HDLによる論理設計・基板製作・プログラミング

の目作入門



CPUの設計と実装 基板設計と製作

プログラミング



オリジナルCPUを実装し、FPGA基板を製作し、 プログラムを動作させるまでの過程を詳しく解説!

技術評論社

本書に記載された内容は、情報の提供のみを目的としています。したが って、本書を用いた運用は、必ずお客様自身の責任と判断によって行って ください。これらの情報の運用の結果について、技術評論社および著者は いかなる責任も負いません。

本書記載の情報は、2012年9月現在のものを掲載していますので、ご 利用時には、変更されている場合もあります。

また、ソフトウェアに関する記述は、とくに断わりのない限り、2012 年9月現在での最新バージョンを基にしています。ソフトウェアはバージ ョンアップされる場合があり、本書での説明とは機能内容や画面図などが 異なってしまうこともあり得ます。本書ご購入の前に、必ずバージョン番 号をご確認ください。

以上の注意事項をご承諾いただいたうえで、本書をご利用願います。こ れらの注意事項をお読みいただかずに、お問い合わせいただいても、技術 評論社および著者は対処しかねます。あらかじめ、ご承知おきください。

◆ Microsoft Windows は米国 Microsoft Corporation の登録商標です。◆その他、本文中に記載されている製品名、会社名等は、すべて関係各社の商標または登録商標です。

はじめに

本書ではオリジナルの CPU をゼロから設計することで、CPU の内部構造を理解するとともに、 CPU 設計の面白さを読者に伝えていきたいと考えています。

本書は CPU の設計をメインターゲットとしていますが、CPU だけでなく周辺機器を制御するため の I/O やバスの設計を行い、SoC としました。CPU だけでなく、ボードの設計やソフトウェアの設 計までコンピュータに必要な要素を全てをカバーします。ハードウェアからソフトウェアまで全てを 自分でゼロから設計・実装し、実機で動かします。CPU の設計から基板の設計、そしてソフトウェ アの設計までを1冊の書籍で扱うことで、それぞれの関連性を包括的に理解することを可能にしてい ます。

CPUの実装には FPGA を使用します。近年、高性能な FPGA が安価で手に入るようになり、個人 レベルで充分楽しむことが可能です。開発ツール等は無償のものを使用し、電子部品は読者が実際 に作ることを考慮し、入手性が高い物を選定しました。製作コストは極力安くなるように配慮しました。

また、CPU 及び I/O、バス等に関しては HDL のソースコードを、ソフトウェアに関してはプログ ラムのソースコードを、技術評論社の Web サイト(http://gihyo.jp/)の本書サポートページで公開 しています。ただし基板に関しては本書に付録として付けるのではなく、既製品をリファレンスモデ ルとすることで対応しています。これにより、読者の興味に応じて作りたい部分だけを作るという読 み方を可能にしています。

本書はエンジニアを目指している学生をメインターゲットとしていますが、前提とする知識を極力 減らし、ハードルを下げることによって様々な層に受け入れられるよう配慮しています。本書は**学ぶ** ことよりも実際に製作することに重点を置いており、「実際に製作することの楽しさ」を知る、という ことが他の技術書と異なる大きなアピールポイントです。FPGA を用いて CPU を設計・実装し、基 板製作を行い、ソフトウェアの開発を行う、これらすべての工程を自らの手で行うという趣旨の書籍 は他に例がありません。雑誌の付属のマイコンを PC でプログラミングして動かすといった手軽な開 発よりも達成感が得られると考えています。

本書では前提知識の敷居を下げ、できるだけ広い層の読者に読んでもらえるよう配慮しています。 しかし、ブール代数や電子回路、プログラミング言語、コンピュータアーキテクチャなどの初歩的な 知識は少なからず必要になります。これらに関しても本文中で最低限の説明をしていますが、紙面の 都合上、体系的に説明するには至らない部分も多いです。これは、「物を作る」ことを主眼とする本 書の性質上、ある程度はご容赦下さい。その代わり、本書の背景となる知識を理解するのに有益な書 籍を、本文中のコラムで紹介していきます。

本書のメインターゲットとなるのは、情報系や電気電子系分野の大学生や高専生としています。ま た、そういった分野を目指す中高生や、コンピュータに興味のある一般の読者にも広く読んでもらえ るように配慮しています。簡単に読める、簡単に理解できるとは言えませんが、意欲を持って読めば 充分に理解できる内容であると考えます。

2012年9月 筆者

3

本書の読み方

本書は大きく3つの章から構成されています。第1章ではCPUを中心に、プログラム やデータを格納するためのメモリや、外部との入出力を行うためのI/O、それらを繋ぐバ スを作成し、簡単なコンピュータシステムを作成します。第2章では、このコンピュータ システムを実機で動作させるための基板を設計・製作します。第3章では、このコンピュ ータシステム用のプログラミングを行い、実機にて動作確認を行います。本書の最大の特 徴は、コンピュータシステムをすべて自作可能ということです。

3つの章はそれぞれ独立しており、読者の興味に応じて好きな章のみを選択して読むこ とができます。



繰り返しますが、本書は大きく3つの章から構成されています。第1章は CPU の論理 設計、第2章は基板設計、第3章はソフトウェア設計です。

第1章のCPUの設計では、CPU、メモリ、I/O、さらにそれらを接続するバスを設計し、 ハードウェア記述言語 Verilog HDL によって実装します。最終的にはそれらを組み合わせ て簡単なコンピュータを作成します。まず、コンピュータ、デジタル回路、Verilog HDL の基礎について説明します。その後、バス、メモリ、CPU、I/O の順でコンピュータを作 成していきます。また、Verilog HDL のシミュレーション環境に関しても説明します。

第2章の基板設計では、製作した CPU やソフトウェアを動作させるための実機を製作 します。CPU の実装には FPGA と呼ばれる、内部構造を書き換えることができる IC を 利用します。製作の手順は、まず必要な部品を選定し、回路図・配線図を作成します。そ の後に実際のプリント基板を製作します。基板の製作では感光基板を利用した方法と、製 造業者に注文する方法を選択することができます。プリント基板が完成したら部品を実装 し、動作確認を取ります。

第3章のソフトウェア設計では、設計した CPU 用のプログラムを作成し、製作した基 板でプログラムを動作させます。まず開発環境について説明します。開発に必要なツール を紹介し、各ツールのインストール方法と使い方を説明します。その後、プログラミング について説明します。サンプルプログラムを用いて、CPU や I/O の使い方を説明し、製 作した基板を使ってプログラムを動かします。

本書での最終的な成果物は実機によるデモプログラムの動作になります。本書では「何 ができるか」よりも「実際に自分の手で製作すること」に重点を置いているため、あまり 高度なデモを用意していません。高度なデモを行うという観点から言えば、市販のマイコ ン基板などを利用したほうがより手軽に製作が行えます。しかし、コンピュータに関わる 全ての部分を自分の手で自作するということは、市販のマイコン基板を動作させるだけで は得られない達成感があります。また、現在、マイコンなどを利用して電子工作をしてい る読者にとっても、論理設計や基板設計、ソフトウェア設計に対してより理解が深まるは ずです。ありものの部品だけでは必要な機能が実装できないケースは多々あります。そん なときに本書を読み直していただけると幸いです。

○目次 contents

はじめに	3
本書の読み方	4
目次	6

第1章 CPUの設計と実装

13

1.1	はじめ	5に	14
1.2	コンヒ	ピュータについて	16
	1.2.1	コンピュータとは	16
	1.2.2	CPUとは	17
	コラム	CPUのビット数	20
	1.2.3	メモリとは	21
	1.2.4	1/0とは	22
	コラム	エンディアン	23
	1.2.5	バスとは	25
	コラム	▼ バスの利点・欠点	27
	1.2.6	まとめ	
	コラム	コンピュータに関する書籍	
1.3	デジタ	マル回路の基礎	29
	1.3.1	デジタル回路とは	29
	1.3.2	数值表現	29
	1.3.3	符号付き2進数	
	コラム	ビットとバイト	31
	コラム	1キロバイトの大きさは?	31
	1.3.4	MOSFETの仕組み	32
	1.3.5	論理演算	

CONTENTS

	1.3.6	CMOS基本論理ゲート	
	1.3.7	記憶素子	
	コラム	▋ セットアップタイムとホールドタイム	
	1.3.8	組み合わせ回路と順序回路	
	1.3.9	クロック同期設計	41
	1.3.10	まとめ	41
	コラム	、デジタルに関する書籍	41
1.4	Veril	og HDLについて	42
	1.4.1	Verilog HDLとは	
	1.4.2	回路記述	
	コラム	▋ デフォルトネットタイプ	
	コラム	■ 組み合わせ回路記述におけるラッチ推定とドントケア	
	コラム	、正論理と負論理	
	1.4.3	シミュレーション記述	
	コラム	、同期回路の信号変化タイミング	65
	1.4.4	Verilog HDLのシミュレーション環境	71
	1.4.5	まとめ	
	コラム	Verilog HDLに関する書籍	
1.5	本章て	ご作成するもの	79
	1.5.1	本章で作成するものの全体像	
	1.5.2	本章のソースコードについて	
	コラム	■ ワードアドレスとバイトオフセット	
1.6	バスの)設計と実装	85
	1.6.1	バスの設計	
	1.6.2	バスの実装	
	1.6.3	まとめ	
1.7	メモリ	りの設計と実装	102
	1.7.1	FPGAのRAM領域	
	1.7.2	ROMの設計と実装	
	1.7.3	まとめ	
	コラム	メモリに関する書籍	

1.8	AZ P	rocessorの設計と実装	107
	1.8.1	CPUについて	
	コラム	CPIとMIPS値	117
	1.8.2	AZ Processorの設計	118
	コラム	■ 命令セットアーキテクチャとマイクロアーキテクチャ	
	1.8.3	AZ Processorの実装	132
	1.8.4	まとめ	
	コラム	コンピュータアーキテクチャに関する書籍	
1.9	I/Oの	設計と実装	195
	1.9.1	タイマ	
	1.9.2	UART	
	コラム	UARTの実用例	
	1.9.3	GPIO	216
	1.9.4	まとめ	
	コラム	I/Oに関する書籍	
1.10	AZPF	R SoCの全体接続	225
	1.10.1	各モジュールの接続	
	1.10.2	クロックモジュールの実装	
	1.10.3	トップモジュールの実装	230
	1.10.4	まとめ	
1.11	AZPF	R SoCのシミュレーション	231
	1.11.1	シミュレーションモデルの作成	
	1.11.2	テストベンチの作成	234
	1.11.3	シミュレーションの実行	
	1.11.4	まとめ	
1.12	おわり	リに	240

第2章 基板設計と製作

241

2.1	はじょ	めに	
2.2	仕様領	휷定	
	2.2.1	基板の名称	
	2.2.2	基板の構成	
	2.2.3	基板のサイズ	
	2.2.4	基板の層数	
	2.2.5	FPGAの選定	246
	2.2.6	周辺回路の選定	246
	コラム	FPGAについて	248
	コラム	JTAGについて	
2.3	部品遺	選定	
	2.3.1	部品の選定基準	
	2.3.2	部品の選定	
	2.3.3	部品の調達	
2.4	回路記	设計	
	2.4.1	データシートの入手	
	2.4.2	コンフィギュレーション回路	
	2.4.3	周辺回路	
	2.4.4	電源回路	
	2.4.5	基板設計環境	
	2.4.6	Eagleによる回路図の作成	
	コラム	ULPについて	
	コラム	S Eagleの使用方法を記載した書籍/マニュアル	
	2.4.7	完成した回路図	
2.5	レイフ	アウト設計	
	2.5.1	基板設計の制約と配線ポリシ	
	2.5.2	FPGA基板のレイアウト設計	
	2.5.3	電源基板のレイアウト設計	
	2.5.4	Eagle によるレイアウト作成	

目次

	2.5.5	完成したレイアウト	
2.6	部品ラ	ライブラリの作成	306
	2.6.1	Symbolの作成	
	2.6.2	Packageの作成	
	2.6.3	Deviceの作成	
2.7	基板の)3D表示	
	2.7.1	使用するツールの説明	
	2.7.2	3Dライブラリの用意	
	コラム	■ 3Dライブラリの管理について	
	2.7.3	基板の3D表示	
2.8	感光基	基板による製作	326
	2.8.1	全体の手順	
	2.8.2	マスク作成	
	2.8.3	マスクの貼り合わせ	
	2.8.4	露光	
	2.8.5	現像	
	2.8.6	エッチング	
	2.8.7	レジスト	
	2.8.8	穴あけ	
	2.8.9	VPortコネクタを裏面に取り付ける場合の処理	
	2.8.10	スルーホール作成	
	2.8.11	ジャンパ配線	
2.9	基板製	としていたい とうしょう とうしゃ とうしん しんしょう しんしょ しんしょ	
	2.9.1	基板製造サービスとは	
	2.9.2	DRC	
	2.9.3	ガーバデータの出力	
	2.9.4	ガーバデータの確認	
	コラム	DFMのチェック方法	
	コラム	レジストマスクの印刷設定	
	2.9.5	P板.com社への注文	
	コラム	■ 面付けデータの用意	

2.12	おわり	に	
	2.11.2	ダイアグプログラム	
	2.11.1	FPGAの認識	
2.11	動作確	認	
	2.10.2	FPGA 基板の実装	
	2.10.1	電源基板	
2.10	部品実	装	
	2.9.6	OLIMEX社への注文	

第3章 プログラミング

379

はじめ	りに	380
開発環	境	381
3.2.1	用意するもの	
3.2.2	FPGA開発環境について	
3.2.3	ISE WebPACK	
3.2.4	UrJTAG	414
コラム	cblsvr-0.1_ft2232	
3.2.5	クロスアセンブラ	
3.2.6	最初のプログラム	436
シリア	'ル通信	442
3.3.1	Tera Term のインストール	
3.3.2	プログラムの作成	
コラム	┃ サブルーチン	
コラム	■ ASCIIコード	
3.3.3	プログラムの実行	
	はじめ 開発環 3.2.1 3.2.2 3.2.3 3.2.4 3.2.5 3.2.5 3.2.6 シリア 3.3.1 3.3.2 コラム 3.3.3	はじめに 開発環境 3.2.1 用意するもの 3.2.2 FPGA開発環境について 3.2.3 ISE WebPACK 3.2.3 ISE WebPACK 3.2.4 UrJTAG 3.2.4 UrJTAG 3.2.5 クロスアセンブラ 3.2.6 最初のプログラム シリアル通信 3.3.1 Tera Term のインストール 3.3.2 プログラムの作成 シリアルーチン 3.3.3 プログラムの実行

3.4	プログ	グラムローダ	
	3.4.1	XMODEM 解説	
	3.4.2	プログラムローダの作成	
	3.4.3	ロード対象のプログラムの作成	
	3.4.4	プログラムの実行	
3.5	割り込	込みと例外	
	3.5.1	割り込みとは	471
	3.5.2	プログラムの作成	
	3.5.3	プログラムの実行	
	3.5.4	例外とは	
	3.5.5	プログラムの作成	
	3.5.6	プログラムの実行	
3.6	7セグ	ブメントLED	
	3.6.1	7セグメントLEDとは	
	3.6.2	7セグメントLEDの制御	
	3.6.3	7セグメントLEDカウンタ動作概要	
	3.6.4	プログラムの作成	
	3.6.5	プログラムの実行	
3.7	応用つ	プログラムの作成	
	3.7.1	動作概要	
	070	プログラムの作成	503
	3.1.2		
	3.7.2 3.7.3	プログラムの実行	

索引		521
謝辞		526
CPU自作入門	おわりに	527

第1章

CPUの設計と実装

本章では、CPU、メモリ、I/O、さらにそれらを接 続するバスを設計し、ハードウェア記述言語 Verilog HDL によって実装します。最終的にはそれらを組み合 わせて簡単なコンピュータを作成します。

本章の最大の特徴は、コンピュータの各要素をハード ウェア記述言語によって実装し、その一つ一つを解説し ていく点です。本章の内容を理解することで、コンピュー タの各要素を理解するとともに、実際に作ることができ るようになるでしょう。

1.1	はじめに	14
1.2	コンピュータについて	16
1.3	デジタル回路の基礎	29
1.4	Verilog HDL について	42
1.5	本章で作成するもの	79
1.6	バスの設計と実装	
1.7	メモリの設計と実装	102
1.8	AZ Processor の設計と実装	107
1.9	1/0の設計と実装	195
1.10	AZPR SoC の全体接続	225
1.11	AZPR SoC のシミュレーション.	231
1.12	おわりに	240

第1章

1.1 はじめに

本章では CPU を中心に、プログラムやデータを格納するためのメモリ、外部との入出 力を行うための I/O、それらを繋ぐバスを作成し、簡単なコンピュータシステムを SoC (System-on-a-Chip) として実装します。SoC とは1つのチップに一連のシステムを集積 する設計手法です。

開発にあたり、まずは CPU の名前を決めます。今回作成する CPU の名前は AZ Processor としました。本書では、コンピュータの設計を最初から最後まで自分たちの手 で行います。このことをアルファベットになぞらえて、A から Z まで全て自分たちの手 で作るという意味を込めました。そして、AZ Processor とメモリ、各種 I/O を搭載し、 バスで結合した SoC を、AZPR SoC(AZ Processor System-on-a-Chip)とします。図 1-1 に AZPR SoC の概要を示します。



▲図 1-1 AZPR SoC の概要

本章の構成を図 1-2 に示します。1.2 節でコンピュータについて、1.3 節でデジタル回路 の基礎について、1.4 節で Verilog HDL についてそれぞれ簡単に説明します。1.2 節から 1.4 節は本章で AZPR SoC を作成していくうえで、最低限必要な前提知識を補う基礎知識

1.1

パートです。既にこれらの知識や設計経験がある方は、読み飛ばして構いません。

1.5 節から 1.10 節が本章のメインとなる設計・実装パートです。1.5 節にて本章で作成 する AZPR SoC について説明します。1.6 節でバス、1.7 節でメモリ、1.8 節で CPU、1.9 節で I/O の設計と実装について説明します。1.10 節で各モジュールを接続し、AZPR SoC が完成します。1.11 節では AZPR SoC のシミュレーションについて説明します。最後に 1.12 節で本章をまとめます。

1.1	はじめに	▶【本章概要】
1.2	コンピュータについて	┣┣ 【基礎知識パート】
1.3	デジタル回路の基礎	コンピュータ、デジタル回路、Verilog HDL の基礎
1.4	Verilog HDL について	すでにこれらの知識がある方は読み飛ばしてください。
1.5	本章で作成するもの	 ▶【設計・実装パート】
1.6	バスの設計と実装	本章のメイン。 パス メモリ CPU I/Oを設計し Verilog HDI
1.7	メモリの設計と実装	で実装します。
1.8	CPU の設計と実装	最終的に簡単なコンビュータを作ります。
1.9	I/O の設計と実装	
1.10	AZPR SoC の全体接続	
1.11	AZPR SoC のシミュレーション	- ▶【AZPR SoC のシミュレーション方法について】
1.12	おわりに	▶ 【本章のまとめ】

▲ 図 1-2 本章の構成

1.2 コンピュータについて

この節では、コンピュータの構成要素とそれぞれの役割について説明します。

1.2.1 コンピュータとは

コンピュータとは、プログラムに従って様々な演算やデータ処理を行う計算機です。近 年、パソコン(パーソナルコンピュータ)は一般家庭に広く普及し、我々の生活に深く関 わってきています。現在では、パソコンだけではなく、携帯電話や家電など、我々の生活 のあらゆる場面でコンピュータが利用されています。

コンピュータは多くの場合、計算やデータの処理などを行う CPU、プログラムやデー タを記憶するためのメモリ、外部とデータのやり取りを行う I/O で構成されています。 各要素はバスで接続され、1 つのコンピュータが構成されます。

