」とする．

```
command -> run
```

プログラムの先頭からプログラムコマンド<end>まで，もしくは最後まで実行して停止する．実行する際に 1 行ずつ実行するコードを表示させ，各コードに対しての結果を表示できるものは表示させる．このコマンドを実行しない限りコマンドが実行されることはない．

プログラム保存	sav	編集中のプログラムを別名で保存する

編集中のプログラムを別名で保存する．ファイル名は半角英数 4 文字で指定する．

1 番最初のプログラムはデフォルトで p001.txt と固定する．また起動時に読み込まれるプログラムも p001.txt．コマンドは小文字の「sav」とする．

```
command -> sav
```

プログラム読み込み	lod	指定したプログラムを読み込む

SD カード内にあるテキストファイルを指定し読み込む．

SD カード内にあるテキストファイルは[pls]コマンドで確認することが出来る．

コマンドは小文字の「lod」とする．

```
command -> lod
< テキストファイル指定 >
```

プログラムリスト表示	pls	SD カード内のテキストファイルをリスト化して表示する

lod コマンドで読み出すファイルを指定するためにここでファイル一覧を表示する．

SD カード内にあるテキストファイルを読み込み，リスト化して表示する．

コマンドは小文字の「pls」とする．

```
command -> pls
```

インタプリタコマンド一覧	ico	インタプリタコマンドを一覧にして表示する

本システムで利用できるインタプリタコマンドの一覧を表示する．

コマンドは小文字の「ico」とする．

```
command -> ico
```

回路コマンド一覧	fco	回路コマンドを一覧にして表示する

本システムで利用可能な回路コマンド一覧を表示する．回路コマンドについての詳しい利用方法は回路コマンド一覧の方に記述する．

コマンドは小文字の「fco」とする．

```
command -> fco
```

プログラムコマンド一覧	pco	プログラムコマンドを一覧にして表示する

本システムで利用可能なプログラムコマンド一覧を表示する．

コマンドは小文字の「pco」とする．

```
command -> pco
```

3.4.2. プログラムコマンド

本節ではインタプリタコマンドの[A][I][E]で入力するプログラムコマンドについて詳しく説明する．またそのときに使用するプログラムコマンド一覧を表 3 に示す．各プログラムコマンドの実行結果については，[4.実装内容]に記述する．

表 3.4　プログラムコマンド一覧

	コマンド	機能
入力	scan	キーボードからの入力
計算	cal	四則演算
分岐	if	条件による分岐
ラベル	labl	ラベル付け
ジャンプ	goto	該当するラベルの位置へ飛ぶ
コメント	rem	コメント
表示	prt	パラメータをコンソール上に表示する
回路選択	F	回路を選択する
終了	end	インタプリタを終了する

また変数として，int 型と float 型各 10 個ずつ用意した．その一覧を下の表 3.3 に示す．

表 3.5　実装されている変数

	変数名	個数	番号
int 型	val	10 個	0~9
float 型	fval	10 個	0~9

int 型は整数，float 型は実数の入力に対応している．

該当する変数を指定する場合，<val_0>のように変数名と番号の間に _（アンダーバー）で区切る．

番号は 0~9 までを指定することが出来る．全て初期値は 0 で設定されている．

定数を指定する場合[#＋数字]で記述する．ex. [#65]

プログラムコマンドを入力する際，以下の仕様に従って入力する．

< program command > [プログラムコマンド][パラメータ 1][パラメータ 2][パラメータ 3]

入力するプログラムコマンドは表 3 を参考にする．

コマンドとパラメータの間は半角スペースで区切って入力する．

基本的には上記の構造になっているが，パラメータの数は各コマンドによって変わるため，利用する場合はコマンド毎に詳細を確認する．

入力	scan	キーボードから入力する

動作概要：キーボードから入力した値を指定した変数へ格納する．

コマンド構造 [scan][パラメータ 1][x][x]

パラメータ 1 に格納する先の変数と番号を入力する．
指定する変数は，あらかじめ用意されているものを指定する．使用できる変数は表 3.3 を参照する．
パラメータ 2 とパラメータ 3 は使用しないため入力しない．
[run]コマンドを実行した際にパラメータ 1 で指定した変数の型に合わせてキーボードから値を入力する．

計算	cal	四則演算

動作概要：指定したパラメータどうしの計算を行う．

コマンド [cal][パラメータ 1][演算子][パラメータ 3]

パラメータ 1 に変数を，パラメータ 3 には変数または定数を指定する．
指定する変数は，あらかじめ用意されているものを指定する．使用できる変数は表 3.3 を参照する．
また演算結果をパラメータ 1 の変数へ格納するため，パラメータ 1 は変数を指定する必要がある．
演算子は以下の 5 つを使用する．

・[+]演算子
パラメータ 1 で指定した変数に，パラメータ 3 で指定した変数または定数を加算する．
その結果をパラメータ 1 の変数へ格納する．
・[-]演算子
パラメータ 1 で指定した変数に，パラメータ 3 で指定した変数または定数を減算する．
その結果をパラメータ 1 の変数へ格納する．
・[*]演算子
パラメータ 1 で指定した変数に，パラメータ 3 で指定した変数または定数をかける．
その結果をパラメータ 1 の変数へ格納する．
・[/]演算子
パラメータ 1 で指定した変数に，パラメータ 3 で指定した変数または定数で割る．
その結果をパラメータ 1 の変数へ格納する．
・[<]演算子
パラメータ 1 で指定した変数へパラメータ 3 で指定した変数および定数を格納する．
[scan]と違い[run]コマンド実行時に自動的に格納されるため，キーボードから入力する必要がない．

分岐	if	条件による分岐

動作概要：2 つのパラメータの比較を演算子に従って行い，その真偽によって処理を変更する．

コマンド
[if][パラメータ 1][演算子][パラメータ 3]

パラメータ 1 とパラメータ 3 に変数または定数を指定する．
指定する変数は，あらかじめ用意されているものを指定する．使用できる変数は表 3.3 を参照する．
演算子は以下の 5 つを使用できる．

・[<=]演算子
パラメータ 1 の値がパラメータ 3 の値以下の場合[真]となり，より大きい場合[偽]となる．
・[<]演算子
パラメータ 1 の値がパラメータ 3 の値より小さい場合[真]となり，以上の場合[偽]となる．
・[--]演算子
パラメータ 1 の値がパラメータ 3 の値と同じ場合[真]となり，違う場合[偽]となる．
・[>]演算子
パラメータ 1 の値がパラメータ 3 の値より大きい場合[真]となり，以下の場合[偽]となる．
・[>=]演算子
パラメータ 1 の値がパラメータ 3 の値以上の場合[真]となり，より小さい場合[偽]となる．

各演算子でパラメータ 1 とパラメータ 3 を比較した結果

[真]の場合	[if]コマンドの次の行が実行され，その後 1 行飛ばして実行される．
[偽]の場合	[if]コマンドの次の行を飛ばして 2 行目が実行される．それ以降は順番に実行される．

ラベル	labl	その行へ名前を付ける．

動作概要：[goto]コマンドで移動する先を指定する．

コマンド
[labl][パラメータ 1][x][x]

パラメータ 1 には文字列を入力する．
パラメータ 2 とパラメータ 3 は使用しないため入力しない．
ここで指定した[labl]の位置を[goto]コマンド実行時に探しだし，指定した[labl]の処理へ移動する．
その後[labl]位置から順番に実行される．

ジャンプ	goto	[labl]で指定した行へ移動する. 繰り返し回数を指定する.

動作概要：指定した場所へ移動する.

コマンド [goto][パラメータ 1][パラメータ 2][x]

パラメータ 1 で[labl]で指定したパラメータを指定し，移動先を指定する.

パラメータ 2 で繰り返し回数を指定する.

パラメータ 3 は使用しないため入力しない.

コメント	rem	実行されないコメントを記述する.

動作概要：プログラム上でコメント文を残す.

コマンド [rem][パラメータ 1][x][x]

パラメータ 1 に文字列を入力する．頭文字に指定はない.

パラメータ 2 とパラメータ 3 は使用しないため入力しない.

全角文字は利用できない.

表示	prt	パラメータまたは文字列をコンソール上に出力する

動作概要：指定した変数または文字列を表示する.

コマンド [prt][パラメータ 1][x][x]

パラメータ 1 に表示したい文字列もしくは変数名を指定する.

パラメータ 2 とパラメータ 3 は使用しないため入力しない.

変数名を指定する場合はそのまま指定できるが，文字列を表示させたい場合は頭文字に[“]をつける必要がある.

変数を表示させたい場合 [prt][val_0][][]　　val_0 に格納されている変数を表示させる. 文字列を表示させたい場合 [prt][“zynq][][]　　[“zynq]と表示させる.

回路選択	F	回路一覧から回路を選択し，回路を切り替え実行する.

動作概要：

コマンド [F][パラメータ 1][パラメータ 2][パラメータ 3]

パラメータ 1 には使用する回路を指定する.

パラメータ 2 とパラメータ 3 については各回路コマンドにより異なる.
使用できる回路は，[3.4.3 実装回路一覧]で説明する.

終了	end	プログラムを終了する.

動作概要
[run]コマンドを終了させる．その後インタプリタコマンド編集画面へ戻る.

コマンド
[end][x][x][x]

パラメータ 1 とパラメータ 2 とパラメータ 3 は使用しないため入力しない.

3.4.3. 実装回路一覧

本節では実装されている回路コマンドについて詳しく説明する．

各コマンドの実行結果は[4.実装内容]に記載する．

表 3.6　回路選択コマンド一覧

回路	回路コマンド
and 回路	and_番号
or 回路	or_番号
nand 回路	nand_番号
nor 回路	nor_番号
xor 回路	xor_番号
not 回路	not_番号
LED 出力	led
button 入力	btn
switch 入力	sw
GPIO　in	gi_番号
GPIO　out	go_番号
７セグメントドライバ	ssd
カウント	co

インタプリタコマンドで[F]を入力した後のコマンドはこちらを参照する．

コマンドを入力する場合，以下の仕様に従って入力する．

コマンド
[F][回路][パラメータ 1][パラメータ 2]

[回路]に回路コマンドを入力する．利用できる回路コマンドは表 3.6 を参照する．

またコマンド間は半角スペースで区切る．

コマンドの構造として[F][出力][入力 1][入力 2]とする．

出力回路コマンド一覧をリスト 3.4 に示す．

リスト 3.4　出力回路コマンド

[and][or][nand][nor][xor][not][led][go][ssd][co]

入力回路コマンドをリスト 3.5 に示す．

リスト 3.5　入力回路コマンド

[and][or][nand][nor][xor][not][gi][sw][btn][ssd][co]

を使う．入力 2 に関しては出力に論理回路を指定した場合に使用する．

and 回路	and	x10

and 回路を 10 個用意.

[F][and_番号][パラメータ 1][パラメータ 2]

番号は 0~9 を指定する．パラメータ 1 とパラメータ 2 は数値を入力する.

数値は 8bit の値を 16 進数表記で入力する．頭文字に[0x]を記述する．ex.[0x09]

パラメータ 1 とパラメータ 2 の論理演算結果が指定した論理回路番号へ格納される.

or 回路	or	x10

or 回路を 10 個用意.

[F][or_番号][パラメータ 1][パラメータ 2]

番号は 0~9 を指定する．パラメータ 1 とパラメータ 2 は数値を入力する.

数値は 8bit の値を 16 進数表記で入力する．頭文字に[0x]を記述する．ex.[0x09]

パラメータ 1 とパラメータ 2 の論理演算結果が指定した論理回路番号へ格納される.

nand 回路	nand	x10

nand 回路を 10 個用意.

[F][nand_番号][パラメータ 1][パラメータ 2]

番号は 0~9 を指定する．パラメータ 1 とパラメータ 2 は数値を入力する.

数値は 8bit の値を 16 進数表記で入力する．頭文字に[0x]を記述する．ex.[0x09]

パラメータ 1 とパラメータ 2 の論理演算結果が指定した論理回路番号へ格納される.

nor 回路	nor	x10

nor 回路を 10 個用意.

[F][nor_番号][パラメータ 1][パラメータ 2]

番号は 0~9 を指定する．パラメータ 1 とパラメータ 2 は数値を入力する.