図3にコンピュータの構成要素を示します。パソコンを例に見てみると、Intel 社製や AMD 社製の CPU に、DDR3 SDRAM などのメモリ、そして、キーボード、マウス、デ ィスプレイなどの I/O で構成されています。これら CPU、メモリ、I/O、バスはパソコ ンに限らず、多くのコンピュータにおいて共通の4大構成要素です。



▲ 図 1-3 コンピュータの構成要素

1.2.2 CPUとは

CPUとは、コンピュータにおける様々な演算やデータ処理を行う装置です。「中心 (Central)となる処理(Processing)を行う装置(Unit)」であることから、これらの頭 文字をとって CPU と呼ばれています。近年、商用 CPU のほとんどは集積回路として実 装され、図 1-4 のようなパッケージで販売されています。



▲図1-4 CPUの外見

CPU は命令に従って様々な処理を行う電子回路です。CPU の処理の流れを図 1-5 に示 します。メモリ内には、CPU が実行可能な命令群がひとまとまりのプログラムとして格 納されています。CPU はメモリ内にある命令を①読み出し(フェッチ)、それがどのよう な処理を行う命令なのかを②解釈(デコード)し、処理を③実行します。

CPU は大きく分けてこの3つの状態を遷移しながら処理を行います。このような、メ モリ内に格納されたプログラムを読み出して実行する方式のことを、プログラム内蔵方式 と呼びます。



▲ 図 1-5 CPU の処理の流れ

■1読み出し(フェッチ)

はじめに、CPU は実行すべき命令を読み出します。CPU は **PC**(**Program Counter**) というレジスタを持っており、実行するべき命令のアドレスを保持しています。命令の読 み出しでは、PC の値をメモリに出力し、該当アドレスの命令を読み出します。

■②解釈(デコード)

次に、CPUは読み出した命令が、どのような処理を行うものなのかを解釈します。命 令の種類には、様々な演算を行う命令、次に実行する命令を制御する命令、メモリや I/O への読み書きを行う命令、CPUの制御を行う命令などがあります。命令は CPU 内にある 命令デコーダと呼ばれるモジュールが解釈します。アドレスや演算結果の保存などに使用 するレジスタを汎用レジスタ(General Purpose Register)と言います。

■③実行

最後に、CPUはデコードによって決定された処理を行ます。CPUは内部にある記憶装 置であるレジスタや、外部にあるメモリからデータを取り出して処理を行い、結果をレジ スタ、またはメモリへ書き戻します。

簡略化した CPU の内部構造を図 1-6 に示します。命令フェッチでは、CPU は PC の値 をメモリに送り、メモリから命令を読み出します。読み出した命令は命令レジスタに保存 されます。命令デコードでは命令レジスタに保存されている命令を解釈し、実行する処理 を決定します。また多くの場合、処理の決定と同時に、演算に使用するデータを汎用レジ スタから読み出します。命令の実行では、汎用レジスタから読み出した値を演算器によっ て処理し、結果を書き戻します。CPU が実行する処理には、演算した結果を汎用レジス タに書き戻す処理や、メモリへ書き込む処理、メモリから読み出す処理などがあります。



▲ 図 1-6 CPU の内部構造

CPU が実行する命令は、その命令が何の操作を実行するかを表すオペレーションコードと、何に対して操作を実行するかを表すオペランドという2つの部分で構成されています。図 1-7 に命令の構造を示します。命令は特定のビット列として表現され、これを機械語と呼びます。



▲ 図 1-7 命令の構造

オペランドには、レジスタアドレス、メモリアドレス、即値などを指定します。即値と は命令中に埋め込まれた定数のことです。オペランドの数やビット幅は CPU や命令によ って様々で、指定できるオペランドの数によって、3 オペランド形式、2 オペランド形式、 1 オペランド形式などに分類されます。

CPU が実行可能な命令の特徴に応じて RISC (Reduced Instruction Set Computer) と CISC (Complex Instruction Set Computer) という 2 つの分類があります。表 1-1 に RISC と CISC の特徴と代表的な製品の比較を示します。

▼表 1-1 RISC と CISC の比較

	命令の 機能	命令数	ハード ウェア	高速化	同じ処理を行う 際のコードサイズ	代表的な製品
RISC	単純	少ない	簡単	向き	多い	IBM Power、Sun MicroSystems SPARC、MIPS、ARM、etc
CISC	複雑	多い	複雑	不向き	少ない	Intel i386、IBM System/360、 DEC PDP、etc

RISC に分類される CPU の命令は、単純でかつ種類も少ないです。対して CISC に分類 される CPU の命令は、複雑でかつ種類も多いです。RISC は命令が単純で少ない代わり に CPU の内部構造を簡略化することができ、高速化に向いています。しかし、同じ処理 を実現する場合、命令一つ一つが単純なため、CISC よりも命令数が多くなってしまいます。 対して、CISC は内部構造が複雑なので高速化には向きませんが、同じ処理を実現する際 の命令数が RISC より少なくてすみます。

RISC アーキテクチャの大きな特徴の1つとして、メモリアクセスをロードストア命令 に限定している点が挙げられます。このようなアーキテクチャを**ロードストアアーキテク チャ**と言います。メモリアクセスをロードストア命令に限定することで、命令セットやパ イプラインを単純化することができます。ロードストアアーキテクチャでは、演算命令の オペランドには必ずレジスタを使用します。

RISC と CISC はそれぞれ一長一短あり、一概にどちら良いとは言えませんが、クロックの高速化が求められる CPU の世界では RISC の方が有利に思えます。近年の Intel 社製 や AMD 社製の CPU は、命令セットこそ CISC ですが、複雑な命令を一度簡単な命令に 分解し、内部的には RISC のように実行しています。

CPU のビット数

COLUMN

CPU のビット数とは、その CPU が扱うことのできるアドレス空間やデータサイズの大きさ を表しています。たとえば 32 ビット CPU で扱うことのできるデータサイズは 32 ビット、ア クセスできるアドレス空間は 4 ギガバイト(2 の 32 乗)となります。最近ではプログラムの 規模やデータサイズ、メモリの容量が大きくなるにつれて、32 ビットでは足りず、64 ビット CPU が一般的になってきました。

CPU のビット数には明確な定義があるわけではなく、レジスタ幅やアドレス幅、命令長やバス幅など、基準となるビット幅は様々です。大抵はその CPU が扱う整数型のデータ幅がビット幅と解釈されることが多いですが、実際は CPU メーカの意思や主張によって異なります。また、ビット数とは別に、CPU が扱うアドレス空間やデータサイズをワードと言う単位で表します。 一般的にワード長は CPU のビット数と同じになります。

1.2.3 メモリとは

メモリとは、命令(プログラム)やデータを格納し、保存しておくための記憶媒体です。 コンピュータ内でデータやプログラムを記憶する記憶媒体のことを、単なるメモリ区別し て**メインメモリ**などと呼びます。

最近の一般的なコンピュータでは、メモリにDRAM (Dynamic Random Access Memory)が使用されています。DRAM とはコンデンサに電荷を溜めることで情報を保持するメモリです。コンデンサに電荷が溜まっている状態を1、溜まっていない状態を0とすることで、デジタル値を表現します。コンデンサに溜まっている電荷は時間と共に減少するため、一定周期ごとに再書き込みによる記憶保持動作(リフレッシュ)を行う必要があります。DRAM はアクセス方式や規格によって、SDRAM (Synchronous DRAM) や DDR SDRAM (Double Data Rate SDRAM) などの種類があります。

メモリは保存したデータを管理するために**アドレス**という概念を使用します。アドレス はデータを保存した場所を示すもので、データの住所のようなものです。単位データ毎に 一意のアドレスが付けられます。多くの場合、単位データとは1バイト(8ビット)単位 です。このようなアドレスの付け方をバイトアドレッシングと言います。メモリとアドレ スの関係を図 1-8 に示します。



▲図1-8 メモリとアドレス

第1章 CPUの設計と実装

メモリを含む記憶装置は高速なものほどコストも高くなります。そのため、「高速で小容 量」のものから、「中速で中容量」のもの、そして「低速で大容量」のものを組み合わせた 記憶構造を使用します。この構造を記憶階層と言います。図1-9に記憶階層の例を示します。



▲ 図 1-9 記憶階層の例

記憶階層において、最も高速な記憶装置は、CPU内にあるレジスタです。CPUはメモ リに比べて非常に高速なので、CPUがメモリに直接アクセスするのは非効率です。メモ リアクセスを高速化するために、CPUとメモリの間にキャッシュメモリと言う高速で小 容量のメモリを配置しています。

キャッシュメモリはメモリから読み出したデータを一時的に蓄えておきます。CPUが メモリアクセスを行う際、必要なデータがキャッシュ上に存在した場合はキャッシュがア クセスを代行し、高速なアクセスを実現します。キャッシュ自体も容量と速度に応じて記 憶階層になっており、1次キャッシュ、2次キャッシュと言ったマルチレベルのキャッシ ュが用いられることが多いです。

1.2.4 1/0とは

I/O (Input/Output) とは、データの入出力を行う装置です。コンピュータはI/Oを通じて、 外部とデータをやり取りします。コンピュータの処理は、外部からデータを入力し、内部 でデータを処理し、外部に結果を出力するという手順で行われます。パソコンを例に挙げ ると、図 1-11 のようにマウスやキーボードからデータを入力し、プロセッサがプログラ ムに従いデータを処理し、ディスプレイなどを通して外部に結果を出力しています。

COLUMN

エンディアン

複数バイトのデータをメモリ上に配置する際、どの順番でメモリ上に格納するかをエンディ アンと言います。例えば 0x12345678 という 4 バイトのデータがあり、これをメモリに格 納する場合、0 番地に 0x12、1 番地に 0x34、2 番地に 0x56、3 番地に 0x78 と格納する 方法をビッグエンディアンと言います。対して、0 番地に 0x78、1 番地に 0x56、2 番地に 0x34、3 番地に 0x12 と格納する方法をリトルエンディアンと言います。それぞれのデータ格 納方法を図 1-10 に示します。

人間にとってはビッグエンディアンのほうがわかりやすいのですが、コンピュータにとっては データ長が異なっても下位バイトの位置が同じであるリトルエンディアンのほうが扱いやすいと 言われています。

CPU によって採用しているエンディアンは異なり、ソフトウェアの互換性や移植性で問題となることもしばしばあります。Intel 社の x86 アーキテクチャではリトルエンディアンが採用されています。対して、Sun MicroSystems(現 Oracle)社製の SPARC プロセッサや、MIPS Technologies 社製の MIPS プロセッサなどはビッグエンディアンを採用しています。

最近のプロセッサは、ソフトウェアの互換性や移植性という観点から両方のエンディアンをサ ポートし、プログラムに応じてエンディアンを切り替え可能なものも多いです。これを**バイエン** ディアンと言います。





▲ 図 1-11 コンピュータの処理の流れ

I/O ヘアクセスする方法は、大きく分けてメモリマップド I/O とポートマップド I/O の2 種類があります。

メモリマップド I/O とは、I/O もアドレスによって管理し、メモリと同様の方法でアク セスする方式です。メモリマップド I/O の概要を図 1-12 に示します。メモリマップド I/O では、メモリアクセス命令で I/O にもアクセスできるため、ハードウェアを簡略化でき るという利点があります。しかし、アドレス空間が I/O に占有されてしまうのが欠点です。



▲図1-12 メモリマップドI/O

ポートマップド I/O とは、I/O ヘアクセスするための専用命令を CPU がサポートする 方法です。ポートマップド I/O の概要を図 1-13 に示します。ポートマップド I/O では、 アドレス空間が全てメモリに割り当て可能であるということと、メモリアクセスと I/O へのアクセスを命令レベルで区別できるという利点があります。しかし、専用命令を必要



とするので、ハードウェアの設計が複雑になるという欠点があります。

▲図1-13 ポートマップドI/O

1.2.5 バスとは

バスとは CPU やメモリ、I/O が互いにデータをやり取りするための共通の経路のことです。バスでは1つの信号線を複数のデバイスが共有し、通信を行います。図 1-14 にバスの例を示します。

バスを介してデータをやり取りする際、アクセスする側のことを**バスマスタ、**アクセス される側のことを**バススレーブ**などと呼びます。図 1-14の例では、CPU がバスマスタ、 メモリや I/O がバススレーブとなります。



▲図1-14 バスの例

バスは一般的にデータ信号、アドレス信号、複数の制御信号で構成されます。データ信 号はやり取りするデータの伝送、アドレス信号はアクセスするアドレスの指定、制御信号 はバスアクセスの制御を行います。各信号のタイミングなど、やり取りを行ううえでの決 まりをバスプロトコルと言います。バスを介してデータのやり取りを行う一連の動作をバ ストランザクションと言います。バストランザクションの例を図 1-15 に示します。



- ▲ 図 1-15 バストランザクションの例
- [I] バスの使用を要求

多くの場合、バスには複数のバスマスタが接続されますが、バスは共通の経路な ので、複数のバスマスタが同時に使用することはできません。そのため、複数のバ スマスタからのアクセス要求を調停する必要があります。バスにアクセスする権利 のことをバス権と言い、調停のことをバスアービトレーションと言います。バスア ービトレーションはバスコントローラ内にあるバスアービタが行います。バスマス タはバスにアクセスする前にまずバスコントローラに対してバス権を要求します。

[I] バスの使用を許可

バスコントローラは複数のバスマスタからの要求を調停し、アービトレーション ポリシに従ってバスの使用を許可します。

[Ⅲ] アクセスのリクエスト

バス権を得たバスマスタはバススレーブにアクセスのリクエストを送ります。リ クエストには「何番地にアクセスするのか」「読み出しアクセスなのか書き込みアク セスなのか」「書き込みアクセスの場合は書き込むデータ」などの情報が含まれます。

バスは共通の経路なので、バスマスタが出力した信号が全てのバススレーブに送 られます。そこで、どのスレーブに対してのアクセスかを区別するために、**チップ** セレクトなどと呼ばれる制御信号を使用します。チップセレクト信号はバススレー ブ毎に個別に設けられています。チップセレクト信号を使ってアクセスするバスス レーブを選択します。

ー般的なバスでは、バススレーブ毎にアドレス空間が割り当てられており、アク セスするアドレスに基づいてバスコントローラ内にあるアドレスデコーダがチップ セレクト信号を生成します。

[Ⅳ] アクセスへの応答

アクセスされたバススレーブは、リクエストに対する応答をバスマスタに返しま す。リクエストに対する応答はレディなどと呼ばれる制御信号を使用します。読み 出しアクセスだった場合は、応答とともに読み出しデータを出力します。

[V] バス権を解放

COLUMN

バスの使用が終了したら、バスマスタはバスコントローラに通知し、バス権を解 放します。

バスの利点・欠点

バスはバスプロトコルが共通であればどんなデバイスでも簡単に接続できるという利点があり ます。また、共通の経路を用いるため、ハードウェアのコストが低いという利点もあります。し かし、データ転送のスループットがあまり良くないという欠点もあります。

近年、1 つのコンピュータ上に複数の CPU を搭載するマルチプロセッサが一般的になってき ています。バスにアクセスする CPU の数が増えるごとに、バスが混雑することは容易に想像で きます。そこで、1 つの経路を全員で共有するバスに代わって、各ノードをネットワーク(相互 結合網)で接続する技術も開発されています。 1.2

COLUMN

1.2.6 まとめ

この節ではコンピュータについて説明しました。多くのコンピュータは CPU、メモリ、 I/O、そしてそれらを接続するバスによって構成されます。コンピュータでは CPU がメ モリ上に格納されている命令を読み出して実行し、I/O からデータの入出力を行うことで 処理を実現しています。

コンピュータに関する書籍

節末のコラムでは、その節の内容に関するより詳しい話や、体系的な知識を得る助けになる書 籍をいくつか紹介していこうと思います。

●コンピュータはなぜ動くのか(矢沢久雄著/日経 BP 社/ ISBN978-4822281656)

この書籍では、コンピュータに関する初歩的な知識を幅広く紹介されています。ハードウェア、 ソフトウェア、プログラミング、ネットワークなど様々な内容を扱っています。コンピュータと それを取り巻く技術を理解するのに役立つでしょう。いわゆる専門書と言う類の書籍ではなく、 コンピュータを専門にしていない方でも比較的読みやすい書籍です。

 構造化コンピュータ構成 第4版 (Andrew S. Tanenbaum 著/長尾高弘訳/ピアソンエデ ュケーション/ ISBN978-4894712249)

この書籍では、コンピュータに関する知識を体系的に学ぶことができます。この書籍はコンピュータサイエンスを専門にする高専生や大学生が教科書として使える内容です。本格的にコンピュータの勉強をしたいという方には大変お勧めです。原著の著者である Tanenbaum という方は、良い教科書を何冊も書いています。

1.5 本章で作成するもの

この節では、この先本章で作成するものについて説明します。また、本章のソースコードの 読み方や、全体に共通して使用されるマクロなどの説明をします。

1.5.1 本章で作成するものの全体像

AZPR SoC は、AZ Processor を中心に、プログラムを格納するための ROM (Read Only Memory)、時間を測るためのタイマ、シリアル通信規格の UART (Universal Asynchronous Receiver Transmitter)、LED やスイッチを制御するための GPIO (General Purpose Input Output)、それらを繋ぐバスで構成されています。

AZ Processor は専用の SPM (Scratch Pad Memory) を持っており、バスを介さず に高速にアクセス可能です。タイマと UART から出力される割り込み要求信号は、AZ Processor に接続されています。また、AZPR SoC は位相が0度と180度の2種類のクロ ックと、リセット信号を供給する必要があります。外部から入力される基底クロックとリ セットをもとに、クロックモジュールが2種類のクロックとリセット信号を生成していま す。図 1-74 に本章で作成する AZPR SoC のブロック図を示します。



▲ 図 1-74 本章で作成する AZPR SoC

1.5

まず 1.6 節で全体を接続するバスを作成します。次に 1.7 節で ROM を作成します。そ して、1.8 節で本章のメインである CPU を作成します。1.9 節では I/O を作成します。最 後に 1.10 節で全体を接続し、AZPR SoC が完成します。

1.5.2 本章のソースコードについて

■ソースコードの読み方

本章では本文中にソースコードをリストとして貼り込み、説明します。本文中に掲載す るソースコードは、何らかの制御を行っている個所を部分的に切り出して貼り込みます。 基本的にモジュール宣言や信号線の定義などは省略した形で掲載します。その代り、各モ ジュールのポートや信号線、ヘッダで定義されているマクロなどは、表として本文中に掲 載します。

実装する Verilog HDL のモジュールは、1 ファイルに1 モジュールとし、ファイル名と モジュール名を同じものとします。表 1-9 にモジュール階層、表 1-10 にヘッダファイル 一覧を示します。

モジュール	モジュールの説明		
chip_top	トップモジュール		
– clk_gen	クロック生成モジュール		
└ x_s3e_dcm	ザイリンクス Digital Clock Manager		
chip	SoC トップモジュール		
— cpu	CPU トップモジュール		
- if_stage	IF ステージ		
– bus_if	バスインタフェース		
if_reg	IF/ID パイプラインレジスタ		
– id_stage	ID ステージ		
- decoder	命令デコーダ		
id_reg	ID/EX パイプラインレジスタ		
ex_stage	EX ステージ		
- alu	算術論理演算ユニット		
ex_reg	EX/MEM パイプラインレジスタ		
- mem_stage	MEM ステージ		
- bus_if	バスインタフェース		
- mem_ctrl	メモリアクセス制御ユニット		
mem_reg	MEM/WB バイプラインレジスタ		

▼表1-9 モジュール階層

続く➡

- ctrl	CPU 制御ユニット
— gpr	汎用レジスタ
spm	スクラッチパッドメモリ
∟ x_s3e_dpram	ザイリンクスメモリマクロ デュアルポート RAM
– rom	ROM
└ x_s3e_sprom	ザイリンクスメモリマクロ シングルポート ROM
– timer	タイマ
– uart	UART トップモジュール
– uart_tx	UART 送信モジュール
– uart_rx	UART 受信モジュール
uart_ctrl	UART 制御モジュール
— gpio	GPIO
L bus	バストップモジュール
 bus_addr_dec 	アドレスデコーダ
– bus_arbiter	バスアービタ
 bus_master_mux 	バスマスタマルチプレクサ
└ bus_slave_mux	バススレーブマルチプレクサ