数値は 8bit の値を 16 進数表記で入力する．頭文字に[0x]を記述する．ex.[0x09]

パラメータ 1 とパラメータ 2 の論理演算結果が指定した論理回路番号へ格納される.

xor 回路	xor	x10

xor 回路を 10 個用意.

[F][xor_番号][パラメータ 1][パラメータ 2]

番号は 0~9 を指定する．パラメータ 1 とパラメータ 2 は数値を入力する.

数値は 8bit の値を 16 進数表記で入力する．頭文字に[0x]を記述する．ex.[0x09]

パラメータ 1 とパラメータ 2 の論理演算結果が指定した論理回路番号へ格納される.

not 回路	not	x10

not 回路を 10 個用意.

[F][not_番号][パラメータ 1][x]

番号は 0~9 を指定する．パラメータ 1 に数値を入力する.

数値は 8bit の値を 16 進数表記で入力する．頭文字に[0x]を記述する．ex.[0x09]

パラメータ 1 の論理演算結果が指定した論理回路番号へ格納される.

LED 出力	led	x1

led に入力した値を 4bit で出力する.

[F][led][パラメータ 1][x]

パラメータ 1 に数値を入力する．数値は 4bit の値を 16 進数表記で入力する.

頭文字に[0x]を記述する．ex.[0x9]

または入力として[sw][btn]を利用できる.

switch 入力	sw	x1

switch 回路

出力先としてではなく入力として利用するため[回路]コマンドとしては利用できない.

他の回路コマンドの入力として利用する.

button 入力	btn	x1

button 回路

出力先としてではなく入力として利用するため[回路]コマンドとしては利用できない.

他の回路コマンドの入力として利用する.

GPIO　入力	gi	x2

Pmod に 8bit までの外部入力が利用できる.

番号は 0~1 が選択できる．利用できるポートは JA と JB である.

また，各ポートのピンは以下のような順番で格納される.

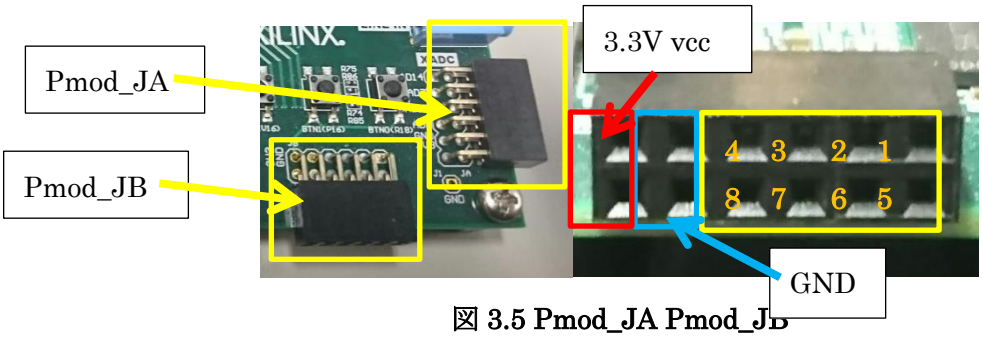


図 3.5 Pmod_JA Pmod_JB

[F][gi_番号][x][x]

パラメータ 1 とパラメータ 2 には何も指定しない.

外部入力した 8bit のデータが[gi_番号]のアドレスに格納される.

このデータを GPIO の出力に利用できる.

GPIO　出力	go	x3

Pmod から 8bit 出力する.

番号は 0~2 が選択できる．利用できるポートは JC と JD と JE である.

以下のようなピン配置で出力する.

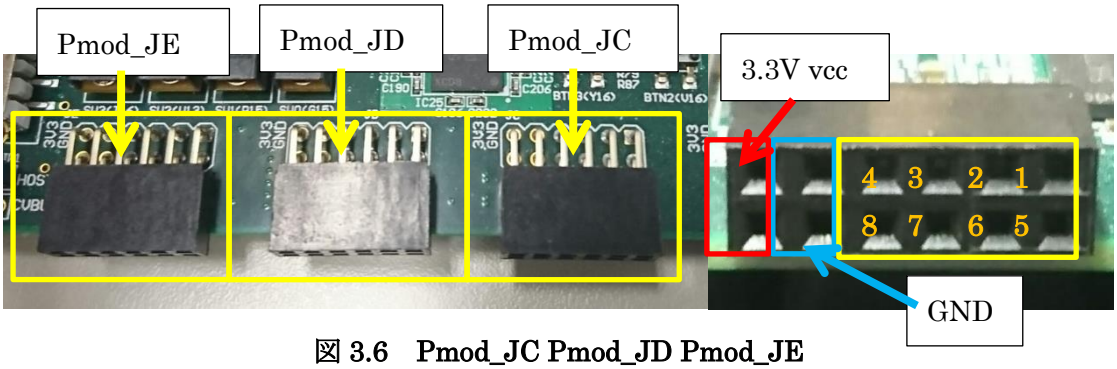


図 3.6　Pmod_JC Pmod_JD Pmod_JE

[F][go_番号][パラメータ 1][x]

パラメータ 1 に 8bit の値を 16 進数で入力する．先頭に[0x]と入力する．ex.[0xff]

パラメータ 2 には何も入力しない.

7 segment driver	ssd	x1

入力したパラメータに対応した値を，7 セグメント用のドライバから出力する.

[F][ssd][パラメータ 1][x]

パラメータ 1 で 4bit の値を入力する.

頭文字に[0x]として 0~F までの 16 進数で入力する.

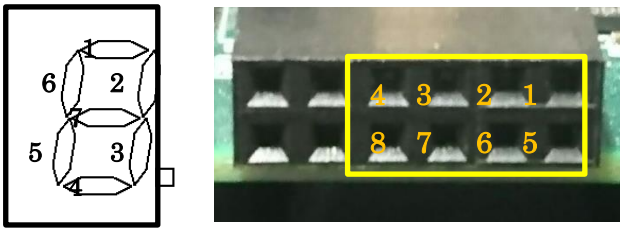


図 3.7　7segment driver の対応ピン配置

カウンタ回路	co	x1

カウンタ回路

[F][co][パラメータ 1][x]

1 カウント周期が 0.1 秒 0.5 秒 1 秒 2 秒と用意する.

パラメータ 1 に周期を設定する．以下の表に従って入力する.

規定された数値以外の物が入力された場合は 5 秒になるよう設定した.

表 3.7　カウント周期

秒	パラメータ 1 コマンド
0.1	1
0.5	2
1	3
2	4

4. 実装内容

本章では実装したプログラムと回路について説明する．CPU 側ではインタプリタとファイルシステムを実装し，FPGA 側では論理回路，7seg ドライバ，カウンタ回路，加算回路を実装した．

Zynq は microSD カードから起動できるため，microSD カードに Boot ファイルを用意し，その中に開発環境を構築する．またファイルシステムを実装することで，インタプリタで編集したコードを保存，読み込みができるようにする．テキスト形式で保存することにより，外部からの編集にも対応させる．FPGA 側の開発はあらかじめ回路を複数用意しておき状況によって切り替えることとする．このため機能は制限されるが，FPGA 側のコードをコンパイルする必要がなく，開発時間を短縮できる．

4.1. CPU 側

本システムはインタプリタによりコマンドを読み込み実行する．また，その時に記述したコードをファイルシステムにより保存することが可能となっている．ここではそのインタプリタとファイルシステムについて詳しく説明する．

4.1.1. インタプリタ

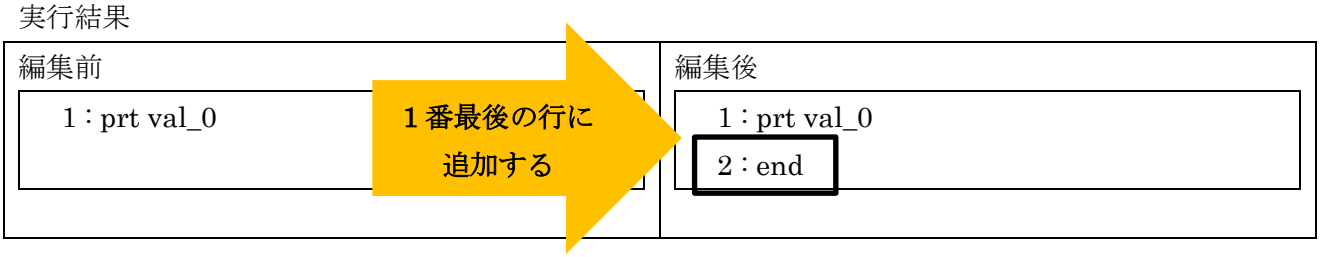
インタプリタの各コマンドについての詳細は[3.4.1 インタプリタコマンド]で説明した．ここでは各インタプリタコマンドの入力例と実行結果について示す．

・A コマンド

入力例

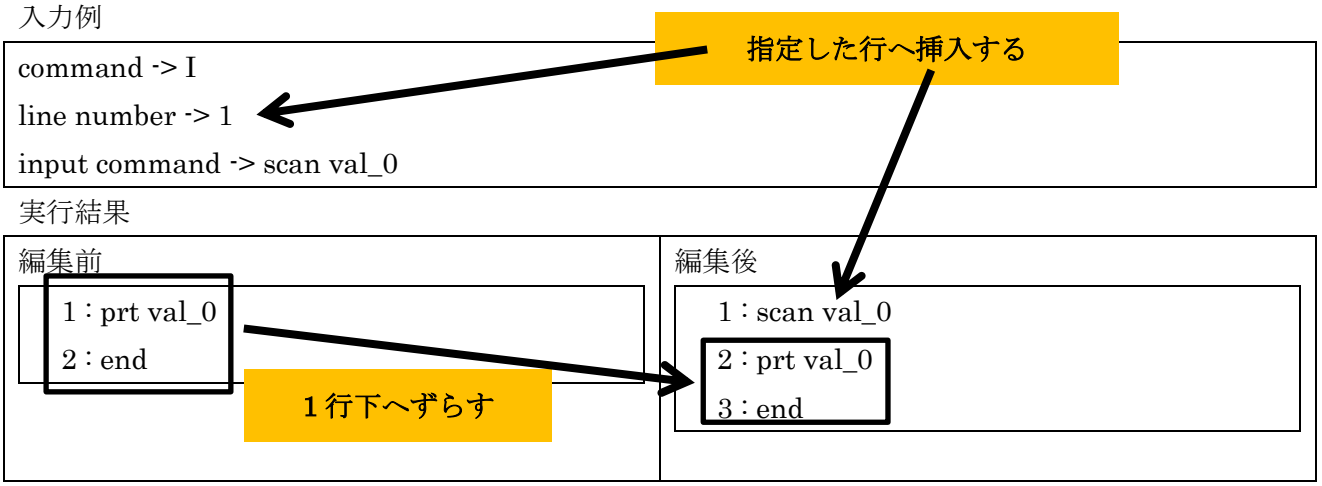
```
command -> A
Add
input command -> end
```