▼表1-10 ヘッダファイル一覧

ファイル名	説明	ファイル名	説明
nettype.h	デフォルトネットタイプ指定	bus.h	バスヘッダ
global_config.h	全体設定	gpio.h	GPIO ヘッダ
stddef.h	共通ヘッダ	rom.h	ROM ヘッダ
isa.h	ISA ヘッダ	timer.h	タイマヘッダ
cpu.h	CPU ヘッダ	uart.h	UART ヘッダ
spm.h	SPM ヘッダ		

■コーディングのポリシ

本書の Verilog HDL のソースコードは、可読性と理解のしやすさを第一に考えて記述 しています。ソースコードは可能な限りコメントを記載し、理解の助けになるよう努めて います。本書は日本語の書籍ですので、コメントも日本語にしています。

ソースコード内でのマジックナンバの使用は極力避け、マクロを使用しています。マジックナ ンバとはソースコードに埋め込まれた定数のことです。マジックナンバを避けることにより、ソ ースコードの可搬性が向上します。マクロに関しては全てヘッダファイル内で定義しています。

ソースコードは1行当たり80文字以内とし、インデントにはタブを使用しています。 ソースコードは1行が端末の1行に収まるほうが読みやすいです。一般的な端末は1行が 80文字なので、Verilog HDLに限らずソースコードは1行当たり80文字以内とすること が多いです。また、インデントなどに使用する文字は、タブを使用することをお勧めしま 1.5

す。タブはテキストエディタの設定によって幅を変更できるため、読む人が自由に変更で きるという利点があります。筆者の環境ではタブをスペース4個分の幅で表示しています。

■識別子とマクロの命名規則

識別子は英小文字、数字、アンダースコア(_) で命名します。制御信号は極性を明確 にするため、負論理の信号線には末尾にアンダースコア(_) を付けます。マクロは英大 文字と英小文字、数字、アンダースコア(_) で命名します。定数は大文字とアンダース コア(_) で命名します。ビットの位置やバスなどを定義する場合は、単語の先頭を大文 字にするアッパーキャメルケースで記述します。

マクロの定義はヘッダファイルに記述します。ヘッダファイルにはインクルードガードを付けて再定義を防止します。インクルードガードとは、同じファイルが2回インクルードされる ことを防ぐテクニックです。インクルードファイル全体を「indef」で囲って、インクルードガ ード用のマクロをファイル内で定義することで、再度同じファイルがインクルードされた場合 は「indef」によって無効にされるようにします。マクロの命名規則について図1-75に示します。

`ifndefINC_GUARD	// インクルードガード
`defineINC_GUARD	// インクルードガード用のマクロ
`define DataBus 31:0	// ビットの位置やバスはアッパーキャメルケース
`define DATA_W 32	// 定数は大文字とアンダースコア
`endif	// インクルードガード

▲ 図 1-75 マクロの命名規則

■全体で共通に使用するマクロ

本章のソースコード全体で共通に使用するヘッダファイルを表 1-11 に示します。

▼表 1-11 共通のヘッダファイル

ファイル名	役割
nettype.h	デフォルトネットタイプの定義
global_config.h	変更する可能性のあるパラメータの定義
stddef.h	共通に使用するマクロの定義

「nettype.h」では Verilog HDL のデフォルトネットタイプを定義しています。ミスを防止 するために基本的にデフォルトネットタイプは無効にしています。**リスト 1-4** に「nettype.h」 を示します。

▼ リスト 1-4 デフォルトネットタイプの定義(nettype.h)

```
11 `ifndef __NETTYPE_HEADER_ // インクルードガード
12 `define __NETTYPE_HEADER_
13
14 /********* デフォルトネットタイプ : いずれか1つを選択 *******/
15 `default_nettype none // none (推奨)
16 // `default_nettype wire // wire (Verilog標準)
17
18 `endif
```

「global_config.h」では変更の可能性があるパラメータを定義しています。使用するボ ードが変わった際に変更されるであろうリセット信号の極性や、FPGA が変わった際に 変更されるであろうメモリの制御信号の極性、実装する I/O の選択等を定義しています。 **表 1-12** に「global_config.h」のマクロー覧を示します。

マクロ名	値	意味			
POSITIVE_RESET	NaN	【リセット信号の極性を選択】			
NEGATIVE_RESET	NaN	Active High でリセットする場合は POSITIVE_RESET を定義 Active Low でリセットする場合は NEGATIVE_RESET を定義			
POSITIVE_MEMORY	NaN	【メモリの制御信号の極性を選択】 Active High の場合は POSITIVE MEMORY を定義			
NEGATIVE_MEMORY	NaN	Active Low の場合は NEGATIVE_MEMORY を定義			
IMPLEMENT_TIMER	NaN	【実装する 1/0 を選択】			
IMPLEMENT_UART	NaN	1 タイマを実装する場合は IMPLEMENT_TIMER を定義 UABT を実装する場合は IMPL FMENT UABT を定義			
IMPLEMENT_GPIO	NaN	General Purpose I/O を実装する場合は IMPLEMENT_GPIO を定義			
DESET EDGE	posedge	リセットエッジ(POSITIVE_RESET が定義された場合)			
	negedge	リセットエッジ(NEGATIVE_RESET が定義された場合)			
	1'b1	リセット有効(POSITIVE_RESET が定義された場合)			
RESEI_ENADLE	1'b0	リセット有効(NEGATIVE_RESET が定義された場合)			
	1'b0	リセット無効(POSITIVE_RESET が定義された場合)			
RESET_DISABLE	1'b1	リセット無効(NEGATIVE_RESET が定義された場合)			
	1'b1	メモリ有効(POSITIVE_MEMORY が定義された場合)			
WEWLENABLE	1'b0	メモリ有効(NEGATIVE_MEMORY が定義された場合)			
	1'b0	メモリ無効(POSITIVE_MEMORY が定義された場合)			
	1'b1	メモリ無効(NEGATIVE_MEMORY が定義された場合)			

▼表1-12 マクロ一覧 (global_config.h)

「stddef.h」は共通に使用するマクロを定義しています。信号レベルのHやL、制御信 号の有効や無効など、共通で使用するマクロを定義しています。表 1-13 に「stddef.h」の マクロー覧を示します。 1.5

COLUMN

マクロ名	値	意味	マクロ名	値	意味
HIGH	1'b1	High レベル	WORD_DATA_W	32	データ幅(ワード)
LOW	1'b0	Low レベル	WORD_MSB	31	最上位ビット(ワード)
DISABLE	1'b0	無効(正論理)	WordDataBus	31:0	データバス (ワード)
ENABLE	1'b1	有効(正論理)	WORD_ADDR_W	30	アドレス幅
DISABLE_	1'b1	無効(負論理)	WORD_ADDR_MSB	29	最上位ビット
ENABLE_	1'b0	有効(負論理)	WordAddrBus	29:0	アドレスバス
READ	1'b1	読み出し	BYTE_OFFSET_W	2	オフセット幅
WRITE	1'b0	書き込み	ByteOffsetBus	1:0	オフセットバス
LSB	0	最下位ビット	WordAddrLoc	31:2	ワードアドレスの位置
BYTE_DATA_W	8	データ幅(バイト)	ByteOffsetLoc	1:0	バイトオフセットの位置
BYTE_MSB	7	最上位ビット(バイト)	BYTE_OFFSET_WORD	2'b00	ワード境界
ByteDataBus	7:0	データバス(バイト)			

▼表1-13 マクロ一覧 (stddef.h)

ワードアドレスとバイトオフセット

CPUは1バイトよりも大きい単位のデータを一度に扱う場合があります。たとえば32ビット CPU であれば32ビット (4バイト)のデータ、64ビット CPU であれば64ビット (8 バイト)のデータを扱います。 CPU が扱うデータサイズのことをワードと言い、1ワードごと にアドレスを付ける方式をワードアドレッシングと言います。CPU 内部ではワード単位でデー タを扱うため、アドレスもワードアドレスで扱ったほうが便利な場合があります。

AZ Processor は 32 ビット CPU で、1 ワードが 32 ビット(4 バイト)です。そこで、4 バイトごとにワードアドレスを割り当てます。AZ Processor のアドレス空間は 32 ビットで す。この 32 ビットのアドレスはバイトアドレスですが、CPU 内では上位 30 ビットをワード アドレス、下位 2 ビット(4 バイト分のアドレス空間)をバイトオフセットとして扱います。図 1-76 にワードアドレスとバイトオフセットの関係を示します。



第1章

1.12 おわりに

本章では、CPU、メモリ、I/O、さらにそれらを接続するバスを設計し、簡単なコンピ ユータシステムを SoC として実装しました。本章では、「どうやってつくるのか」に焦点 を絞って解説してきました。

本章で扱ったコンピュータシステムの背景は非常に広く、説明が一足飛びになっている 部分も多々あると思います。しかし、学術的な知識の本質を理解するためにも、実際に「も のをつくる」ことが非常に重要だと思っています。本章の内容がコンピュータシステムを 理解する上での一助になれば幸いです。またそれと同時に、「ものをつくる」ことの面白 さを少しでも伝えられれば嬉しく思います。
第2章

基板設計と製作

本章では、第1章で作成した AZPR SoC を動作さ せる基板の設計と製作を行います。AZPR SoC の実 装には FPGA を使用します。基板の構成は FPGA の 他に、スイッチや LED などの入出力に使用する周辺回 路、各デバイスに必要な電圧を供給する電源回路から なります。

本章の前半では FPGA をはじめ、実装する部品を選 定し、回路図・レイアウトを作成します。後半では基板 を実際に製作する手順について説明します。感光基板 による製作と、基板製造会社に製造を依頼する方法を 選択できます。基板に部品を実装し、動作確認を取る 過程について説明します。

2.1	はじめに	
2.2	仕様策定	244
2.3	部品選定	
2.4	回路設計	
2.5	レイアウト設計	
2.6	部品ライブラリの作成	
2.7	基板の 3D 表示	
2.8	感光基板による製作	
2.9	基板製造サービスへの注文	
2.10	部品実装	
2.11	動作確認	
2.12	おわりに	

第2章

2.1 はじめに

第2章では、第1章で作成した AZPR SoC を動作させる基板を設計・製作します。最 初に第2章で製作する基板の製作例を**写真 2-1** に示します。基板は FPGA 基板と電源基 板から構成されます。FPGA 基板の中央には、AZPR SoC の論理回路を構成する① FPGA を搭載します。周辺回路として、② FPGA のコンフィギュレーション回路、③ LED、④ 7 セグメント LED、⑤プッシュスイッチ、⑥リセット回路、⑦水晶発振器、そして⑧ BOX ヘッダによる汎用入出力回路を搭載します。電源基板は、各デバイスに必要な電圧を供給 する⑨電源回路を搭載します。この基板について、設計と製作の順に説明します。

なお、基板の製作を行わない場合はリファレンスボードを購入してください。本書のサ ポートページに案内があります。



▲ 写真 2-1 本書で製作する基板

製作のフローを図 2-1 に示します。

まず基板の設計について説明します。①仕様策定では、どのような基板を設計するのか 仕様を決めます。この段階で基板の製作に必要な部品表を作成します。

②部品調達について説明します。電子部品はなるべく入手性の高いものを選定したので、 秋葉原の店頭でほとんどの部品を揃えることができます。一部の部品はインターネット通 信販売で入手する必要があるため、ウェブサイトからの購入についても説明します。

その後、Eagle という基板 CAD を用いて基板を設計します。③回路図作成を行い、続いて実際の配線パターンである④レイアウト作成を行います。さらに、回路図とレイアウ

トの作成に必要である⑤ライブラリ作成について説明します。また、必須ではありません が⑥基板の 3D 表示についても説明します。

続いて基板の製作について説明します。⑦感光基板を用いて製作する方法と、⑧基板製 造サービスへの発注を選ぶことができます。感光基板を用いた製作では、感光基板を使用 してパターンのエッチングやレジスト処理、穴加工などを行います。必要となる道具が多 いので、なるべくコストをかけず、片面基板で製作する方法と、コストをかけても仕上が りが良くなる両面基板での製作を選択可能です。基板製造サービスに関しては P 板.com 社 と OLIMEX 社という 2 社について注文方法を記載します。

基板の製造後、⑨部品実装を行います。部品実装は第2章で最も製作難易度が高い箇所 です。本書で製作する基板では中級者以上のはんだ付けのスキルが必要になります。

部品実装後は、⑩正しく基板が動作することを確認します。AZPR SoC を使用した動作確認用 のダイアグプログラムにて動作を確認します。全ての機能が正しく動作すれば基板の完成です。



▲ 図 2-1 第2章の読み方

第2章

2.12 おわりに

本章では、FPGA 基板の設計と製作について説明しました。前半では基板の設計を行 いました。必要な部品を選定し、回路設計・レイアウト設計を行いました。後半は基板の 製作手順を説明しました。基板の製作は、感光基板を用いる方法と、基板製造サービスに 依頼する方法について説明しました。

基板設計を行う際には、レイアウトのパターン設計の段階で部品の実装のしやすさを考 慮すると、設計と製作のバランスが取れた基板が作れます。

最後に、完成した AZPR EvBoard は写真 2-41 のようになります。



▲ 写真 2-41 完成した AZPR EvBoard

第3章 プログラミング

本章では、AZ Processor 用のプログラムを作成し、 AZPR EvBoard でプログラムを動作させます。まず、 AZPR EvBoard の開発環境について説明します。そ の後、AZ Processor のプログラミングについて説明 します。プログラミングの説明では、AZ Processor や AZPR SoC の機能にアクセスするサンプルプログ ラムを説明し、AZPR EvBoard でプログラムが動作 するまでを解説します。

3.1	はじめに	
3.2	開発環境	
3.3	シリアル通信	
3.4	プログラムローダ	
3.5	割り込みと例外	
3.6	7 セグメント LED	
3.7	応用プログラムの作成	
3.8	おわりに	

3.1 はじめに

本章では、AZ Processor 用のプログラムを作成し、AZPR EvBoard でプログラムを動 作させます。まず、AZPR EvBoard の開発環境について説明します。開発に必要なツー ルを紹介し、各ツールのインストール方法と使い方を説明します。次に、AZ Processor のプログラミングについて説明します。サンプルプログラムを用いて AZ Processor や AZPR SoC に搭載されている I/O の使い方を説明し、AZPR EvBoard を使ってプログラ ムを動かします。最終的な応用プログラムとして、キッチンタイマーを作成します。

3.2 節で、開発環境を説明します。節の最後に、AZPR EvBoard の LED を制御するプ ログラムを作成し、動作を確認します。3.3 節から 3.7 節は AZ Processor のプログラミン グの解説をします。3.3 節では、UART を使用したシリアル通信のプログラムによってパ ソコン上に文字を表示させます。3.4 節では、XMODEM を使用したプログラムローダを 作成します。3.5 節では、割り込みと例外について説明します。3.6 節では、7 セグメント LED の制御について説明します。最後に 3.7 節で応用プログラムの説明をします。

3.2 開発環境

本節では、AZ Processor のクロス開発環境について解説します。そして、AZPR EvBoard に実装した LED の制御を行うプログラムについて解説します。

パソコン上で動作するアプリケーションの開発のように、実行環境と同じシステムでプ ログラムを開発することをセルフ開発と言います。これに対して、実行環境とは異なるシ ステムでプログラムを開発することをクロス開発と言います。携帯電話や家電のような組込 み機器では一般にクロス開発が行われています。AZ Processor のプログラム開発もクロス 開発を行います。従って、パソコンでプログラムを作成し、実行ファイルを AZ Processor に転送して実行するという開発手順になります。

3.2.1 用意するもの

AZ Processor のプログラム開発には、AZPR EvBoard、AC アダプタ、パソコン、 USB ケーブルが必要となります。

AZPR EvBoard

AZPR EvBoard は、第2章で解説している FPGA 評価基板です。

■AC アダプタ

AC アダプタは AZPR EvBoard に給電するために使います。2.3.2 項に記載されている ように、出力電圧が 5[V]、出力電流が 1[A] 以上のものを使用してください。

パソコン

開発に使用するパソコンの OS は、Windows 7 を対象としています。AZPR EvBoard との接続は USB ポートを介して行うため、パソコンに USB ポートが搭載されている必要 があります。

■USB ケーブル

USB ケーブルは、パソコンと AZPR EvBoard を接続するために使います。パソコンの USB コネクタは一般に A コネクタです。一方、AZPR EvBoard の USB コネクタはミニ B コネクタとなっています。従って、使用する USB ケーブルは片方のコネクタが A コネ



クタ、もう片方のコネクタがミニBコネクタとなっている必要があります。写真 3-1 に A コネクタ、写真 3-2 にミニBコネクタを示します。





▲ 写真 3-1 USB A コネクタ

▲写真 3-2 USB ミニ B コネクタ

3.2.2 FPGA 開発環境について

FPGA を開発するにあたり、ISE WebPACK と UrJTAG という2種類のツールをイン ストールします。

ISE WebPACK は、ザイリンクス社から提供されている FPGA のための統合開発環境 です。論理合成、配置配線、コンフィギュレーションなどの FPGA の開発に必要な複数 のツールが含まれています。論理合成とは、ハードウェア記述言語によるソースコードを、 ゲートレベルのネットリストに変換することを言います。このネットリストに従い、ゲー トやネットを FPGA の論理ブロックと I/O ピンに割り当てる作業を配置配線と言います。

UrJTAGは、デバイスにアクセスし、JTAG操作を実行するためのツールです。AZPR EvBoard に搭載されている USB のコンフィギュレーション回路を利用して FPGA やコン フィギュレーション ROM への書き込みを行うため、USB をサポートしている UrJTAG を使用します。

これらのツールを使って、第1章で作成したソースコードを変換し、書き込みを行いま す。ファイル変換と使用するツールの対応を図 **3-1** に示します。



▲ 図 3-1 ファイル変換と使用するツールの対応

ISE Project Navigator

ISE Project Navigator は ISE WebPACK に含まれるツールです。ISE Project Navigator を使用して、第1章で作成した Verilog で記述されたソースファイルから、FPGA のコンフ ィギュレーション情報を格納したファイルである BIT ファイルを作成します。

IMPACT

iMPACT は、ISE WebPACK に含まれるツールです。iMPACT というツールを使い、 BIT ファイルから JTAG 操作を記述したファイルである SVF ファイルを作成します。

多くの FPGA は、プログラム素子に SRAM を使用しています。SRAM は揮発性メモ リのため、電源を一度 OFF にしてしまうとコンフィギュレーション情報が消えてしまい ます。電源を OFF にしてもコンフィギュレーション情報が消えないように、FPGA に接 続されている不揮発性メモリのコンフィギュレーション ROM に対してプログラミングを 行うと、電源 ON 時にコンフィギュレーション ROM から FPGA にコンフィギュレーシ ョンが行われます。コンフィギュレーション ROM にコンフィギュレーション情報を書き 込むには MCS ファイルを作成する必要があります。BIT ファイルから iMPACT を使い、

第3章 プログラミング

MCS ファイルを作成します。そして、BIT ファイルと同様に、iMPACT を使って、MCS ファイルから SVF ファイルを作成します。

SVF ファイルは JTAG 操作を記述したファイルです。コンフィギュレーション情報を SVF 形式でファイルに出力し、次に説明する UrJTAG で使用します。

UrJTAG

UrJTAG を使って SVF ファイルに記述されている JTAG 操作を実行し、FPGA やコン フィギュレーション ROM への書き込みを行います。

3.2.3 ISE WebPACK

ここでは、ISE WebPACK のインストール手順と、使い方を説明します。

■インストール

ISE WebPACK はザイリンクス社の以下のウェブサイトからダウンロードできます。

ザイリンクス社ウェブサイト

http://japan.xilinx.com/

ダウンロードのリンクから、図 **3-2** に示す ISE Design Suite の「Windows 用フルイン ストーラ」をクリックします。

ダウンロード時に、ログイン画面が表示されます。ISE WebPACK を使うためには、 ザイリンクス社のアカウントが必要なので、アカウント作成を行います。アカウント作成 画面を図 3-3 に示します。必須入力のフォームに入力し、[Create Account] ボタンをク リックしてアカウントを作成してください。

アカウント作成が終了すると、登録したメールアドレスにメールが送られてきます。メ ールに記載された URL にアクセスすると、登録したユーザ名とパスワードを使ってログ インできるようになります。ログインすると ISE Design Suite のインストーラをダウン ロードできます。ダウンロードしたファイルを解凍して、解凍したファイルの中にある 「xsetup.exe」をダブルクリックすると、インストーラが起動します。[次へ] ボタンをク リックして、インストールを進めます。図 3-4 に示す「インストールするエディションの 選択」の画面では、[エディションのリスト]から「ISE WebPACK」を選択します。



▲ 図 3-2 ISE Design Suite インストーラのダウンロード画面

https://secure.xilinx.com/webreg/create	User.do?goto=https%3A%2	F%2Fsecur	e.xilinx.com%3A44 🏠	⊽ C' ∰ - G	Google	م	俞
			サインイン 言語の選邦	{▼ 流料	ダウンロ	ード お問い合わせ	
				ą	-ワードかパー	9番号を入力 アドバンスサーチ	9
	イノペーション	쵌믑	アブリケーション	サポート	購入	ザイリンクスにつ	17
Create Account							
				3 1	禄方法		
アカウントの作成を完了するために、認証のメールを 物の項目は必須入力です。	6送付いたします。			新 新 坊 よ く	1のバスワー とも数字を1つ	ilは、半角英数字混合(:)含む)、7文字以上を使	少 用
ユーザー ID*				6	ください。		
会社メール アドレス *							
パスワード*							
パスワード再入力。							
姓 *							
名*							
姓 (半角英致)*							
名 (半角英数)*							
	Create Account						

▲ 図 3-3 アカウント作成の画面



▲ 図 3-4 インストールするエディションの選択

インストールが進むと、図 3-5 の「Xilinx License Configuration Manager」ダイアロ グが表示されます。ここで「Get Free ISE WebPack License」を選択し、[Next] ボタン をクリックします。



▲ 🛛 3-5 Xilinx License Configuration Manager (1 / 2)

続いて、図 3-6 のダイアログが表示されるので、[Connect Now] ボタンを選択します。



▲ 🛛 3-6 Xilinx License Configuration Manager (2 / 2)

ウェブブラウザにザイリンクス社のウェブサイトのログイン画面が表示されるので、 先程取得したユーザ名とパスワードを入力してログインします。ログイン後の図 3-7 の 画面で、[ISE Design Suite: WebPACK License] を選択して、[Generate Node-Locked License] ボタンをクリックします。



▲ 図 3-7 ライセンスを取得する画面

以上でインストールが完了し、図 3-8 に示す「インストール完了」の画面が表示されます。

ISE 14.2 and Vivado 2012.2 Insta	iller [100%]	
XILINX. ISE DESIGN SUITE	Install Completed Congratulations! You have successfully installed Xilinx ISE WebPACK. The environment variables are written to the .settings[32[64] bat file for each application encompassing settings[32[64] bat at "C:Wilinx14.2USE_DS". In order to set the variable environment, you must source the settings[32[64].bat file from "C:Wilinx14.2USE_DS". The shortcuts created by the ISE Design Suite Installer source the appropriate settings launching each tool. Command line and script users should source the settings script launching the tools.	n and an es in your : script prior to prior to
ISE 14.2 and Vivado 2012.2 Installer Welcome Accept License Agreements Select Products to Install Select Installation Options Select Destination Directory -> Installation		
Capyright (a) 1995-2012 Xilinx, ha. All rights reserved XUENX the Xilinx lags and after designated brands included herein are trademarks of Xilinx, ha. All after taskemarks are the property of their respective owners.		
		<u> </u>

▲ 図 3-8 インストール完了

■BIT ファイルの作成

BIT ファイルは、FPGA のコンフィギュレーション情報を格納したファイルです。 ISE Project Navigator を使って作成します。ISE Project Navigator は、ソースファイ ルを管理し、論理合成、配置配線などを行うためのツールです。ここでは、ISE Project Navigator を使い、BIT ファイルを作成する方法を説明します。

まず、ISE Project Navigator を起動し、メニューバーから [File] → [New Project] を選択して、新しいプロジェクトを作成します。ISE 起動時の ISE Project Navigator ウ ィンドウを図 **3-9** に示します。

「New Project Wizard」ダイアログが起動するので、「Create New Project」の画面で は、プロジェクトのファイルパスとソースのタイプを入力します。図 **3-10** で示すように [Top-level source type] は「HDL」を選択します。

JSE Project Navigator (P.28xd)	
Elle Edit View Project Source Process Tools Window Layout Help	
Start ↔ □ ♂ ×	
Welcome to the ISE® Design Suite	
Project commands	
Open Project Project Browser_	
New Project	
Recent projects	
Double click on a project in the list below to open	
Additional resources	
Tutorials on the Web	
Design Mesources Application Notes	
Arrest.	
UURSUIE	
<	- F
Console 🕲 Errors 🗘 Warnings 🙀 Find in Files Results	

▲ 図 3-9 ISE 起動時の ISE Project Navigator ウィンドウ

New Project	Wizard	×
Create New Proj Specify project locati	ect n and type. ons and comment for the project	
Name: Location: Working Directory: Description:	azpr D#FPGA¥azpr D#FPGA¥azpr	
Select the type of f Top-level source ty HDL More Info	sp-level source for the project	T Next Cancel

▲ 図 3-10 New Project Wizard (1 / 2)

次に表示される「Project Settings」の画面では、ターゲットデバイスの選択を行います。 AZPR EvBaord の FPGA は Spartan 3E の XC3S250E、パッケージが VQ100、スピード グレードが -4 なので、[Family]の Value に「Spartan3E」、[Device]の Value に「XC3S250」、 [Speed] の Value に「-4」を入力します。入力した画面を図 **3-11** に示します。

iect	
Value	
None Specified	
All	
Spartan3E	
XC3S250E	
VQ100	
-4	
HDL	
XST (VHDL/Verilog)	
ISim (VHDL/Verilog)	
Veriloa	
Store all values	
VHDL-93	
	Next Cance
	ject Value None Specified All Spartan3E XC3S250E VQ100 -4 HDL XST (VHDL/Verilog) ISim (VHDL/Verilog) Verilog Store all values VHDL-93 I

▲ 図 3-11 New Project Wizard (2 / 2)

[Next] ボタンを押して「New Project Wizard」ダイアログを進めます。「New Project Wizard」ダイアログの完了後、プロジェクトにソースコードを追加します。図 3-12 で 示すように、「xc3s250e-4vq100」と表示されている部分を右クリックし「Add Copy of Source」を選択します。



▲図 3-12 Add Copy of Sourceの選択

「Add Copy of Source」ダイアログでは、第1章で解説した AZPR SoC のソースコード をすべて指定します。ここで指定するヘッダファイルの一覧を表 3-1 に、ソースファイル の一覧を表 3-2 に示します。

▼表3-1 ヘッダファイル一覧

ファイル名	説明
nettype.h	デフォルトネットタイプ指定
global_config.h	全体設定
stddef.h	共通ヘッダ
isa.h	ISA ヘッダ
cpu.h	CPU ヘッダ
spm.h	SPM ヘッダ
bus.h	バスヘッダ
gpio.h	GPIO ヘッダ
rom.h	ROM ヘッダ
timer.h	タイマヘッダ
uart.h	UART ヘッダ

▼表3-2 ソースファイルの一覧

ファイル名	モジュールの説明					
chip_top.v	トップモジュール					
— clk_gen.v	クロック生成モジュール					
└_ chip.v	SoC トップモジュール					
- cpu.v	CPU トップモジュール					
if_stage.v	IF ステージ					
bus_if.v	バスインタフェース					
if_reg.v	IF/ID パイプラインレジスタ					
- id_stage.v	ID ステージ					
- decoder.v	命令デコーダ					
id_reg.v	ID/EX パイプラインレジスタ					
ex_stage.v	EX ステージ					
alu.v	算術論理演算ユニット					
ex_reg.v	EX/MEM パイプラインレジスタ					
- mem_stage.v	MEM ステージ					
— mem_ctrl.∨	メモリアクセス制御ユニット					
mem_reg.v	MEM/WB バイプラインレジスタ					
- ctrl.v	CPU 制御ユニット					
- gpr.v	汎用レジスタ					
spm.v	スクラッチパッドメモリ					
— rom.v	ROM					
— timer.v	タイマ					
— uart.v	UART トップモジュール					
uart_tx.v	UART 送信モジュール					
uart_rx.v	UART 受信モジュール					
uart_ctrl.v	UART 制御モジュール					
— gpio.v	GPIO					
L bus.v	バストップモジュール					
bus_addr_dec.v	アドレスデコーダ					
bus_arbiter.v	バスアービタ					
- bus_master_mux.v	バスマスタマルチプレクサ					
bus_slave_mux.v	バススレーブマルチプレクサ					

指定したファイルはプロジェクトを作成したフォルダにコピーされます。図 **3-13** のように「Adding Source Files」ダイアログが表示されるので、[OK] をクリックします。

_	File Name	Associatio	n	Library		-
1	@ alu.v	All		work		
2	Dus.h	All		work	-	1
3	Ø bus.v	All	-	work		
4	bus_addr_dec.v	All	-	work		ľ
5	Ø bus_arbiter.v	All	-	work	-	
6	🔘 bus_master_mux.v	All	-	work		
7	🖉 bus_slave_mux.v	All	-	work	-	
8	💿 chip.v	All	-	work	-	
9	Chip_top.v	All	-	work		
10	⊘ clk_gen.v	All	-	work	-	
11	📀 cpu.h	All	-	work	-	
12	📀 cpu.v	All	-	work	-	
13	🔘 ctrl.v	All	-	work		

▲ 🖾 3-13 Adding Source Files

ソースコードが追加された後のISE Project Navigator ウィンドウを図3-14に示します。

- I:	JSE Project Navigator (P.28xd) - D:¥FPGA¥azpr¥azprxise - [Design Summary]											
X E	Eile <u>E</u> dit <u>V</u> iew P <u>r</u> oject <u>S</u> ource <u>P</u> roc	ess	<u>T</u> ools <u>W</u> indow La <u>y</u> out <u>H</u> elp								- 8 >	×
	00 X B B X B B X B B S	4	* / / Ø Ø / B 🕅 🕞 H		🖻 🎤 K? 🕨	2 1	9					
Desi	gn ↔□♂×	6	Design Overview	^		(:hip_top	Project Sta	tus			1
町	View: A Implementation A Implementation	0	Summary		Project File:	azpr xise		Parse	r Errors		No Errors	11
æ	Hierarchy	G	IOB Properties Modulo Loval Utilization		Module Name:	chip_top		Imple	mentati	on State:	New	11
窗	e azpr	9	Timing Constraints		Target Device:	xc3s250	e-4vq100		Errors			11
=	B xc3s250e-4vq100	ζĴ	- Pinout Report		Product Version:	ISE 14.2			Warnin	es:		11
22	Automatic `includes	179K	Clock Report	=	Design Goal:	Balance	d		Routin	e Results:		11
6	⊕ W top_top (chip_top.v)	泪	- 🙆 Static Timing		Design Strategy:	Xilinx De	efault_		Timing			1
63		AA	Errors and Warnings			Lunlocke	<u>d)</u>		Constr	aints:		41
		(2)	🖹 Parser Messages	i i	Environment:				Score:	Iming		Ш
			Synthesis Messages									1
			Iranslation Messages			r	and a line of	Dementer			r_1	al
			Place and Route Messages		Report Name		Statue	Generated	Frrors	Warnings	Infos	41
			Timing Messages		Synthesis Report		otatus	Cleffer a ce a	LITOIS	warnings	11108	1
	No Processes Running		🗋 Bitgen Messages		Translation Report							11
91	Processes: chip_top		All Implementation Messa…		Map Report							1
91	Design Summary/Reports		Detailed Reports	-	Place and Route Re	port						1
GIVE	🕒 🐲 Design Utilities		Design Properties	=1	Power Report							11
	🐵 🦅 User Constraints		Enable Message Filtering		Post-PAR Static Tir	ning						11
	⊕ 🤁 Synthesize - XST		Optional Design Summary Contents		Report							-11
	🐵 🤃 Implement Design		- Show Clock Report		Bitgen Report							4
	Generate Programming File		Show Failing Constraints									ł
	Analyze Design Using ChinSc-		Show Warnings			Se	condary	Reports			E-1	1
	 Analyze besign osing empter 		Show Errors		Report Name		Statu	:	Gene	erated]
						Date G	ienerate	d: 08/21/201	2 - 02:18	28		
	Start 🕫 Design 🖒 Files 🖺 Li 🗸 🕨	Σ	Design Summary	_	×							1
Cons	iole	_									+□8>	×
JQ I	NFO:ProjectMgmt - Include file	four	nd: 'D:/FPGA/azpr/uart.h' in fi	le	"D:/FPGA/azpr/	uart_	x.v"	line 17				Z,
Q I	NFO:ProjectMgmt - Parsing desig	n hi	lerarchy completed successfully									1
	Launching Design Summary/Report Viewer											
1	III											1
	Console 🖾 Errors 🔬 Warnings 🙀	Find i	n Files Results	-								1
												J
		_		_					_			æ

▲ 図 3-14 ソースコード追加後の ISE Project Navigator ウィンドウ

ここで、ソースコードで宣言のみ行っているモジュールを作成します。図 3-15 の枠で 囲まれている部分のように、宣言のみ行っているモジュールは「?」マークのアイコンが 表示されます。



▲ 図 3-15 宣言のみ行っているモジュール

AZPR SoC では、「x_s3e_dcm」と「x_s3e_sprom」と「x_s3e_dpram」が「?」マー クのアイコンになっています。まずは、「x_s3e_dcm」を作成します。「chip_top (chip_ top.v)」を右クリックして、「New Source」を選択します。図 3-16 に「New Source」を 選択している画面を示します。



▲ 図 3-16 New Source の選択

「New Source」をクリックすると図 **3-17** に示す「New Source Wizard」ダイアログ が起動します。「Select Source Type」の画面では、左側は「IP (CORE Generator & Architecture Wizard)」を選択し、[File name] にはモジュール名である「x_s3e_dcm」 を入力します。

- New Source Wizard	×
Select Source Type Select source type, file name and its location. BMM File ChipScope Definition and Connection File Timlementation Constraints File TIP (CORE Generator & Architecture Wizard) Schematic Schematic User Document Verilog Module Verilog Module Verilog Module VHDL Library VHDL Library VHDL Library VHDL Dackage VHDL Test Bench Embedded Processor	Eile name: x_s3e_dcm Logation: D:¥FPGA¥azpr¥ipcore_dir
More Info	Next Cancel

▲ 🖾 3-17 New Source Wizard (1 / 2)

[Next] をクリックすると図 **3-18** に示す「Select IP」の画面になるので、ここで、[FPGA Features and Design] → [Spartan-3E, Spartan-3A] → [Single DCM _SP] を選択します。

				×
🚱 🍃 New Source Wizard				
Select IP				
Create Coregen or Architecture Wizard IP Core.				
View by Function View by Name				
Name	*	Version	AXI4	A) ^
Spartan-3E, Spartan-3A	M_SP)	13.1 13.1 13.1 13.1 13.1	_	
Search IP Catalog	ily IP co	<u>m</u> patible wi	CI ith chose	ear en part
More Info		<u>N</u> ext		Dancel

▲ 図 3-18 New Source Wizard (2 / 2)

[Next] ボタンをクリックして次に進むと、図 3-19 に示す「Xilinx Clocking Wizard」ダ イアログが表示されます。AZPR SoC では、オシレータから入力される 10[MHz] のクロック と、180 度位相の異なる反転クロックが必要です。反転クロックを生成するために、このダ イアログでは、「CLK180」のチェックボックスにチェックを入れます。また、「Input Clock Frequency」に「10」を入力し、「MHz」を選択します。他の項目は、変更する必要ありません。

👯 Xilinx Clocking Wizard - General Setup	×
CLKIN CLKFB DCM_SI PSEN PSEN PSINCDEC PSCLK	CLK90 CLK90 CLK20 CLK2V CLK2V CLK2X CLK2X180 CLK7X180 CLKFX CLFX
Incut Clock Frequency 10 MHz ns V	iase Shift vge: NONE . alge: 0 0
CLKIN Source	Feedback Source
 External Single Differential 	 External Internal None Single Differential
Divide By ⊻alue	Feedback Value
Use Duty Cycle Correction	< Back Next > Cancel

▲ 🛛 3-19 Xilinx Clocking Wizard

次は、「x_s3e_sprom」を作成します。「x_s3e_dcm」を作成したときと同様に、「chip_ top」を右クリックして「New Source」を選択します。「New Source Wizard」ダイ アログの「Select Source Type」の画面では、「IP(CORE Generator & Architecture Wizard)」を選択し、[File name] にはモジュール名である「x_s3e_sprom」を入力します。 「Select IP」の画面では、図 3-20 のように「All IP versions」にチェックを入れ、[Memories & Storage Elements] → [RAMs&ROMs] → [Block Memory Generator] を選択します。 筆者の環境で Block Memory Generator の Version 7.2 を使用したところ、この後で説明 する Synthesize でエラーが発生することが確認されました。ザイリンクス社へ問い合わ せ、2012 年 8 月 28 日現在バグが含まれているとの回答を得ましたので、[Block Memory Generator] は図 3-20 のように「Version 7.2」以外を選択してください。

Select IP							
Create Corege	en or Archit	tecture Wizard IP Core.					
View by	Function	View by <u>N</u> ame					
Name			*	Version	AXI4	AXI4-Stream	AXI4-Lite
	🖏 Block	k Memory Generator		6.2	AXI4		AXI4-Lite
	- 🌾 Block	k Memory Generator		6.3	AXI4		AXI4-Lite
	A Disel	Montony Constant		6.4	AVTA		AVT4 Libe
	📲 🖏 🖓	k Memory Generator		7.1	AXI4		AXI4-Lite
	S BIOCH	c memory Generator		7.2	AV14		AX14-Lite
1	- 🎖 Distr	ibuted Memory Genera	ator	6.1			•
Coursely ID C							<u> </u>
Search IP C	ATAIOS						Ciear
All IP v	ersions				Onl:	y IP co <u>m</u> patible w	ith chosen pa

▲ 図 3-20 New Source Wizard

「New Source Wizard」ダイアログが完了すると、図 **3-21** で示すように「Block Memory Generator」ダイアログが起動します。ここで、[Memory Type] を指定します。 「x_s3e_sprom」を作成するときは「Single Port ROM」を選択します。

Block Memory Generator		x
View		
IP Symbol &	Logic RE Block Memory Generator	xilinx.com:ip:blk_mem_gen:7.2
	Memory Type Single Port ROM Curcong Options Curcong Options Common Clock	•
	Addressing Options Enable 32-bit Address	
ADDRA[3:0]	ECC Options	
RSTA→→ DSITERR RCM→→ RDADDRECC[1:0]	Algorithm Defines the algorithm used to concatenate the block RAM primitives. See the datasheet for more information.	
INJECTOBITERR	Minimum Area Low Power Fixed Primitives	
	Primitive (Write Port A) : 8bc2 ~ Actual Primitive(s) Used : 8bc2	
IP Symbol 🦉 Power Estimation		nerate Cancel Help

▲ 図 3-21 Block Memory Generator (1 / 4)