実行結果



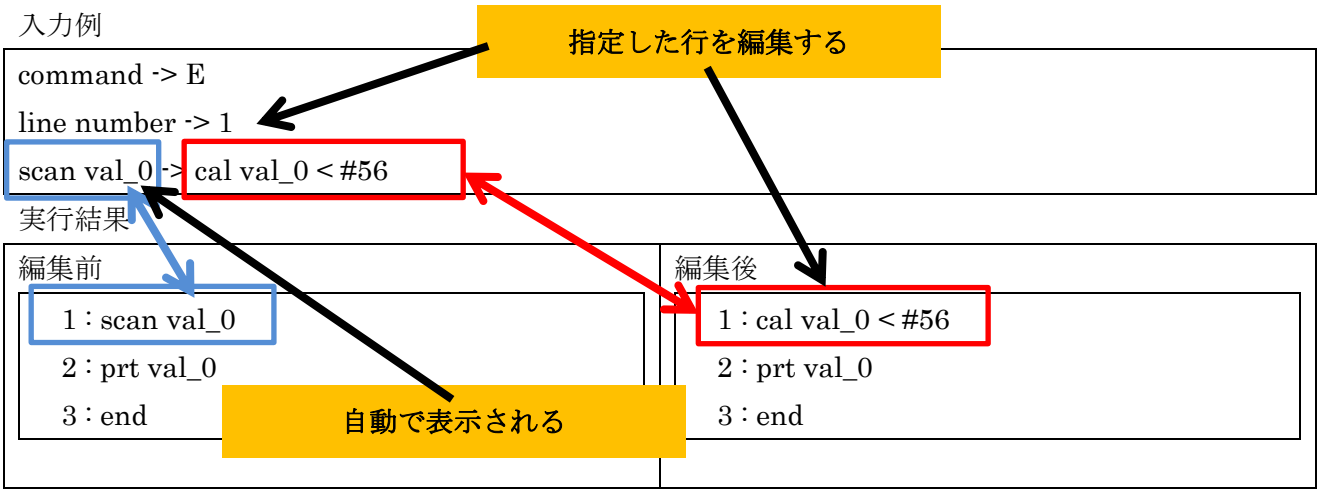
・I コマンド

入力例



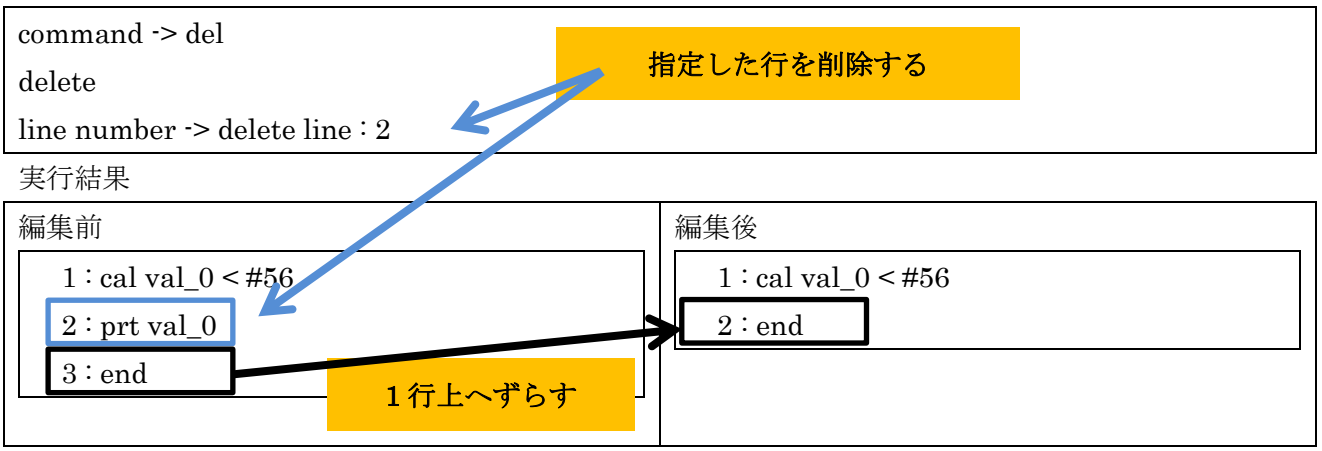
・E コマンド

入力例



・del コマンド

入力例



・lst コマンド

入力例

```
command -> lst
```

実行結果

```
 -- program list display --

  1 : cal val_0 < #56
  2 : end
```

現在編集中のプログラムコードが表示される

・run コマンド

入力例

```
command -> run
```

実行結果

```
 -- program run --

000 : [cal][val_0][<][#56]
val[0] = 56
001 : [end][][][]
 -- end --
```

実行されるコマンド

実行結果

１行ずつ実行される

・pls コマンド

入力例

```
command -> pls
```

実行結果

```
-> p001
```

・pco コマンド

入力例

```
command -> pco
```

実行結果

```
--- --- Program Command List --- ---


scan      input for keyboard
cal       calculate
if        comparison
labl      label
```

```
goto    go to label
rem     comment
prt     parameter display
f       circuit change
end     program finish
```

・ico コマンド

入力例

```
command -> ico
```

実行結果

```
--- --- Interpreter Command List --- ---


A       program Add
I       program Insert
E       program 1 line Edit
del     program 1 line Delete
lst     program List
run     program Run
sav     program Save
lod     program Load
pls     program list in SD card
pco     program command list
ico     interpreter command list
fco     FPGA circuit list
```

・fco コマンド

入力例

```
command -> fco
```

実行結果

```
--- --- FPGA circuit command --- ---

and     and     circuit   0~9
or      or      circuit   0~9
nand    nand    circuit   0~9
nor     nor     circuit   0~9
xor     xor     circuit   0~9
not     not     circuit   0~9
led     led     led circuit
go      GPIO out          0~2
ssd     7 segment driver
co      count circuit
```

4.1.2. ファイルシステム

本システムで記述したコードをファイルシステムにより SD カードにテキスト形式で保存することが可能．また他に以下の機能を実装した．

・SD カード内のテキストファイルの名前を変更する機能.

本システムを起動中読み込まれている時点のテキストファイルの名前を変更することが出来る．その際にファイル名は半角英数で 4 文字以内で指定する事．また，別の名前で保存した場合に p001.txt が新しく生成される．

・記述したコードを SD カードに保存する機能.

記述したコードは自動的に保存され，保存するために特別なコマンドを必要としない．

・SD カード内のテキストファイルを一覧表示する機能.

記述したコードはテキストファイルで SD カード内に保存される．

ファイル名を指定して読み込む場合この機能でファイル名を確認する．表示される文字は 4 文字までとなっている．

4.2. FPGA 側

本システムで利用できる回路の実行結果を載せる.

・and コマンド

[F][and_番号][パラメータ 1][パラメータ 2]

パラメータ 1 とパラメータ 2 に入力した 8bit の値を計算して and 番号のアドレスへ格納する.

入力例 [F][and_0][0x09][0xaf]

図 4.1　and 回路

・or コマンド

[F][or_番号][パラメータ 1][パラメータ 2]

パラメータ 1 とパラメータ 2 に入力した 8bit の値を計算して or 番号のアドレスへ格納する.

入力例 [F][or_0][0x09][0xaf]

図 4.2　or 回路

・nand コマンド

[F][nand_番号][パラメータ 1][パラメータ 2]

パラメータ 1 とパラメータ 2 に入力した 8bit の値を計算して nand 番号のアドレスへ格納する.

入力例 [F][nand_0][0x09][0xaf]

図 4.3　nand 回路

・nor コマンド

[F][nor_番号][パラメータ 1][パラメータ 2]

パラメータ 1 とパラメータ 2 に入力した 8bit の値を計算して nor 番号のアドレスへ格納する.