第3章 プログラミング

[Next]ボタンをクリックして次に進むと、図3-22に示す画面になります。この画面では、 「Memory Size」の [Read Width] と [Read Depth] を入力します。 [Read Width] は「32」、 [Read Depth] は「2048」に設定します。他の項目は、変更する必要ありません。

🖞 Block Memory Generator		
View		
IP Symbol	Port A Options Memory Size Read Wolth 32 Read Depth 2048 Range: 14608 Range: 29011200	Generator xılınx.com:ip:blk_mem_gen:7
ACCR410.0] ENA	Operating Mode Write First Read First No Change	Enable Always Enabled Use ENA Pin
Q IP Symbol Q Power Estimation	Dotasheet	< gack Poge 3 of 6 [Hext >] [Generate] Cancel] Help

▲ 🖾 3-22 Block Memory Generator (2 / 4)

[Next]ボタンをクリックすると、図 3-23 に示す画面になります。この画面で、「Memory Initialization」の [Load Init File] にチェックを入れ、[Browse] ボタンを押して、初期 化ファイルを指定します。初期化ファイルには、ブロック RAM の初期値を指定する形 式である COE ファイルを指定します。COE ファイルの作成方法は、3.2.5 項で説明しま す。ここで COE ファイルを指定することにより、ROM に初期値が設定されます。AZPR EvBoard に電源が入ったときや、リセットが行われたときには、AZ Processor は ROM からプログラムが読み出され、ここで指定した COE ファイルに記述されたプログラムが 実行されます。

「Memory Initialization」の設定が完了したら、最後に[Generate] ボタンをクリック して設定を完了します。

Block Memory Generator	
View	
P Symbol & ×	logi CORE Block Memory Generator xilm.com:ip:blk_mem_gen:7.7
P Symbol ADDRA(19.9) BUA	Optional Output Registers Port A Subscreen Su
TR Sumhol 🖞 Power Estimation	Datasheet < gack

▲ 図 3-23 Block Memory Generator (3 / 4)

最後に「x_s3e_dpram」を作成します。途中までは「x_s3e_sprom」と同様の手順で 進めます。「chip_top」を右クリックして「New Source」を選択します。「New Source Wizard」ダイアログの「Select Source Type」の画面では、「IP(CORE Generator & Architecture Wizard)」を選択し、[File name] にはモジュール名である「x_s3e_ dpram」を入力します。「Select IP」の画面では、[Memories & Storage Elements] \rightarrow [RAMs&ROMs] \rightarrow [Block Memory Generator] を選択します。「Block Memory Generator」ダイアログの [Memory Type] では「True Dual Port RAM」を選択します。 **図 3-24** で示す画面では、「Memory Size」の [Write Width] と [Write Depth] を指定 します。[Write Width] は「32」、[Write Depth] は「4096」に設定します。

View		
Symbol		Generator xlinx.com.ip:blk_mem_gen
ADDRA(11:5) DIN4(11:5) ENA REGOEA WEQDS WEQDS HORDER RESP	Write Depth 4096 Range: 29011200 Operating Mode Write First No Change	Read Depth: 4096 Enable Always Enabled Use ENA Pin
	Port 8 Options Memory Size Write Width 32 • Write Depth: 4096	Read Wildh: 32 - Read Depth: 4096
Engelion Este → REOCES → WEB(20) RETE → CLKS →	Operating Mode Write First Read First No Change	Enable Always Enabled Use ENB Pin
_		

▲ 図 3-24 Block Memory Generator (4/4)

「x_s3e_sprom」と「x_s3e_dpram」を作成する際に、「Block Memory Generator」ダ イアログで設定変更を行う項目を**表 3-3** にまとめました。

▼表3-3	Block Memory	Generator	で設定変更す	る項目
-------	--------------	-----------	--------	-----

	x_s3e_spromの変更内容	x_s3e_dpramの変更内容
Memory Type	Single Port ROM	True Dual Port RAM
Read Width	32	
Read Depth	2048	
Write Width		32
Write Depth		4096
Load Init File	チェックボックスにチェックを入れ、COE ファイルを指定する。	

これで宣言のみとなっていたモジュールの作成は完了です。プロジェクトを確認すると、 図 3-15 のように「?」マークとなっていたアイコンが図 3-25 のように変化しています。

次に、論理合成を実行します。「chip_top」を選択し、「ISE Project Navigator」ウィン ドウの左下にある「Synthesize-XST」をダブルクリックします。「Synthesize-XST」を選 択する画面を図 3-26 に示します。



▲ 図 3-25 モジュールの作成後

▲図 3-26 Synthesize-XSTの選択

論理合成が完了したら、次は、配置配線を行います。ここでもう1つファイルを作成し ます。配置配線には、様々な制約を記述したファイルが必要です。制約というのは、モ ジュールの入出力の信号線と FPGA のピンの対応や、タイミング、面積などがあります。 これらの制約を記述したファイルが、制約ファイルになります。

制約ファイルは、「AZPR_EvBoard.ucf」というファイルを作成し、テキスト形式で記述します。制約に関するより詳しい話は、ザイリンクス社のウェブサイトよりダウンロードできる以下の「制約ガイド」を参照してください。

制約ガイド

http://japan.xilinx.com/support/documentation/dt ise.htm

第3章 プログラミング

制約ファイルには最低限2種類の情報を記述します。1つは入力クロックのタイミング 制約、もう1つは FPGA のピンに対する制約です。入力クロックのタイミング制約は、 以下のように記述します。

```
NET "clk_ref" TNM_NET = "CLK" ;
TIMESPEC "TS_CLK" = PERIOD "CLK" 100 ns HIGH 50%;
```

1行目は、「clk_ref」という信号線をクロック信号として認識させ、「CLK」という名前 を付けるための記述です。2行目で「CLK」に対してタイミング情報を記述しています。 10[MHz]の周波数を周期に変換した100[ns]と、Hである期間が50%であるという記述で す。

また、FPGAのピンに対する制約は以下のように記述します。

NET clk_ref LOC = P83;

これは、RTLのトップモジュールの「clk_ref」という信号線を FPGA の「P83」というピンに対応させるという意味です。制約は基板上の配線に合わせて決める必要があります。AZ Processor の信号線と AZPR EvBoard のピンの対応を**表 3-4** に示します。

信号線	ピン	信号線	ピン	信号線	ピン
clk_ref	83	gpio_out<4>	16	gpio_io<2>	62
reset_sw	85	gpio_out<5>	90	gpio_io<3>	63
uart_rx	70	gpio_out<6>	86	gpio_io<4>	65
uart_tx	71	gpio_out<7>	11	gpio_io<5>	66
gpio_out<16>	54	gpio_out<8>	3	gpio_io<6>	67
gpio_out<17>	53	gpio_out<9>	2	gpio_io<7>	68
gpio_in<0>	22	gpio_out<10>	5	gpio_io<8>	33
gpio_in<1>	23	gpio_out<11>	9	gpio_io<9>	34
gpio_in<2>	24	gpio_out<12>	10	gpio_io<10>	35
gpio_in<3>	26	gpio_out<13>	95	gpio_io<11>	36
gpio_out<0>	91	gpio_out<14>	94	gpio_io<12>	40
gpio_out<1>	92	gpio_out<15>	4	gpio_io<13>	41
gpio_out<2>	12	gpio_io<0>	60	gpio_io<14>	57
gpio_out<3>	15	gpio_io<1>	61	gpio_io<15>	58

▼表 3-4 AZ Processor の信号線と XC3S250E のピンの対応

gpio_io 信号には「PULLDOWN」という記述を追加します。

NET "gpio io<0>" LOC = "P60" | PULLDOWN;

「PULLDOWN」という記述により、FPGAの「P60」を FPGAの内部抵抗で GND と 接続することができます。AZPR EvBoard では、gpio_io は BOX ヘッダに接続されており、 BOX ヘッダに外部デバイスを接続していない場合、FPGA のピンはどこにも接続されて いない状態になります。この状態では FPGA に対して H が入力されているか L が入力さ れているかわからなくなってしまうため、抵抗を介して GND に接続する必要があります。

作成した「AZPR_EvBoartd.ucf」をプロジェクトに追加して配置配線を行う準備が完 了です。「ISE Project Navigator」ウィンドウの左下にある「Implement Design」をダブ ルクリックすると、配置配線が実行されます。「Implement Design」を選択する画面を図 **3-27**に示します。

次に BIT ファイルを作成します。まず、図 **3-28** で示す画面で「Generate Programming File」を右クリックして「Process Properties」を選択します。



▲図 3-27 Implement Designの選択



▲ 図 3-28 Process Properties の選択

第3章 プログラミング

「Process Properties」ダイアログが開いたら、左側「Category」より「Startup Options」を選択します。ここでは、[FPGA Start-Up Clock]を設定します。FPGA を直 接コンフィギュレーションする場合には Value を「JTAG Clock」、コンフィギュレーション 情報をコンフィギュレーション ROM に書き込む場合には Value を「CCLK」に設定します。 「Process Properties」ダイアログを図 **3-29** に示します。

stegory	Switch Name	Property Name	Value
- General Options	-g StartUpClk:	FPGA Start-Up Clock	CCLK
Startup Options	-g DonePipe:	Enable Internal Done Pipe	CCLK
Readback Options	-g DONE_cycle:	-g DONE_cycle: Done (Output Events)	
	-g GTS_cycle:	Enable Outputs (Output Events)	Default (5)
	-g GWE_cycle:	Release Write Enable (Output Events)	Default (6)
	-g LCK_cycle:	Wait for DLL Lock (Output Events)	Default (NoWait)
	-g DriveDone:	Drive Done Pin High	
	Property disc	ilay level: İStandard 💌 闭 Display switcl	h names Default

▲ 🖾 3-29 Process Properties

「Generate Programming File」をダブルクリックすると、プロジェクトのフォルダに BIT ファイルが生成されます。トップモジュールの名前が「chip_top」なので、「chip_ top.bit」というファイル名となります。

■MCS ファイル作成

ここでは BIT ファイルから MCS ファイルを作成す る方法を説明します。まず、「ISE Project Navigator」 ウィンドウの左下にある「Generate Target PROM/ ACE File」をダブルクリックします。「Generate Target PROM/ACE File」が表示されている画面を 図 **3-30** で示します。

「Generate Target PROM/ACE File」をダブル クリックすると、「ISE iMPACT」ウィンドウが起 動します。「ISE iMPACT」ウィンドウでは、図 **3-31** で示すように左上の領域の「iMPACT Flows」



▲ 図 3-30 Generate Target PROM/ACE Fileの選択

から「Create PROM File(PROM File Formatter)」をダブルクリックして、PROM File Formatter ダイアログを起動します。PROM File Formatter ダイアログを図 **3-32** に示し ます。



▲ 図 3-31 Create PROM File (PROM File Formatter)の選択

tep 1. Select Storage Targ	et Stop 2. Ad	d Storage Device(s)		Step 3.		Enter D	ata
torane Davice Tune -	Target FPGA	Spartan3E 🚽		eneral File Deta		Value	
- Xilinx Flash/PROM	Storage Device (bits)	51 2K 🔟		Checksum Fill Value	FF		
Spartansan	Add Storage Device	Remove Storage Device	vice	Output File Name	. Untitled		
Configure Single FPGA Configure MidBoot FPGA Configure MidBoot FPGA Configure MidBoot FPGA Configure MidBoot FPGA Configure MidBoot FPGA Configure MidBoot FPGA Configure FOR Parallel PROM				Output File Location	C:\Xilinx\14.2\ISE_DS\		B
				Flash/PROM F	le Property	Value	-
				File Format		BIN	
				Use Power-of-2 for Start Addr		No	
				Number of Bitstream		2	_
				Bitstream 0 Start A	ddress	0	
				Bitstream 1 Start A	ddress	675840	
				Add Non-Configura	ition Data Files	Yes	-
				Number of Data Fil	e		
	Auto Select PROM			•			Þ
scription:							
I PROM File Formatter will guide you through the st ROM device can then be used to configure the targe dditional capabilities of the PROM File Formatter inclu	eps to format bitstream BIT files into a P t FPGA. de:	ROM file that is compatible with :	Xilin x19 a	nd third-party PROM	programmers. T	'he programmed	

▲ 🖾 3-32 PROM File Formatter (1 / 3)

PROM File Formatter ダイアログでは、最初に Step 1. の入力を行います。Step 1. では、 「Xilinx Flash/PROM」を選択して矢印ボタンを押します。矢印ボタンを押すと、Step 2. の入力に移ります。図 3-33 に Step 2. の入力画面を示します。

Step 1. Select Storage Targe	t <i>Step 2</i> .	Add Storage	Device(s)	Step 3.		Enter D)ata
Storage Device Type :	PROM Family	Platform Flag	h 💌	eneral File Deta		Value	
Xiinx Flash/PROM Non-Volatile FPGA Separtme 24M	Device (bits)	xof02s	[2 M] 🖵	Checksum Fill Value	FF		
SPI Flash	Add Storage	Device Remove Sto	orage Device	Output File Name	Untitled		
Configure Single FPGA Configure MultiBoot FPGA	xcf02s	[2 M]		Output File Location	C:\Xiinx\14.2	ISE_DS\	P
Configure Single FPGA Configure MultiBoot FPGA				Flash/PROM Fi	ile Property	Value	
Configure from Paralleled PROMs				File Format		BIN	
				Use Power-of-2 for	Start Addr	No	
				Number of Bitstrea	m	2	_
				Bitstream 0 Start A	ddress	0	
				Bitstream 1 Start A	ddress	675840	
				Add Non-Configura	ition Data Files	Yes	
				Number of Data Fil	e		_
	Auto Select	PROM		<			Þ
ascription:							
1 this step, you will select the appropriate target PROMG • PROM Family. This selection allows you to che • Device: This selection identifies the specific de • Add Storage Device: After selecting the PRC • Remove Storage Device: Use this button to) and their sizes. lose the specific family you a loce in the PROM Family sele M Family and Device, use thi delete a storage device from	re targeting. cted above. s button to add the devic i the list below. Select the	e to the list of Storage D e devloe you wish to dele	evices targeted. te and click this butt	on to remove it	from the list.	

▲ 図 3-33 PROM File Formatter (2 / 3)

Step 2. では、「Device」にコンフィギュレーション ROM の型番である「xcf02s」を選択し、 [Add Storage Device] ボタンを押した後、矢印ボタンを押します。矢印ボタンを押すと、 Step 3. の入力に移ります。図 3-34 に Step 3. の入力画面を示します。

tep 1.	Select Storage Targe	t	Step 2.	Add S	torage Device(s)	Step 3.	Ente	r Data
torage Device Ty	pe :		PROM Family	P	atform Flash	eneral File Deta	Value	
Non-Volatile Fl	OM PGA		Device (bits)	20	f02s [2 M] 💌	Checksum Fill Value	FF	
SPI Flash			Add Storage D	evice F	emove Storage Device	Output File Name	Untitled	
Configure Configure BPI Flash	Single FPGA MultiBoot FPGA		xcf02s	[2 M]		Output File Location	C:\Xilinx\14.2\ISE_DS\	P
Configure Single FPGA Configure MultiBoot FPGA					Flash/PROM F	ile Property Va	ue	
Configure	from Paralleled PROMs					File Format	MCS	í
			Auto Select F	PROM				
scription:			Auto Select F	PROM				

▲ 🛛 3-34 PROM File Formatter (3 / 3)

[File Format]に「MCS」を選択します。[OK]をクリックすると、図 3-35 のように「Add Device」ダイアログが表示されるので、[Yes] ボタンをクリックし、「chip_top.bit」を選択します。



▲ 図 3-35 Add Device ダイアログの表示

AZPR EvBoard に搭載されている FPGA は1つなので、コンフィギュレーション用 の BIT ファイルも1つだけ選択します。「chip_top.bit」選択後の「Add Device」ダイア ログでは [No] ボタンをクリックします。「ISE iMPACT」ウィンドウの左下の領域の 図 3-36 に示す「Generate File」をダブルクリックすると MCS ファイルが作成されます。 MCS ファイル作成完了の画面を図 3-37 に示します。



▲図 3-36 Generate Fileの選択





■SVF ファイル作成

SVF は Serial Vector Format の略で、JTAG 操作を記述したファイルです。コンフィ ギュレーションデータを SVF 形式でファイルに出力し、3.2.4 項で説明する UrJTAG とい うツールで使用します。UrJTAG で SVF ファイルを再生し、デバイスに対して JTAG 操 作を行います。ここでは、SVF ファイル作成の手順を説明します。

まず、Windows 7 のスタートメニューから iMPACT を起動します。図 3-38 に「ISE iMPACT」ウィンドウを示します。「ISE iMPACT」ウィンドウの左上の領域の「iMPACT Flows」から「Boundary Scan」をダブルクリックします。



▲図 3-38 Boundary Scanの選択



ISE IMPACT (P28xd) - [Boundary Scan]	the second s	X
Eile Edit View Operations Output	Debug Window Help	_ _
	No mandon men	
	7:	
Boundary Scan	ight click device to select operations	SVF FILE MODE
SystemACE	Jorgana, L.S.J.	
Create PROM File (PROM File For…	TDI £XIUM*	
🐵 📓 WebTalk Data	L	
	xc3s250e	
	chip_top.bit TDO	
iMPACT Processes ↔		
Available Operations are:		
	31	
	Boundary Scan	
Console		⇔⊡æ×
Q INFO:IMPACI:SUI - 'I': Added D	vice xcsszoue successfully.	<u>^</u>
<		•
Console 🛛 Errors 🔬 Warnings		
	Boundary-Scan File Generation SVF-STAPL-XSVF SVF	D:/FPGA/svf/azpr_s3e_250E_led.svf SVF

Boundary Scan 実行後の画面を図 3-39 に示します。

▲図 3-39 Boundary Scan 後の画面

図 3-39 の「Right click device to select operations」と表示された領域を右クリック し、図 3-40 に示す [Output File Type] → [SVF File] → [Create SVF File] を選択し、 SVF ファイル作成を開始します。

Add Xilinx Device Add Non-Xilinx Device	Ctrl+D Ctrl+K			
Initialize Chain	Ctrl+I			
Cable Auto Connect Cable Setup				
Output File Type	Þ	SVF File	Þ	Create SVF File
		STAPL File	۲	Append to SVF File
		XSVF File	•	

▲図 3-40 Create SVF Fileの選択
「Add Device」ダイアログで書き込むファイルを選択します。ここで BIT ファイルを 選択する場合と MCS ファイルを選択する場合で手順が別れます。

BIT ファイルを選択すると、図 3-41 のように「xc3s250e」が追加されます。



▲ 図 3-41 BIT ファイル追加後の画面

MCS ファイルを選択すると、図 **3-42** のように「Select Device Part Name」ダイアロ グが表示されます。ここで、PROM デバイスとして、「xcf02s」を選択します。[OK] ボ タンをクリックすると、図 **3-43** のように「xcf02s」が追加されます。

🛞 Select Device Part Name		×
Select PROM		
Part Name:	xcf02s	
Ōĸ	Gancel	Help

▲ 🛛 3-42 Select Device Part Name

ISE iMPACT (0.61xd) - [Boundary Scan]		
Belle Edit View Operations Output	Debug Window Help	_ <i>6</i> ×
	× ۹	
iMPACT Flows ↔	2 ** ** 2 Ket Addition to extent encountries	
MMPACT Processes → □ ਰ × Available Operations are:	Eight dick drvice to select operations TDI xett23a TDO zerones	
	Bourdary Scan	
Console		↔ ⊡ & ×
'1': Loading file 'D:/FPGA6_201 done.	111219/azpr.mcs'	* *
Console C Errors A Warnings		
	No Cable Connection No Fil	e Open

▲ 図 3-43 MCS ファイル追加後の画面

書き込むファイルを選択した後、追加されたデバイス上で右クリックを行った画面を図 3-44 に示します。ここで、「Program」を選択します。



▲ 図 3-44 Program の選択

図 3-45 で示す「Device Programming Properties」ダイアログが開くので、そのまま[OK] ボタンを押します。

Device Programming Properties - De	vice 1 Programming Properties	
Boundary-Scan		
Device 1 (FPGA xc3s250e)	Property Name	Value
	FPGA Device Specific Programming Properties	5
	Pulse PROG	
:		
	OK Cancel	Help Help

▲ 🖾 3-45 Device Programming Properties

最後に、図 3-46 のように「Program Succeeded」と表示されたのを確認後、メニュー バーから [Output] → [SVF File] → [Stop Writing to File] を選択し、SVF ファイル の作成を終了します。



▲ 図 3-46 SVF ファイル作成完了

以上の手順で SVF ファイルが生成されます。

3.2.4 UrJTAG

ここでは、UrJTAG のインストールと使用方法について説明します。

■インストール

UrJTAG、FT2232 ドライバ、libusb-win32 のインストールを行います。

UrJTAG

UrJTAG は SVF ファイルに記述された JTAG 操作を実行し、コンフィギュレーション するために使います。UrJTAG は以下のウェブサイトからダウンロードできます。

UrJTAG

http://urjtag.org/

ウェブブラウザで上記 URL にアクセスし、図 **3-47** で示す「Download」をクリックしてください。



▲ 図 3-47 UrJTAG のウェブサイト

図 3-48 で示すページに移動するので、最新版のインストーラをダウンロードします。

) 🕘 sourceforge.net/projects/urj	tag/files/	<u>∽</u>	C Google	<u>۶</u>
ourceforge (Search Browse Bl	og Help		Log In
UTION CENTERS Go Parallel HT	ML5 Center Private Cloud Smarter IT Ex	rents Newsletters		
bax 🖁	nple, Secure Sharing om Anywhere	1111	TO IT Service	R
Universal JTAG library, ser	Ver and LOOIs and an armini, kawk, vapier Support Develop Hosted Apps Tracke	r Mailing Lists Forums	Code	
Universal JTAG library, set Summary Files Reviews Looking for the latest version Dow	Vef and tools are an amini, kawk, vaper Support Develop Hosted Apps Tracke mload urjtag-0.10.exe (887.9 kB)	er Mailing Lists Forums	Code	a .'
Universal JTAG library, ser Summary Files Reviews Looking for the latest version Dow Home Name •	Vef and tools an amini, kawk, vaper Support Develop Hosted Apps Tracke mload urjtag 0.10.exe (887.9 kB) Modified • Size •	er Mailing Lists Forums	Code	a '
Universal JTAG library, ser Summary Files Reviews Looking for the latest version Dow Home Name 9 In urjtag	Ver And LOOIS and Ammini, kawk, vaper Support Develop Hosted Apps Track mload urjtag.0.10.exe (887.9 kB) Modified • Size • 2009-04-17	er Mailing Lists Forums	Code	a '
Universal JTAG library, ser Summary Files Reviews Looking for the latest version Dow Home Name 9 urjtag Totals: 1 Item	Ver and tools a a amini, kawk, vaper Support Develop Hosted Apps Tracke mload urjtag 0.10.exe (887.9 kB) Modified • Size • 2009-04-17	er Malling Lists Forums	Code	d 1

▲ 図 3-48 UrJTAG インストーラのダウンロードページ

図 3-49 で示すように、ダウンロードしたファイルを右クリックして、「管理者として実行」を選択します。インストーラに従って進めていけばインストールすることができます。



▲ 図 3-49 管理者として実行の表示

■FT2232 ドライバ

FPGA のコンフィギュレーションは、AZPR EvBoard に搭載されている FT2232 とい うデバイスを通して行うため、ドライバのインストールが必要になります。Windows 7 では、FT2232 を接続すると、ドライバは自動でインストールされます。USB ケーブルで パソコンと AZPR EvBoard を接続し、電源を ON にするとドライバのインストールが始

第3章 プログラミング

まります。ドライバのインストールが完了すると、FT2232 が認識されます。

Windows 7 が FT2232 を認識できているかどうかの確認は、デバイスマネージャーか ら行います。スタートメニューからコンピューターを右クリックして「プロパティ」を選 びます。デバイスマネージャーを選択し、デバイスマネージャーウィンドウを表示させま す。「ポート (COM と LPT)」を展開して、通信ポートを確認します。FT2232 は 2 チャ ンネルの USB シリアル変換 IC なので、正しく認識されていれば USB Serial Port が 2 つ 表示されます。図 3-50 では、「USB Serial Port (COM3)」と「USB Serial Port (COM4)」 と表示されています。

ファイル(E) 操作(A) 表示(⊻) ヘルプ(H)
Image: State of the stat

▲ 図 3-50 デバイスマネージャー

libusb-win32

libusb-win32 は、USB デバイスを扱うためのドライバです。以下のウェブページから ダウンロードできます。

libusb-win32

http://sourceforge.net/apps/trac/libusb-win32/wiki/

ウェブブラウザで上記 URL にアクセスし、図 **3-51** で示す「Download」の項目から 「project download site」をクリックしてください。

Firefox 👻					×
🛊 libusb-win32	+				
Sourceforge.net/apps/trac/libus	b-win32/wiki	☆ ⊽ C	🛃 = Google	٩ م	
In the luture the project aut	ministrators will add a new bob optic	in to the license to address the	whice licensing issue	•	*
Support					
If something isn't working as e packs for your OS before reque	xpected, make sure that you have in esting for any support.	nstalled the the latest version of	f libusb-win32 and the	alatest service	
Available Support Options:					
 A th mailing list is available for channel. Please th subscribe The project page offers differ describe your problems and y [*]testibusb-win.exe[*], etc.). T Debug version of the libusb information can be printed out 	r discussions, questions, bug report to the list first before posting. rent forms which can be filled out tr rour system as precise as possible ('his will make solving problems a lot win32 are provided from libusb-1.2.3 ut to facilitate easier debugging prod	s, feature request, and other iss o get support, to report bugs, or DS, service packs, version of lib easier. .0 onwards. Together with ⇔ Min ess.	sues. It is the preferm to request new feat usb-win32, type of de crosoft DebugView, d	ed support ures. Please evice, output of etailed debug	Е
You should also read the libust	-win32 FAQ (Frequently Asked Que	stions) Wiki Page.			
Download	_				
Source code and binary packat Subversion.	ges can be downloaded from the \Rightarrow	project download site. ource co	ode is also available v	ia anonymous	
Installation					
Installation					
Filter Driver Installation					
 Please use the latest release Versions up until 0.1.12. An use later versions (1.1.14.0 recommend users to use the The filters driver in installed Starting with 1.2.2.0, a GUT filter driver to a particular U recommended. Log in as a user with adminis	version. vession. Bugs related to the filter or newer). For 64bit Windows Vistal Iatest relase version available. Jatest relases version available. by a user friendly GUI installer which for installing the filter driver (Filter V 80 device. You can still use the com trator privileges. driver installer (libusb-win32-devel- f driver installer (libusb-win32-devel- driver installer). Do not run the insta	drivers under Vista and Windows 7/2008/2008R2, the version sho makes the install and uninstall giverarly is the preferred way to us mand line install-filter.exe applic ilter-x.x.x.x.zip and then unzip, ilter from an USB storage device,	s 7 and some XP insta uld be 1.2.0.0 or later process easier and m se the filter. It only t ation to install class f or libusb-win32-devel , this is especially imp	Illations. Please . We always ore secure. ries to attach the ilter. But it is not -filter-x.x.x.x.exe ortant for version	s ,

▲図 3-51 libusb-win32のウェブサイト

図 3-52 で示すページに移動するので、最新版のファイルをダウンロードします。

liburb win22 Browco Filos at Source						
sourceforge.net/projects/libusb-win32/f	files/		C ≂ C	🛃 + Google	۹ م	E
OUTCEFOTGE Search	Brows	e Blog Help			Log In	or
LUTION CENTERS Go Parallel HTML5 Cente	r Private Cloud Smarter IT	Events Newsletters				
Free File Opener	Downloa	d Now		IBM Smith		
1) One Disarries at 2 The Takes Haute Dar	Heating 3, Start Hard Tree 2	territoad.	-	Anter and	V	1
ibush wie22						
Summary Files Reviews Wiki Ma	iling Lists Trac Code F	Feature Requests Bugs	Patches	Support Requests Nev	ws Donate]
Summary Files Reviews Wilki Ma Looking for the latest version Download libu Home	iling Lists Trac Code i usb-win32-bin-1.2.6.0.zip (913 Modified :	Feature Requests Bugs	Patches	Support Requests New	ws Donate]
Summary Files Reviews Wilki Ma Looking for the latest version Download libu Home Name 9	iling Lists Trac Code F usb-win32-bin-1.2.6.0.zip (913 Modified †	eature Requests Bugs	Patches	Support Requests New	ws Donate]
Summary Files Reviews Wiki Ma Looking for the latest version Download libr Home Name 9 I libusb win32 releases	illing Lists Trac Code H usb-win32-bin-1.2.6.0.zip (913 Modified # 2012-01-17	eature Requests Bugs	Patches	Support Requests Nev	ws Donate]
Summary Files Reviews Wiki Ma Looking for the latest version Download libr Home Name = Ibusb-win32-releases Ibusb-win32-anapshots	iling Lists Trac Code H usb-win32-bin-1.2.6.0.zip (913 Modified = 2012-01-17 2011-07-13	Eeature Requests Bugs	Patches	Support Requests New	ws Donate]
Summary Files Reviews Wiki Ma Looking for the latest version Download libu Home Name 9 I libusb.win32.releases I libusb.win32.snapshots Totals: 2 Items	iling Lists Trac Code usb-win32.bin-1.2.6.0.zip (913 Modified • 2012-01-17 2011-07-13	Feature Requests Bugs	Patches	Support Requests New	ws Donate	
Summary Files Reviews Wiki Ma Looking for the latest version Download libu Home Name e Ilibusb-win32-releases Ilibusb-win32-releases Ilibusb-win32-releases Ilibusb-win32-snapshots Totals: 2 Items	iling Lists Trac Code f usb win32 bin-1.2.6.0.zip (913 Modified ¢ 2012-01-17 2011-07-13	Feature Requests Bugs	Patches	Support Requests New	ws Donate]
Summary Files Reviews Wilki Ma Looking for the latest version Download libu Home Name ? I libusb-win32-releases I libusb-win32-releases Totals: 2 Items	illing Lists Trac Code I usb-win32.bin-1.2.6.0.zip (913 Modified • 2012-01-17 2011-07-13	Feature Requests Bugs	Patches	Support Requests New	ws Donate	
Summary Files Reviews Wilki Ma Looking for the latest version Download libu Home Name 9 i libusb-win32-releases i libusb-win32-snapshots Totals: 2 Items	illing Lists Trac Code f usb-win32.bin-1.2.6.0.zip (913 Modified • 2012-01-17 2011-07-13	Size +	Patches	Support Requests New	vs Donate	
Summary Files Reviews Wiki Ma Looking for the latest versio Download libu Home Name • Ibusb-win32-releases Ibusb-win32-releases Ibusb-win32-releases Totals: 2 Items	iling Lists Trac Code f usb win32.bin-1.2.6.0.zip (913 Modified • 2012-01-17 2011-07-13	Feature Requests Bugs	Patches	Support Requests New	ws Donate	

▲ 図 3-52 libusb-win32 のダウンロードページ

パソコンと AZPR EvBaord を接続したまま、ダウンロードしたファイルを解凍し、「bin」 フォルダ内の「inf-wizard.exe」を実行します。図 **3-53** で示す「libusb-win32 Inf-Wizard」 ダイアログが表示されます。

libusb-win32 Inf-Wizard	
Information	
This program will create an .inf file for your device.	
Before clicking "Next" make sure that your device is connected to the	e system.
Next >	Cancel

▲ 🖾 3-53 libusb-win32 Inf-Wizard (1 / 3)

[Next] ボタンをクリックすると、図 3-54 で示す画面が表示されます。

elect your d nen either co	evice from the l onnect it or click	ist of detected devices below. If your device isn't liste "Next" and enter your device description manually.
Vendor ID	Product ID	Description
0x0403	0x6010	Dual RS232 (Interface 1)
0x0403	0x6010	Dual RS232 (Interface 0)
•		

▲ 図 3-54 libusb-win32 Inf-Wizard (2 / 3)

ここで、どれか1つデバイスを選択し、[Next] ボタンを押します。図 3-54 では、 「Description」が「Dual RS232 (Interface 1)」のデバイスを選択しています。ダイアログ を進めると、図 3-55 で示す画面が表示されます。

TNI:	libusb-win32 Inf-Wizard		- • ×
	Information		
	A windows driver installat following device:	ion package has been created fo	r the
	Vendor ID:	0x0403	
	Product ID:	0x6010	
	Interface # (MI):	0x01	
	Device description:	Dual RS232 (Interface 1)	
	Manufacturer:	Future Technology Devices Internat	onal, Ltd
	This package contains libusb-w platforms: x86, x64, ia64.	in32 v1.2.6.0 drivers and support for	the following
	Install Now		
			Done

▲ 🖾 3-55 libusb-win32 Inf-Wizard (3 / 3)

ここで [Install Now] ボタンを押し、ドライバをインストールします。

■UrJTAG の起動方法

Windows 7 のスタートメニューから JTAG Shell を実行すると、図 **3-56** で示す画面が 表示されます。「jtag>」というプロンプトに続けて、コマンドが入力可能になります。



▲ 🖾 3-56 JTAG Shell

第3章 プログラミング

■UrJTAG の設定

AZPR EvBoard で使用している xc3s250e と xcf02s を UrJTAG で使うために、設定フ ァイルとパーツリストを追加する必要があります。変更が必要なファイルは本書のサポー トページで公開していますが、ここではどのような変更を行うかを説明します。UrJTAG が「C:¥Program Files (x86)¥UrJTAG¥」にインストールされている前提で説明します。 別のフォルダにインストールした人は、適宜読み替えてください。

まず、「C:\#Program Files (x86)\#UrJTAG\#data\#xilinx\#」に「xc3s250e」というフォ ルダを作成します。この中に「STEPPINGS」と「xc3s250e」という2つのテキストフ ァイルを作ります。なお、このテキストファイルに「.txt」などの拡張子は付けません。 「STEPPINGS」には以下の内容を入力します。

0000 xc3s250e	0
0001 xc3s250e	1
0010 xc3s250e	2
0011 xc3s250e	3
0100 xc3s250e	4
0101 xc3s250e	5
0110 xc3s250e	6
0111 xc3s250e	7
1000 xc3s250e	8
1001 xc3s250e	9
1010 xc3s250e	10
1011 xc3s250e	11
1100 xc3s250e	12
1101 xc3s250e	13
1110 xc3s250e	14
1111 xc3s250e	15

もう1つのファイルの内容を入力するために、BSDLファイルを使います。BSDLファ イルはザイリンクス社のウェブサイトで公開されているため、次の手順でダウンロードし ます。

ウェブブラウザを起動し、ザイリンクス社のウェブサイトを表示します。「今すぐダウ ンロード」を選択し、「デバイスモデル」を選択します。次に、「BSDL Modules」から「Spartan シリーズ FPGA」を選択します。このときに表示されている画面を図 **3-57** に示します。

	com/support/download/index.htm	☆ ▼ C	Soogle P A
		My Account Sign Out 言語の違択 マ	安祥 タすぐダウンロード お問い合わせ キーワードがパーン番号を入力 アドバンスサーチ
		製品 アプリケー・コン サポ	ニート 購入 ザイリンクスについて
ホーム > ダウンロード			
ダウンロー	-ド		
			クイックリンク ● WNDOWS 3264 ビナ (6.2008) ● LAUX 3264 ビナ (6.4068) ● イゼンスに開するヘルフ
デザインツール Model Tpe	デバイスモデル CAE ベンダー ライブラリ		
	F Virtex 3/J = 7" EPGA		
BSDL Models			
BSDL Models	■ Artixシリーズ FPGA		
BSDL Models IBIS-AMI IBISモデル	 Artix シルーズ FPGA Zyng-7000 EPP 		
BSDL Models IBIS-AMI IBIS モデル HSPICE および Eldo モデル	 Artixシルーズ FPGA Zvno-7000 EPP Spartan シリーズ FPGA 		
BSDL Models IBIS-AMI IBIS モデル HSPICE および Eldo モデル パッケージ熱モデル	Attx シリーズ FPGA Zyme-7000 EPP Spartan シリーズ FPGA CPLD		

▲ 図 3-57 BSDL ファイルのダウンロード(1/3)

「Spartan シリーズ FPGA」を選択した後に表示される画面から、「Spartan-3E - BSDL Modules」の「BSDL Models」を選択します。このときに表示されている画面を図 3-58 に示します。

) 🕘 japan.xilinx.	com/support/download/index.htm	☆ マ C 👌	🗸 Google	<u>م</u>	î	8
デザインツール	デバイス モデル CAE ベンダー ライブラリ					
Model Type					*	
BSDL Models	Spartan-3AN - BSDL Models					
IBIS-AMI	BSDL Models (ZIP - 343 KB) MD5 Sum Value: 2006239933889df1e906cc6a503dd7b0	Device Model Type Last Updated	BSDL Models 04/10/2009			
IBISモデル						
HSPICE および Eldo モデル	Spartan-3A - BSDL Models					
バッケージ熱モデル	BSDL Models (ZIP - 479 KB) MD5 Sum Value: 35a95dad3ae8819123ba1b2043a7505e	Device Model Type Last Updated	BSDL Models 04/10/2009			
	Āutomotive BSDL Models (ZIP - 202 K8) MD5 Sum Value: 99257c662a487729dbbdd7c225458689					
	Spartan-3E - BSDL Models					
	BSDL Models (P - 384 KB) 243d8b7eca78f89f386685ebc0da39f	Device Model Type Last Updated	BSDL Models 05/29/2008			
	Automotive BSDL Models (ZIP - 500 KB) MDS Sum Value: 3e62f3b907d33e1751cc4fa2a3ac3bce					
	Spartan-3 - BSDL Models					
	BSDL Models (ZIP - 1.75 MB) MD5 Sum Value: 216b0f2eeac032b635d78fa7e4b7291f	Device Model Type Last Updated	BSDL Models 12/21/2006		Е	
	Automotive BSDL Models (ZIP - 480 KB) MD5 Sum Value: c4f5635cff58ad4ca990f358160ed173					

▲ 図 3-58 BSDL ファイルのダウンロード(2/3)

「BSDL Models」の選択後、ダウンロードが開始します。ダウンロードには、ISE WebPACK をダウンロードしたときと同様にログインが必要になります。ここでダウン ロードしたファイルを解凍し、解凍したファイル内にある xc3s250e.bsd を「C:¥Program Files (x86)¥UrJTAG¥data¥」にコピーします。

次に、JTAG Shell のプロパティの設定をします。JTAG Shell を起動し、タイトルバー を右クリックします。右クリックメニューからプロパティを選択します。JTAG Shell の プロパティを選択している画面を図 **3-59** に示します。



▲ 図 3-59 JTAG Shell のプロパティ選択

JTAG Shell のプロパティでは、図 3-60 で示す画面で簡易編集モードにチェックを入れ、 図 3-61 で示す画面で「画面バッファーサイズ」の「高さ」を「2000」とします。

III "JTAG Shell"のプロパティ	
オプションフォントレイアウト画面の色	
カーツルのサイズ ● 小(S) ● 中(M) ● 大(L) コマンドの履歴 バッファーサイズ(B): 200 バッファーサイズ(B): 200 「愛 閣島編集モード(Q) 「愛 和人モード(J) 現在のコードページ 982 (ANSL/OEM - 日本語 Shift-JJS)	
OK +*>/セル	

▲図 3-60 JTAG Shell のプロパティ (1 / 2)

🔳 "JTAG Shell"のプロパティ		×
オプション フォント レイアウト 画	面の色	
9793775 9477937763-	■回日/9ファーのサイズ - 「幅(W): 高さ(U): ウインドウのサイズ 幅(D): 高さ(E): ウインドウの位置 左から(L): 上から(T): ▼ システム設定を使う(P)	80 (1) 2000 (2) 80 (4) 25 (4) 661 (2) 159 (2) (3)
	OK	キャンセル