入力例 [F][nor_0][0x09][0xaf]

図 4.4　nor 回路

・xor コマンド

[F][xor_番号][パラメータ 1][パラメータ 2]

パラメータ 1 とパラメータ 2 に入力した 8bit の値を計算して xor 番号のアドレスへ格納する．

入力例 [F][xor_0][0x09][0xaf]

図 4.5　xor 回路

・not コマンド

[F][not_番号][パラメータ 1][x]

パラメータ 1 に入力した 8bit の値を計算して not 番号のアドレスへ格納する．

入力例 [F][not_0][0x09][x]

図 4.6　not 回路

・go コマンド

[F][go_番号][パラメータ 1][x]

パラメータ 1 で入力した 8bit の値を，指定した GPIO ポートより出力する．

入力例 [F][go_2][0x07][x]

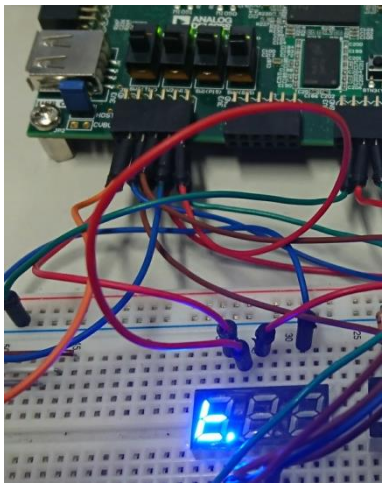


図 4.7 go コマンド実行

・ssd コマンド

[F][ssd][パラメータ 1][x]

パラメータ 1 で入力した 4bit の値を，7segment で表示する際に適した 8bit の値として出力する．

入力例 [F][ssd][0x2][x] [F][go_2][ssd][x]

図 4.8 ssd コマンド実行

・co コマンド

[F][co][パラメータ 1][x]

パラメータ 1 で入力した値と対応したカウンタ周期でカウンタ回路が動く．

GPIO の出力に合わせて 8bit とする．

入力例 [F][co][1][] [F][led][co][]

図 4.9 led で co コマンド実行

回路を切り替える方法はいくつか存在し，全体をリコンフィグする方法，回路をあらかじめ複数用意しておき部分的に書き換える方法（パーシャルリコンフィグ），回路を複数実装しておき必要に応じて回路を選択し切り替える方法とある．しかし，リコンフィグする場合は一度電源を落とす必要があること，パーシャルリコンフィグを利用する場合は開発するうえで有料のライセンスが必要であり，利用者が後から機能を追加する場合コストが掛かってしまうというデメリットがある．そこで今回は回路を複数実装し，都度切り替えるという方法をとる．

パーシャルリコンフィグとは，FPGA の回路を部分的にリコンフィグできるシステムのことをいう．この機能を使うことでシステムを落とすことなく回路を変更することができる．また，通常の変更方法と違い回路を SD カードの中から読み出し現在駆動している回路と切り替えるため，容量を気にすることなく大規模な回路を複数実装することができる．

4.3. 実行手順

本節では本システムの実行手順を示す．本システムを使用する上で，zybo 1 台，Teraterm のインストールされた PC 1 台，microUSB ケーブル，microSD カードを必要とする．