▲ 図 3-61 JTAG Shell のプロパティ (2 / 2)

その後、JTAG Shell から、以下のコマンドを実行します。

jtag> bsdl dump xc3s250e.bsd

ここで表示された内容コピーして「xc3s250e」に貼り付けます。JTAG Shell ウィンド ウでは、マウスの左ボタンを押しながら表示された内容を選択し、右ボタンを押すことで コピーできます。

xcf02sの設定ファイルの追加も xc3s250e と同じ要領で行います。「C:¥Program Files (x86)¥UrJTAG¥data¥xilinx¥」に「xcf02s」というフォルダを作成します。この中に、 「STEPPINGS」と「xcf02s」という2つのテキストファイルを作り、「STEPPINGS」に は以下の内容を入力します。

0000	xcf02s	0
0001	xcf02s	1
0010	xcf02s	2
0011	xcf02s	3
0100	xcf02s	4
0101	xcf02s	5
0110	xcf02s	6
0111	xcf02s	7
1000	xcf02s	8
1001	xcf02s	9
1010	xcf02s	10
1011	xcf02s	11
1100	xcf02s	12
1101	xcf02s	13
1110	xcf02s	14
1111	xcf02s	15

「xcf02s」には、同様に JTAG Shell での出力結果を入力します。xcf02s の BSDL ファ イルは、「BSDL Models」から「コンフィギュレーション」を選択し、「Platform Flash BSDL Models」からダウンロードすることができます。このときに表示される画面を図 **3-62** に示します。

reiox マ fイリンクス:ダウ	×−−× +		ر ها ها
🕘 japan.xilinx.c	m/support/download/index.htm	☆ マ C 🎆 צ Google	۹ م
デザインツール	デバイス モデル CAE ベンダー ライブラリ		
Model Type	❀ Kintexシリーズ FPGA		
BSDL Models	● Virtex ジリーズ FPGA		
IBIS-AMI	I Artixシリーズ FPGA		
IBIS モデル	Zynq-7000 EPP		
HSPICE および	Spartan シリーズ FPGA		
ビッケージ林モデル	CPLD		
155 5 Mac 5 10	* 従来製品		
	ヨンフィギュレーション		
	Configuration PROMs - BSDL Models		
(Platform Flash BSDL Models (P - 150 KB) MDS Sum Value: 153adf8c23d68k5 frca4037a918c676	Device Model Type BSDL Models Last Updated 02/29/2008	
	XC18V00 BSDL Models (z)P - 128/8) MD5 Sum Value: 7d20856ca657e2a3d7e6ecb6f70f1355		
	System ACE - BSDL Models		
	BSDL Models (2)P - 45.7 KB) MDS Sum Value: 2c964605029616cc750080abSca1aa60	Device Model Type BSDL Models Last Updated 11/10/2002	

▲ 図 3-62 BSDL ファイルのダウンロード(3/3)

ダウンロードしたファイルを解凍し、解凍したファイル内にある xcf02s.bsd を 「C:¥Program Files (x86)¥UrJTAG¥data¥」コピーします。その後、JTAG Shell から、 以下のコマンドを実行し、表示された内容コピーして「xcf02s」に貼り付けます。

jtag> bsdl dump xcf02s.bsd

最後に、「C:¥Program Files (x86)¥UrJTAG¥data¥xilinx¥PARTS」に以下の記述を書 き足すことで、パーツリストに xc3s250e の追加を行います。

0001110000011010	xc3s250e	xc3s250e

xcf02s はすでにパーツリストに記述されているので追加の必要はありません。

■FPGA のコンフィギュレーション方法

FPGA のコンフィギュレーション方法を説明します。パソコンと AZPR EvBoard を USB ケーブルで繋いで、JTAG Shell を起動します。以下のコマンドを JTAG Shell に入 力して実行すると、FPGA を認識します。

jtag> cable jtagkey
jtag> detect

detect コマンドを実行したとき、以下のように表示されます。

```
IR length: 14
Chain length: 2
Device Id: 00010001110000011010000010010011 (0x0000000011C1A093)
 Manufacturer: Xilinx
  Part(0):
                  xc3s250e
 Stepping:
  Filename:
              c:\program files (x86)\urjtag\data/xilinx/xc3s250e/xc3s250e
Device Id: 111101010000010001010000100100101 (0x0000000F5045093)
 Manufacturer: Xilinx
  Part(1):
                  xcf02s
 Stepping:
               15
  Filename:
               c:\program files (x86)\urjtag\data/xilinx/xcf02s/xcf02s
```

ここで表示されているように、xc3s250e が part0、xcf02s が part1 になります。 BIT ファイルを選択して SVF ファイルを作成したときは、以下のコマンドを実行して、 xc3s250e を選択します。

jtag> part 0

MCS ファイルを選択して SVF ファイルを作成したときは、以下のコマンドを実行して、 xcf02s を選択します。

jtag> part 1

SVF ファイル作成で作成した SVF ファイルが「D:¥sample.svf」にある場合は、以下のようにファイルパスを指定してコマンドを実行することで FPGA のコンフィギュレー

ションが開始します。

jtag> svf D:¥sample.svf

以下のように progress オプションをつけると、コンフィギュレーションの進捗が表示 されます。

jtag> svf D:¥sample.svf progress

cblsvr-0.1_ft2232

AZPR EvBoard の FPGA のコンフィギュレーション方法として、fenrir 氏によって作成された「cblsvr-0.1_ft2232」というツールを使うこともできます。

cblsvr-0.1_ft2232

COLUMN

http://fenrir.naruoka.org/archives/000644.html

このツールは、ザイリンクス社のコンフィギュレーションツールである「iMPACT」と組み合わせて、AZPR EvBoard に搭載されている FT2232 経由でコンフィギュレーションを行うことができます。この方法は、BIT ファイルや MCS ファイルを使うため、SVF ファイルを作る必要はありません。使い方は上記 URL を参照してください。

ただし、fenrir氏に確認をしましたが、cblsvr-0.1_ft2232はISE 11までしか動作確認で きていません。最新版のISEでは動作しない可能性があります。また Windows 7はISE 12 からの対応のため、ISE 11以前のバージョンを使う場合は、Windows XPのパソコンを用意 する必要があります。

3.2.5 クロスアセンブラ

アセンブラはアセンブリ言語で書かれているプログラムを、機械語に変換するソフトウ ェアです。CPUが実行できる機械語は数字の列で表現されるため、人間にとっては非常 にわかりにくくなっています。そのため、機械語の命令に1対1に対応し、人間が理解し やすいように記号化されたニーモニックと呼ばれる命令語を使います。

アセンブリ言語でプログラムを作成し、アセンブラを使って機械語に変換をするという 手順で開発を進めます。アセンブリ言語と機械語の対応を図 **3-63** に示します。



▲ 図 3-63 アセンブリ言語と機械語

■インストール

アセンブリ言語は機械語と1対1対応なので、CPUのアーキテクチャに依存します。 AZ Processor は独自の命令セットのため、対応したアセンブラを用意しました。AZ Processor のアセンブラなので AZPR ASM という名前にしました。本書のサポートペー ジからダウンロードしてください。

ダウンロードしたら、「C:¥azpr¥azprasm¥」というフォルダを作成して、アセンブラ 本体である「azprasm.exe」というファイルをその作成したフォルダの中に置いてくださ い。次に Windows 7 のスタートメニューを表示させて、「コンピューター」を右クリック し「プロパティ」を選択します。左側の「システムの詳細設定」を選択して、図 3-64 に 示す「システムのプロパティ」ダイアログを表示します。

システムのプロパティ
コンピューター名 ハードウェア 詳細設定 システムの保護 リモート
Administrator としてログオンしない場合は、これらのほとんどの変更はできません。 パフォーマンス
視覚効果、プロセッサのスケジュール、メモリ使用、および仮想メモリ
[段定(S)
2-ザーブロファイル ログオンに関連したデスクトップ設定
起動と回復 システム起動、システム障害、およびデバッグ情報
設定(1)
環境変数(<u>M</u>)
OK キャンセル 適用(A)

▲図3-64 システムのプロパティ

「システムのプロパティ」ダイアログの「詳細設定」タブから、[環境変数] ボタンをク リックします。「環境変数」ダイアログを図 **3-65** に示します。

変数	値
TEMP	%USERPROFILE%¥AppData¥Local¥Temp
TMP	%USERPROFILE%¥AppData¥Local¥Temp
	新規(<u>N)</u> 編集(<u>E</u>) 削除(<u>D</u>)
ステム環境変数(変数	新規(1 <u>)</u> 編集(E) 削除(D) S) (值
ステム環境変数(変数 ComSpec	新規(1) 編集(E) 前期(D) S) 値 C:¥Windows¥system32¥cmd.exe
ステム環境変数(変数 ComSpec FP NO HOST C	新規(1)
ステム環境変数(変数 ComSpec FP_NO_HOST_C NUMBER OF P	新規(1)) 編集(E) 前時(D) ⑤ 値 C¥Windows¥system32¥cmd.exe HE NO POC 4

▲ 図 3-65 環境変数

「環境変数」ダイアログが表示されたら、ユーザ環境変数の[新規] ボタンをクリック します。図 3-66 に示す「新しいユーザ変数」ダイアログが表示されるので、[変数名] に 「Path」と入力し、[変数値] に「azprasm.exe」を置いたファイルパスを入力します。

第3章 プログラミング

新しいユーザー変数	x
変数名(<u>N</u>): 変数値(V):	Path C¥azpr¥azprasm
	OK キャンセル

▲ 図 3-66 新しいユーザ変数

本書のとおりに進めている場合は、以下の文字列を入力します。

C:¥azpr¥azprasm

「ユーザ環境変数」に「Path」という変数名がすでに存在する場合は、[編集] ボタン をクリックし、「;」で区切ってから「azprasm.exe」を置いたファイルパスを付け加えます。 たとえば、「Path」の変数値が「C:¥hoge」となっていた場合は、変数値を以下のように 書き加えます。

C:¥hoge;C:¥azpr¥azprasm

Path の設定が終わると、アセンブラのインストールは完了です。コマンドプロンプト を起動します。「azprasm」と入力して実行してください。以下のように Usage が表示さ れていれば、正しくインストールできています。

```
C:¥Users¥respon>azprasm
Usage: azprasm [ -o outfile ] infile
```

もし、以下のように表示される場合は設定が間違っているため、アセンブラがインスト ールできていません。

```
C:¥Users¥respon>azprasm
'azprasm' は、内部コマンドまたは外部コマンド、
操作可能なプログラムまたはバッチ ファイルとして認識されていません。
```

「azprasm.exe」が置いてあるフォルダ名や、環境変数の設定が正しく行われているか 確認してください。

■使い方

コマンドプロンプトを起動し、ソースコードを指定して「azprasm」コマンドを実行す ると、機械語に変換されたファイルを出力します。「azprasm」コマンドのオプションを **表 3-5** に示します。

▼表3-5 azprasm コマンドのオプション

オプション	説明
-o outfile	outfile に出力するバイナリファイルの名前を指定します。
-p prgfile	prgfile で指定した名前で PRG ファイルを作成します。
coe coefile	coefile で指定した名前で COE ファイルを作成します。ハイフンが 2 つ必要なこと に注意してください。

「-o」オプションは、出力するバイナリファイルの名前を指定します。指定しない場合は、 出力ファイルの名前は「outfile」となります。「-p」オプションは、指定した名前で「PRG ファイル」を作成します。「PRG ファイル」は第1章で紹介された iverilog でシミュレー ションを行うときに使用するメモリの初期値を指定するファイルです。詳細は 1.4.3 項の 「メモリイメージの読み込み」を参照してください。「-p」オプションを指定しない場合は、 「PRG ファイル」は作成されません。「--coe」オプションは、指定した名前で「COE フィ アル」を作成します。「COE ファイル」はブロック RAM の初期値を指定するテキスト形 式のファイルです。ISE Project Navigator から呼び出される「Block RAM Generator」 ダイアログで使用します。詳細は 3.2.3 項の「BIT ファイルの作成」を参照してください。 「--coe」オプションを指定しない場合は、「COE ファイル」は作成されません。

以下にコマンドプロンプトへの入力例を示します。このコマンドは、ソースコードに 「sample.asm」をソースコードとして AZPR ASM で機械語に変換し、「sample.bin」とい う名前の出力ファイルを生成します。

C:¥Users¥respon>azprasm -o sample.bin sample.asm

第3章 プログラミング

■プログラムの書式

■ニーモニック

AZPR ASM のニーモニック一覧を**表 3-6** に示します。各命令の詳細は、本書のサポート ページからダウンロードできる「AZ Processor Specification Sheet」を参照してください。

▼表3-6 ニーモニック一覧

種類	命令
論理演算命令	ANDR, ANDI, ORR, ORI, XORR, XORI
算術演算命令	ADDSR, ADDSI, ADDUR, ADDUI, SUBSR, SUBUR
シフト命令	SHRLR, SHRLI, SHLLR, SHLLI
分岐命令	BE, BNE, BSGT, BUGT, JMP, CALL
メモリ命令	LDW, STW
特殊命令	TRAP
特権命令	RDCR, WRCR, EXRT

■ディレクティブ

ディレクティブとは、アセンブラに対する指示を行うための記述です。AZPRASMの ディレクティブー覧を**表 3-7**に示します。

▼表3-7 ディレクティブ一覧

ディレクティブ	機能
LOCATE	プログラムの先頭番地の変更
EQU	シンボルの設定
high	アドレスの 16 ビット目から 31 ビット目までの値を 16 ビットの整数に変換します。
low	アドレスの 0 ビット目から 15 ビット目までの値を 16 ビットの整数に変換します。

LOCATE

LOCATE は、プログラムの先頭アドレスの変更を行います。たとえば、プログラムに 以下の設定することで、プログラムの先頭アドレスを 0x20000000 番地に変更します。

LOCATE 0x20000000

プログラムの先頭アドレスを変更した様子を図 3-67 に示します。



▲ 図 3-67 プログラムの先頭番地を変更した様子

• EQU

EQUは、シンボルの設定を行います。シンボルとは、プログラム中に出てくる値を文 字列に置き換えたものです。シンボルを使うことで、プログラムを読むときに、命令を実 行している対象がわかりやすくなります。以下に EQU の記述例を示します。

SYMBOL EQU 100

このように記述することによって、プログラム中の SYMBOL という文字列が 100 という値と同じ意味を持ちます。シンボルを使った記述の例を以下に示します。

ADDUI r0,r1,SYMBOL

上記の命令は、以下の命令と同じ意味になります。

ADDUI r0,r1,100

• high

high は、アドレスの 16 ビット目から 31 ビット目までの値を 16 ビットの数値に変換し ます。ラベルに対しても使うことができます。以下に high の記述例を示します。

high(LABEL)

high による値の変換の様子を図 3-68 に示します。



▲ 図 3-68 high による値の変換

• low

low は、アドレスの0ビット目から15ビット目までの値を16ビットの数値に変換しま す。ラベルに対しても使うことができます。以下にlow の記述例を示します。

low(LABEL)

low による値の変換の様子を図 3-69 に示します。



▲ 図 3-69 low による値の変換

■ラベルの形式

命令の存在するメモリアドレスにラベルを付けることができます。ラベルの文字列に続いて「:」を記述します。以下にラベルの記述例を示します。「XORR r0,r0,r0」という命令が格納されているアドレスに対して、LABEL というラベルを指定しています。

LABEL: XORR r0,r0,r0

ラベルはディレクティブの high や low の引数や、分岐命令で指定する分岐先のアドレスの代わりに指定することができます。

■命令の形式

命令は、ニーモニックとオペランドで構成されます。ニーモニックとオペランドの間に は1文字以上の半角スペースかタブが必要になります。

● ニーモニック

ニーモニックとして指定できる命令は、表 3-6 を参照してください。

● オペランド

オペランドには、命令の操作対象となるデータを記述します。オペランドとして、ラベル、汎用レジスタ、CPU 制御レジスタ、定数を指定することができます。

ラベルには、プログラム内で使用した文字列を記述します。ラベルが16ビット以上の 値をとるときは、ディレクティブのhighや、lowを使って記述します。汎用レジスタは r0~r31で指定します。rの後に汎用レジスタの番号を記述します。CPU制御レジスタ はc0~c7で指定します。cの後にCPU制御レジスタのRegister Addressを記述します。 定数には、整数定数と文字定数があります。整数定数は、8進数と10進数と16進数で記 述することができます。接頭語に0をつけると8進数になります。接頭語に0xをつける と16進数になります。文字定数は「」で括って記述します。整数定数の表記を**表 3-8** に 示します。

▼表 3-8 整数定数の表記

進数	例
8	0173
10	123
16	0x7B

以下に命令の記述例を示します。

LABEL:		
	XORR	r0,r0,r0
	ADDUI	r0,r1,0x100
	BE	r0,r1,LABEL

■コメントの形式

コメントはプログラムの実行に関係しない文です。「;」以降の文はコメントとみなされ ます。

3.2.6 最初のプログラム

最初のプログラムとして LED を制御するプログラムを作成します。LED は、AZ Processor の GPIO Output Port レジスタに割り当てられています。GPIO の制御レジス タは、メモリマップされており、ベースアドレスは 0x80000000 です。図 3-70 と表 3-9 に、GPIO Output Port レジスタの詳細を示します。

31			(
	OUT	_DATA	
31:0	Output Port Data (OUT_DATA)	Default : 0x0	

▲ 図 3-70 GPIO Output Port レジスタのビットマップ

▼表 3-9 GPIO Output Port レジスタの詳細

ビット	詳細
0~15	7 セグメント LED の制御を行う。
16	LED1 の制御を行う。
17	LED2 の制御を行う。
18~31	割り当てなし。

オフセットが 0x4 なので、GPIO Output Port レジスタはメモリアドレスの 0x80000004 番地にマップされています。AZPR EvBoard 上に搭載されている LED1 は GPIO Output Port レジスタのビット 16、LED2 はビット 17 に割り当てられています。ビット 0 からビ ット 15 は 7 セグメント LED に割り当てられています。7 セグメント LED は、3.6 節で解 説するので、ここでの解説は省略します。ビット 18 からビット 31 までは、どのデバイス にも割り当てられていません。

GPIO Output Port は負論理なので、値を0にするとLED が点灯し、1にするとLED が消灯します。LED1 を点灯し、その他のLED を消灯するプログラムを**リスト 3-1** に示します。

▼ リスト 3-1 LED 制御プログラム (led.asm)

1	;;;;	シンボルの	の定義		
2	GPI	D_BASE_AD	DDR_H	EQU 0×8000	;GPIO Base Address High
3	GPI	D_OUT_OFF	SET	EQU 0x4	;GPIO Output Port Register Offset
4					
5	;;;	LED点灯			
6		XORR	r0,r0,r0	Ð	
7		ORI	r0,r1,GF	PIO_BASE_ADDR_H	;GPIO Base Address上位16ビットをr1にセット
8		SHLLI	r1,r1,16	5	;16 ビット左シフト
9		ORI	r0,r2,0>	<2	;出力データを上位16ビットをr2にセット
10		SHLLI	r2,r2,16	5	;16 ビット左シフト
11		ORI	r2,r2,0>	KFFFF	;出力データを下位16ビットをr2にセット
12		STW	r1,r2,GF	PI0_OUT_OFFSET	;GPIO Output Portに出力データを書き込む
13					
14	;;;;	無限ルーフ	プ		
15	LOOF	P:			
16		BE	r0,r0,L0)0P	;L00Pに戻る
17		ANDR	r0,r0,r0	9	;NOP

■シンボルの定義

プログラムの最初にシンボルの定義を行っています。プログラム中で GPIO Output Port レジスタにアクセスするために、GPIO 制御レジスタのベースアドレスと、GPIO Output Port レジスタのオフセットを定義しています。