実行手順は以下の通りである．

統合開発環境の入った boot ファイルを microSD カードの中に入れる．

ジャンパーピンの設定を SD にする．

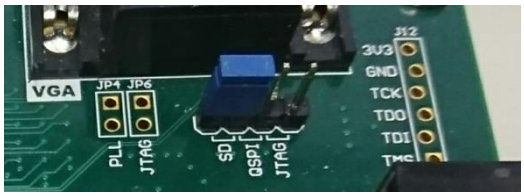


図 4.11　ジャンパーピン

Zynq と PC を USB ケーブルで繋ぐ．

Zynq の電源を入れる．

Zynq のドライバをインストールする．

PC で teraterm を起動する．

teraterm の通信速度を zynq と合わせるために，115200 に設定する．

Zynq のリセットスイッチを押す．

図 4.12　リセットスイッチ

インタプリタプログラムが起動する．

本システムで実際に実験している状況を次に示す．

下記の図 4.13 はボード上の LED を点灯させている状況である．

下記のプログラムは 4bitLED に信号を送り，対応した箇所を光らせるプログラムである．

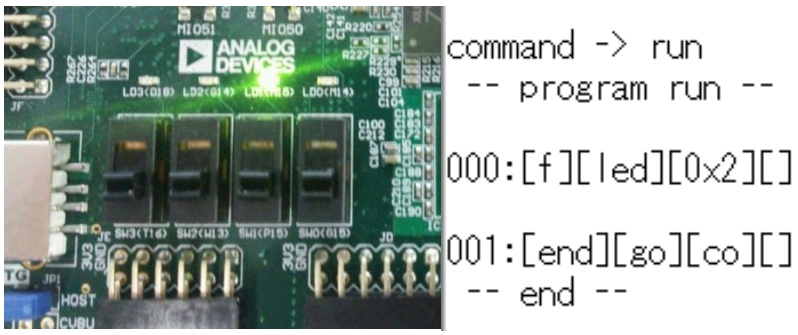


図 4.13　状況　実行画面

下記の図 4.14 は CPU のプログラムのみでの実験画面である．回路を動かすだけでなく CPU を使用しての演算もできるようになっている．

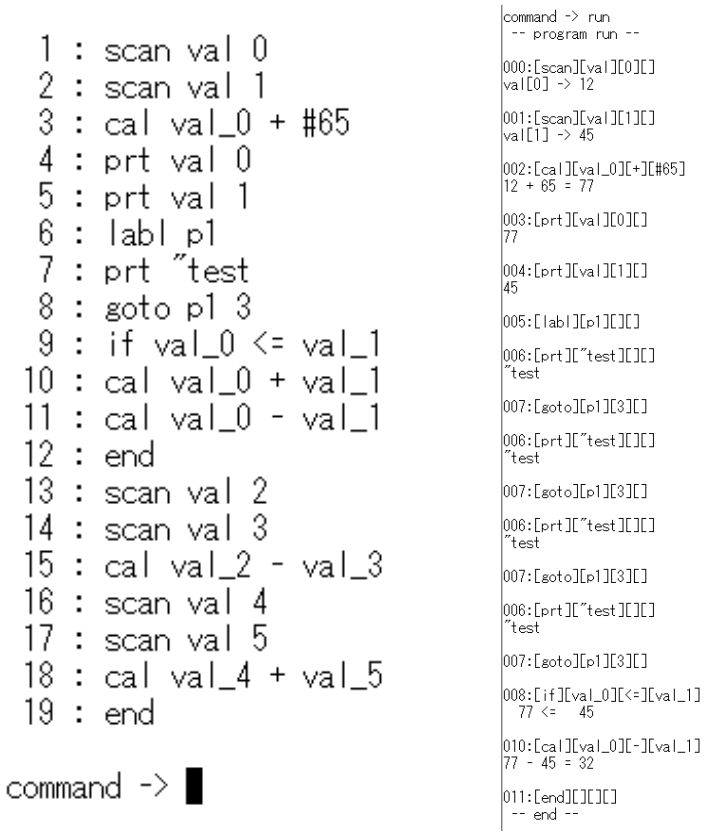


図 4.14　開発画面　実行画面

・GPIO での外部出力の様子

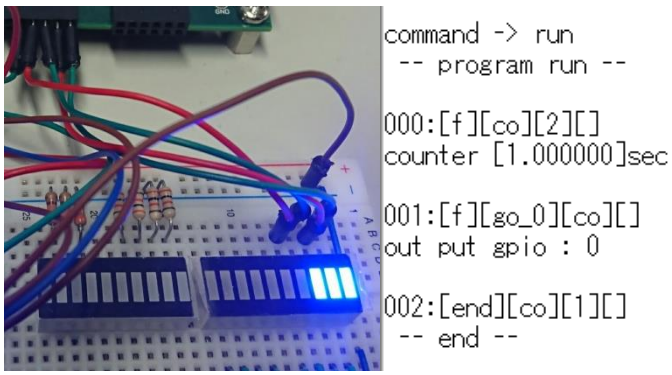


図 4.15　状況　実行画面

4.4. 開発環境

本研究では Xilinx 社が提供している SoC の一つである，zynq の評価ボードである zybo を使用した．今回提案するシステムを開発する際に利用した開発環境と開発機器は以下のとおりである．

表 4.1　開発機器　Zybo スペック

CPU	ARM Cortex A-9 650MHz Dual Core
FPGA チップ	ZYNQ XC7Z010-1CLG400C
ロジックスライス	4400
6 入力 LUT	17600
FF	35200
ブロック RAM	60
ロジックセル	28160
DSP スライス	80
PLL	2
HDMI 入力	1
HDMI 出力	1
Ethernet	1
USB	1
Pmod	6
VGA	1

図 4.16　zybo

表 4.2　開発環境

	開発環境	開発言語
CPU 側	xilinx SDK	C 言語
FPGA 側	Vivado2016.2	VHDL

後から回路を利用者が追加できるように，ライセンスの必要ない開発環境を利用する．

5. 検証

本論文で提案したシステムを実際に使用し，基本的な機能の確認および本システムが実際に簡易的な実験環境として，または教育用として有用かどうかの検証を行った．

検証内容は以下の通りである．

・回路の切り替えが実際に意図した動作で実現可能であること．
・並列処理が行えるように複数の回路を同時に動かせること．
・並列処理を感覚的に扱うに向いているか，感覚的に扱えるか．
・教育用としてまたは簡易的な実験環境として実行までの手順が単純で扱いやすいか．
・実装した回路について適切であるかどうか．

結果

・回路の切り替えが実際に意図した動作で実現可能であること．

実際に GPIO で出力しているコードを図 5.1 に状況を図 5.2 に示す．

```
002:[f][ssd][0x2][]
ssd_out 91

003:[f][go_2][ssd][]
out put gpio : 2
```

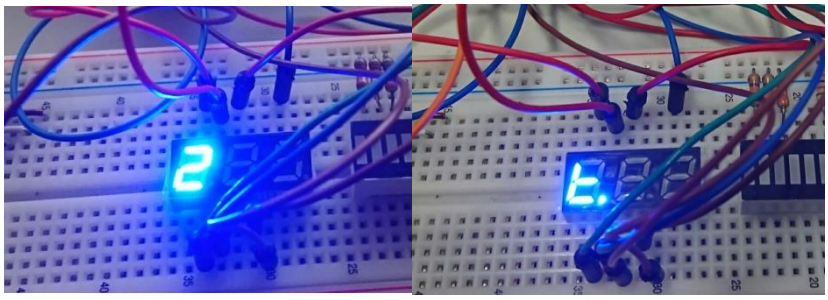
(a)SSD 出力

```
000:[f][co][2][]
counter [1.000000]sec

001:[f][go_2][co][]
out put gpio : 2
```

(b)カウンタ出力

図 5.1　回路の切り替え開発コード



(a)SSD 出力　　(b)カウンタ出力

図 5.2　回路の切り替え

上記の図は 16 進数の値を入れ，その値を表示させる SSD 回路と，1 秒間隔でカウントアップする回路から SSD 回路に値を代入し 8bit でカウントしている回路である．SSD 回路は 8bit の信号で制御でき，SSD 回路用に構築された回路に値を代入するとその数値と対応した数値が表示されるような構成になっている．またこの 8bit は自由に代入することも出き，カウントアップのような信号を拾って点灯させることも可能である．このように同じ GPIO ポートから違う回路を経由して出力させることが可能であることを確認した．

・並列処理が行えるように複数の回路を同時に動かせること．

実際に複数の回路を動作している状況を図 5.3 に示す．GPIO_0 と GPIO_2 からそれぞれカウントア

ップの回路を出力している状態である．

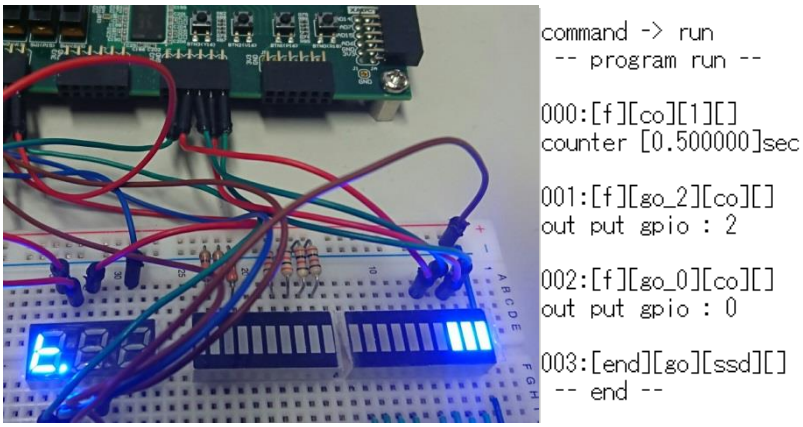


図 5.3　複数の回路動作状況と開発コード

上記の図は違う GPIO の出力ポートから同じカウントアップ回路の信号を拾って 0.5 秒刻みで外部の LED へ出力している図である．このように同じ回路から同時に別の回路へ信号を送り，同時に実行させることが可能であることを確認した．

開発コードはカウントアップの周期の設定と，出力先の設定だけで実現可能である．

・並列処理を感覚的に扱うに向いているか，感覚的に扱えるか．

回路の最大の特徴である並列処理，同時処理について，複数回路を実装し別々の出力から同時に処理することができることを図 5.3 で確認した．また図 5.1 や図 5.3 から実際に動作するまでの開発コードが短く，コードを記述してからすぐに実行できるため，手軽に実行することができ，感覚的に扱うことが出来ると考える．

・教育用としてまたは簡易的な実験環境として実行までの手順が単純で扱いやすいか．

機能を制限し開発工程を簡略化することで実験がスムーズに行えた．表 5.1 で示したコードで実行した場合両方同じ結果を示す．その状況を図 5.4 に示す．

表 5.1　例：zynq ボード上の LED の点灯のコード

	既存の開発環境での開発コード	提案する開発環境での開発コード
プログラム	library IEEE; use IEEE.STD_LOGIC_1164.ALL; entity LED is Port(LED : out STD_LOGIC_VECTOR(3 downto 0);); end LED; architecture RTL of LED is begin 　　LED <= “0010”; end RTL;	[F][led][0x2] [end]
ピン	set_property PACKAGE_PIN M14 [get_ports {LED[0]}] set_property PACKAGE_PIN M15 [get_ports {LED[1]}] set_property PACKAGE_PIN G14 [get_ports {LED[2]}]	なし

設定	set_property PACKAGE_PIN D18 [get_ports {LED[3]}] set_property IOSTANDARD LVCMOS33 [get_ports {LED[*]}]	

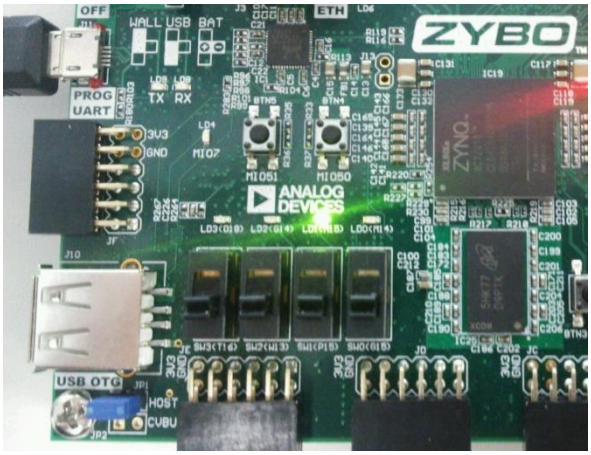


図 5.4　実行結果

このコードは共にボード上の LED を制御して点灯させるものである.

本システムは仕様上回路をあらかじめ用意するためピン設定は全て配置済みであり，再設定の必要がない．また既にコンパイル済みであり，コンパイル時間を省略することができ，決められたコマンドで即座に実行することができる．コマンドが決まっているため余計なコードを書かずに済み，バグの心配を大幅に減らせる．しかしながら実装されている回路しか利用することが出来ず，汎用性が低く利用できる機能に限界がある.

・実装した回路について適切であるかどうか

FPGA 回路の選定でハードウェアでの処理が得意とするものを選択．特にソフトウェアによって動作が安定しないタイマーを 8bit カウンタとして実装した.

考察

実装されている回路は少ないものの簡単に回路を組み合わせ，実行できることを確認した．しかしながら回路への入力に対してのコマンドがまだ少し複雑であり，簡略化を図るにあたり機能を制限する以外の方法で，実現する事を考える必要がある.

提案したシステムは，既存の開発環境で開発を行うより大幅に開発工程を削減したため，利用者の意図が即座に実行に移せる環境であると言える．また短いスパンで複数の回路を短いコードで同時に，繰り返し実行できるため，限られた組み合わせから回路の配置を意識して実験をすることにより，FPGA の並列処理を感覚的につかむことが出来ることを期待する．教育用としても開発環境を統一することにより，開発環境を切り替えることなく CPU と FPGA の両方に対してスムーズに実験出来るようになったと考える.

今後の課題として，よりたくさんの人に利用して貰いより多くの意見を得ること．また今現在実装している回路ではできることが限られるため，今後は回路の種類を増やし，汎用性を広げることが必要であると考える.

6. むすび

本論文では, 近年組み込みシステムにおいて利用されることが増えてきた FPGA がこれからも幅広い分野，多くの場面で利用されると考える．しかし FPGA の需要が高まる一方，世間的に求められている FPGA 技術者の数が少なく，その原因として FPGA の処理方法である並列処理にあるのではないかと考え，FPGA の技術的支援が必要であると考えた．そこで FPGA 初心者でも実際に FPGA に触れ，動かしてみて感覚を掴むことで FPGA 導入へのアシストを目的とし，FPGA 初心者のための実験環境として，また教育用としてのシステムを検討し，小規模開発下における zynq の開発環境として統合開発環境を提案した．本システムを利用することで FPGA を手軽に扱うことができ，簡単な実験であれば実行可能であることを確認した．またプログラミング未経験であったとしても機能を制限し，機能を単純化したことにより，現存する開発環境よりは着手しやすいと考える．今後の課題として回路の種類を増やし汎用性を広げることがあげられる．