■LED 制御

まず、r0 に 0 をセットしています。r0 は常に 0 が格納されているレジスタとして使い ます。今後もプログラムの先頭で記述するので覚えておいてください。7 行目から 8 行目で、 r1 に GPIO 制御レジスタのベースアドレスである 0x80000000 を格納しています。AZ Processor では即値として指定できる値は 16 ビットなので、図 3-71 で示すように GPIO 制御レジスタのベースアドレス上位 16 ビットである 0x8000 を r1 に格納して、SHLLI 命 令で16ビット左シフトを行っています。



▲ 図 3-71 ベースアドレスをセットする命令とr1の値

9 行目から 11 行目で、GPIO に出力するデータとして r2 に 0x2FFFF を格納しています。 LED1 に割り当てられているビット 16 が 0、LED2 と 7 セグメント LED に割り当てられ ているビット 0 から 15 までとビット 17 は 1 となっています。12 行目で、STW 命令によ って GPIO Output Port レジスタである 0x80000004 番地に r2 の値を書き込んでいます。

■無限ループ

GPIO へのアクセスが完了すると、これ以上命令を実行する必要はありません。しかし、 AZ Processor はクロックごとに次の命令を読み込もうとします。この以上命令を実行し ないように、BE 命令でラベル LOOP となっているところに戻る無限ループを行っていま す。このような終了処理をダイナミックエンドと言います。AZ Processor では、BE 命令 を含む分岐命令の直後は遅延スロットとなっています。遅延スロットの命令は、直前の分 岐命令による分岐が行われるかどうかに関わらず、必ず実行されます。本書では、分岐命 令の次は NOP として ANDR r0,r0,r0 を明示的に書いています。

NOP というのは、No Operation の略で何もしない命令ということです。命令の意味を 考えてもらえばわかると思いますが、r0 と r0 の論理積を計算して r0 に格納しています。 結果として r0 は ANDR 命令実行前と変わらない値になります。 AZPR EvBoard を使って動作確認をします。まずは、先ほど説明した LED 制御プログ ラムのソースコードを作成します。テキストエディタを起動し、**リスト 3-1** どおりにプロ グラムを入力し、テキストファイルとして保存します。ここでは、LED 制御プログラム を記述したテキストファイルの名前を「led.asm」とします。

次に、アセンブラを使ってソースコードを変換します。コマンドプロンプトを起動し、 ファイルを作成した場所に移動します。ソースコードが「D:¥azpr¥program¥」にある 場合は、以下のように入力することで移動することができます。

C:¥Users¥respon>D:	
D:>cd azpr¥program	
D:¥azpr¥program>	

アセンブラを実行して、ソースコードを変換するには、以下のコマンドを入力します。

D:¥azpr¥program>azprasm led.asm -o led.bin --coe led.coe

「led.asm」が置いてあるフォルダと同じ場所に AZ Processor で実行させる機械語のフ ァイル「led.bin」と ISE Project Navigator の「Block Memory Generator」ダイアログで ブロック RAM の初期化を行うための形式になっている「led.coe」が作成されます。AZ Processor で実行させる機械語のファイルを BIN ファイルと呼ぶこととします。今回の手 順では、プログラムは COE ファイルを使って AZ Processor の ROM 領域の初期化を行 うため、「led.bin」は使いません。

次に、BIT ファイルを作成します。BIT ファイルを作成する手順については、3.2.3 項 の「BIT ファイルの作成」を参照してください。「x_s3e_sprom」を作成するときに、先 ほどアセンブラで作成した「led.coe」を読み込ませます。「Block Memory Generator」ダ イアログの画面で、図 3-72 のように「Memory Initialization」の[Load Init File] にチ ェックを入れ、[Browse] ボタンを押して、「led.coe」を指定します。

iew		
Symbol	المعادة المعادة المعامة المعادة معادة المعادة معادة	xilinx.com:ip:blk_mem_gen:7
ADDRA[10:0] ENA RECICA RECICA RECICA RESCA CUX NUECTORITERR NUECTORI NUECTORITERR NUECTORI NUECTORITERR NUECTORI NU	Copional Output Registers Port A Register Port A Output of Memory Primitives Register Port A Output of Memory Core Register Port A Input of SoftECC logic Use RECCER Pin (separate enable pin for Port A output registers) Pipeline Stages within Mux Memory Initialization Dece File D:program\led.coe Fill Remaining Memory Locations Remaining Memory Locations (Hex) 0	xilinx.com:ip:bit_mem_gen:

▲ 🖾 3-72 Block Memory Generator

BIT ファイルから SVF ファイルを作成します。3.2.3 項の「SVF ファイル作成」の手順に従って「led.svf」を作成してください。iMPACT の画面の Create SVF File では「led.svf」を指定し、「Add Device」ダイアログで先程作成した「led.bit」を選択します。 Device 上で Program を選択したら、SVF ファイル「led.svf」が作成されます。

最後に、UrJTAGでSVFファイルを再生します。まず、AZPR EvBoardの電源を入れて、 USB ケーブルでパソコンと接続します。パソコンにデバイスが認識されたら、UrJTAG を起動して 3.2.4 項の「FPGA のコンフィギュレーション方法」で説明した手順でコンフ ィギュレーションを行います。

jtag> cable jtagkey jtag> detect jtag> part 0 jtag> svf led.svf progress

以上の作業で、メモリの ROM 領域にプログラムが書きこまれた AZ Processor のコ ンフィギュレーションが完了しました。これまでの手順が正しく行えているならば、AZ Processor がプログラムを実行して、**写真 3-3**のように LED1 が点灯し、他の LED が消 灯します。



▲ 写真 3-3 LED 制御プログラム実行の様子

次の節からは、これまでに説明したツールを利用して開発を進めます。そのため、ここ でプログラムが動作していることを確認しておくことは非常に重要になります。もし期待 どおりの結果にならなかった場合は、もう一度この節を戻って手順を確認してください。

3.3 シリアル通信

本節では、シリアル通信プログラムについて説明します。AZPR SoC に実装されている UART でパソコンと通信を行い、パソコンの画面に文字を表示させます。

シリアル通信の規格である RS-232(シリアルポートとも呼ばれる)は、パソコンのマ ザーボードにも搭載されており、周辺機器などの接続で広く使われています。現在は、徐々 にシリアルポートが搭載されているマザーボードの数が減ってきており、ノートパソコン にはほとんど搭載されていません。

AZPR EvBoard には USB シリアル変換 IC が搭載されており、パソコンと USB ケーブ ルで接続することで、シリアル通信ができるようになっています。そのため、シリアルポ ートが搭載されていないパソコンでも AZPR EvBoard とシリアル通信を行うことができ ます。

3.3.1 Tera Term のインストール

パソコン側にはシリアルポートの入出力を行うための端末エミュレータが必要となり ます。シリアルポートから入力された文字はこの端末エミュレータ上に表示されます。 Windows 7 には、Tera Term という有名な端末エミュレータがあるので、これを利用し ます。Tera Term は以下のウェブページからダウンロードできます。

Tera Term

http://sourceforge.jp/projects/ttssh2/releases/

ウェブブラウザで上記 URL にアクセスし、図 **3-73** で示すリンクから最新版のインス トーラをダウンロードします。ファイル名に「exe」が含まれている方がインストーラです。 ダウンロードしたファイルを実行するとインストーラが起動します。インストーラに従 って進めていけばインストールすることができます。

3.3

Firefox -					- • ×
● ダウンロード - Tera Term (テラター… +					
Sourceforge.jp/projects/ttssh2/releases/			☆ ≂ C	f Google	۹ 🗈 ۲
📕 Tera Term					
概要 - ニュース - ダウンロード - ソースコード - Wiki -	文書 マフォ	ーラム - メーリングリスト - チケット	· •		
ダウンロード			2	Robons 8/Virosi I	Pradio 2012 (\$1,000)
Tera Term は、オリジナルの Tera Term Pro 2.3 @ の原作者公認の後継版です。オープンソースで開 発されており、UTF-8 表示に対応しています。ま た、SSH 対応モジュール TTSSH を拡張し、SSH2 ブロトコルをサポートしています。	このブロジュ 会会会会 ッソイート {1	ェクトは星いくつ? 注意 I2 『いいね! く55 夏+1 く 1	19	2-3-0	PA-14
● Tera Termプロジェクトのリリースは下のリリース:	ファイル一覧が	からダウンロードできます。		1331)	10 Para
		3		-1/1-1.02	L-1127
ンステム要件 オペレーティングシステム: Windows 95/98/ME Wind	lowe NT/200	0. Windows XP		7-24-22740	
				Number 1990 Duration 199 Date 1980 Distant 1980	10,000 10 14 (1,140)
リリースファイル一覧				Contraction in the	
パッケージ/リリース/ファイル	サイズ	日付/ダウンロード数		F. Tara Tarm. (1921)	
▼ Tera Term				Multiplication of	
▼4.75-[リリースノート/変更履歴]		2012-08-31 17:30	_	I substrate in 1	
eraterm=4.75.exe	11.0 MB	19296		(14Pmar (40))	
e teraterm-4.75.zip	6.4 MB	5114		A Reculation (MI)	

▲ 図 3-73 Tera Term のダウンロードページ

3.3.2 プログラムの作成

シリアル通信を使って、文字データを送信するプログラムを作成します。文字はプログ ラムの学習でお馴染みの「Hello,world.」にします。まずは UART のバッファクリアを行 います。次に、文字データを1文字ずつ送信します。これらの処理を行うために使用する UART 制御レジスタのビットマップを図 3-74、図 3-75 に示します。

	Access : Read/Write	Offset : 0x0	
31			2 1 0
	Reserv	ved	TxI Rx
1	Transmit Interrupt (TxI)	Default : 0x0	
1 デー	 Transmit Interrupt(Txl) ·タの送信が完了した場合にセットされる。この bit ;	Default:0x0 が1の場合は送信割込みが発生する。	
1 デー 0	Transmit Interrupt (TxI) タの送信が完了した場合にセットされる。この bit ; Receive Interrupt (RxI)	Default:0x0 が1の場合は送信割込みが発生する。 Default:0x0	

▲図 3-74 UART Status レジスタのビットマップ

-	Acces	s : Read/Write		Offset : 0x4			
31				8	37		0
		Res	served			DATA	
7:0		Data (DATA)		Default : 0x0			
この	レジスタ	に書き込んだ場合データを送信	する。このし	レジスタを読み出した場合受信	データな	を読み出す。	

▲ 図 3-75 UART Data レジスタのビットマップ

■バッファクリア

シリアル通信は電気信号なので、ケーブルの接続時などにハードウェアのノイズが信号 に乗ってしまい、UARTの送受信データ領域(バッファ)に余計なデータが入ってしま うことがあります。余計なデータを取り除くため、データ送受信を行う前に、バッファの クリアを行います。バッファクリアのフローチャートを図 **3-76** に示します。



▲図3-76 バッファクリアのフローチャート

UART Status レジスタからステータスの読み出しを行います。UART Status レジスタ の Receive Interrupt ビットは、データを受信したかどうかを示します。ステータスがデ ータを受信したことを示していた場合は、UART Data Register からデータを読み出しま す。その後、UART Status Register の Receive Interrupt ビットをクリアします。ステー タスが受信していないことを示していた場合は、バッファクリアの処理が終了となります。

■データ送信

データ送信のフローチャートを図 3-77 に示します。



▲ 図 3-77 データ送信のフローチャート

UART Data Register に送信データを書き込みます。次に、UART Status Register か らステータスの読み出しを行います。UART Status Register の Transmit Interrupt ビ ットは送信が完了したかどうかを示します。送信が完了したことを示すまで、繰り返し ステータスを読み出します。UART Status Register が送信完了したことを示した場合、 Transmit Interrupt ビットのクリアを行い、データ送信処理を終了します。このように処 理が完了していないか定期的に問い合わせを行う処理方式をポーリングと言います。

シリアル通信で文字データを送信するプログラムでは、ここで紹介したバッファクリア とデータ送信を行います。実際のプログラムを**リスト 3-2** に示します。

第3章 プログラミング

▼ リスト 3-2 シリアル通信で文字を出力するプログラム (serial.asm)

1 ;;; シンボルの定義 2 UART BASE ADDR H EQU 0x6000 ;UART Base Address High 3 UART STATUS OFFSET EQU 0x0 ;UART Status Register Offset 4 UART DATA OFFSET EQU 0x4 ;UART Data Register Offset 5 UART RX INTR MASK EOU 0x1 ;UART Receive Interrupt Mask 6 UART TX INTR MASK EQU 0x2 ;UART Transmit Interrupt Mask 7 8 9 XORR r0.r0.r0 10 r0,r1,high(CLEAR_BUFFER) 11 ORI ;CLEAR BUFFERの上位16ビットをr1にセット 12 SHLLI r1,r1,16 13 ORI r1,r1,low(CLEAR BUFFER) ;CLEAR BUFFERの下位16ビットをr1にセット 14 15 ORI r0,r2,high(SEND CHAR) ;SEND CHARの上位16ビットをr2にセット 16 SHLLI r2, r2, 16 17 ORI ;SEND CHARの下位16ビットをr2にセット r2,r2,low(SEND CHAR) 18 19 ;;; UARTバッファクリア 20 CALL ;CLEAR BUFFER呼び出し r1 21 ANDR r0,r0,r0 ;NOP 22 23 ;;; 文字表示 24 ORI 25 r0,r16,'H' ;r16に'H'をセット CALL 26 r2 ;SEND CHAR呼び出し 27 ANDR r0,r0,r0 :NOP 28 29 ORI r0,r16,'e' ;r16に'e'をセット 30 CALL r2 ;SEND CHAR呼び出し 31 ANDR r0,r0,r0 :NOP 32 33 ORI r0,r16,'l' ;r16に'し'をセット r2 34 CALL ;SEND CHAR呼び出し 35 ANDR ;NOP r0,r0,r0 36 37 ORI r0,r16,'l' ;r16に'し'をセット 38 CALL r2 ;SEND CHAR呼び出し 39 ANDR r0,r0,r0 :NOP 40 41 ORI r0,r16,'o' ;r16に'o'をセット 42 CALL r2 ;SEND CHAR呼び出し 43 ANDR r0,r0,r0 ;NOP 44 45 ORT r0,r16,',' ;r16に','をセット 46 CALL r2 ;SEND CHAR呼び出し 47 ANDR r0,r0,r0 ;NOP 48 49 ORT r0,r16,'w' ;r16に'w'をセット CALL 50 r2 ;SEND CHAR呼び出し
シリアル通信

3.3

51		ro ro ro		NOD	
52	ANDIX	10,10,10	,	NOT	
53	ORI	r0.r16.'o'	:	r16に'o'をセット	
54	CALL	r2	;	SEND CHAR呼び出し	
55	ANDR	r0,r0,r0	;	NOP	
56					
57	ORI	r0,r16,'r'	;	r16に'r'をセット	
58	CALL	r2	;	SEND_CHAR呼び出し	
59	ANDR	r0,r0,r0	;	NOP	
60	ODT	m0 m10 111			
62	ORI	r0,r10, 'l'	;		
63		$r_0 r_0 r_0$,		
64	ANDIX	10,10,10	,	.1101	
65	ORI	r0.r16.'d'	:	r16に'd'をセット	
66	CALL	r2	;	SEND CHAR呼び出し	
67	ANDR	r0,r0,r0	;	NOP	
68					
69	ORI	r0,r16,'.'	;	r16に'.'をセット	
70	CALL	r2	;	SEND_CHAR呼び出し	
71	ANDR	r0,r0,r0	;	NOP	
72					
73	;;; 無限ルー	·)			
74	RF	r0 r0 100P		毎限ループ	
76		r0, r0, r0	,		
/0	FINDIN	10,10,10	,		

■シンボルの定義

UART 制御レジスタのベースアドレスとアクセスするレジスタのオフセットを定義 しています。また、UART Status レジスタの Receive Interrupt ビットと Transmit Interrupt ビットのマスクするための値を定義しています。

■サブルーチンコールの設定

プログラム中でアクセスするサブルーチン他の処理から呼び出せるように、汎用レジス タにラベルの値を格納しています。

rl にはラベル CLEAR_BUFFER の値を、r2 にはラベル SEND_CHAR の値を格納しています。

■UART バッファクリア:

rlをオペランドとして CALL 命令を実行し、CLEAR_BUFFER サブルーチンを呼び出 しています。CALL 命令の次は遅延スロットとなるため、NOP で埋めています。プログ ラムを最適化する場合は、ここは NOP ではなくてもよいのですが、本書ではプログラム の可読性と説明の容易性のため、最適化は行っていません。

■文字データ送信

r16 に送りたい文字を入れて、SEND_CHAR サブルーチンを呼び出しています。1 文字 ずつの送信のため、25 行目から 71 行目で、文字データをセットし、SEND_CHAR サブ ルーチンを呼び出す処理を繰り返しています。

■無限ループ

プログラムの終了処理を行っています。LED 制御のプログラムと同じく、この行以降 の命令を読み込まないように、ラベル LOOP に戻る無限ループとなっています。

■CLEAR_BUFFER サブルーチン

CLEAR_BUFFER サブルーチンをリスト 3-3 に示します。

▼ リスト 3-3 CLEAR_BUFFER サブルーチン (serial.asm)

```
78 CLEAR BUFFER:
79
        ORI
                r0,r16,UART BASE ADDR H
                                          ;UART Base Address上位16ビットをr16にセット
80
        SHLLI r16, r16, 16
81
82
    CHECK UART STATUS:
83
        LDW
                r16,r17,UART STATUS OFFSET ;STATUSを取得
84
85
        ANDI
                r17, r17, UART RX INTR MASK
86
        BE
                r0,r17, CLEAR BUFFER RETURN ;Receive Interrupt bitが立っていなければ
                                            CLEAR BUFFER RETURNを実行
        ANDR
                                           :NOP
87
                r0,r0,r0
88
89
    RECEIVE DATA:
90
        LDW
                r16,r17,UART DATA OFFSET
                                          ;受信データを読んでバッファをクリアする
91
92
        LDW
                r16.r17.UART STATUS OFFSET :STATUSを取得
93
        XORI
                r17,r17,UART RX INTR MASK
94
        STW
                r16,r17,UART STATUS OFFSET
                                          ;Receive Interrupt bitをクリア
95
96
        BNE
                r0,r0, CHECK UART STATUS
                                           ; CHECK UART STATUSに戻る
97
        ANDR
                r0,r0,r0
                                           ;NOP
    CLEAR BUFFER RETURN:
98
99
        JMP
                r31
                                           ;呼び出し元に戻る
100
        ANDR
                r0,r0,r0
                                           :NOP
```

CLEAR_BUFFER

79 行目から 80 行目で、r16 の値に UART 制御レジスタのベースアドレスである

0x60000000を格納しています。83行目で、UART Status レジスタの値をr17に格納します。 85 行目から 86 行目で、r17 に格納した UART Status レジスタの Receive Interrupt ビッ トが1となっていないか確認し、0の場合はラベル_CLEAR_BUFFER_RETURN に分岐 します。1の場合は、90 行目で UART Data レジスタを読み出し、92 行目から 94 行目で UART Status レジスタの Receive Interrupt ビットを0クリアします。その後、96 行目 でラベル_CHECK_UART_STATUS に戻ります。ラベル_CLEAR_BUFFER_RETURN では、99 行目で CLEAR_BUFFER サブルーチンの呼び出し元に戻ります。

■SEND_CHAR サブルーチン

SEND_CHAR サブルーチンをリスト 3-4 に示します。

▼ リスト 3-4 SEND_CHAR サブルーチン (serial.asm)

103	SEND_CHAR:		
104	ORI	r0,r17,UART_BASE_ADDR_H	;UART Base Address上位16ビットをr17にセット
105	SHLLI	r17,r17,16	
106	STW	r17,r16,UART_DATA_OFFSET	;r16を送信する
107			
108	_WAIT_SEND_	DONE:	
109	LDW	r17,r18,UART_STATUS_OFFSET	;STATUSを取得
110	ANDI	r18,r18,UART_TX_INTR_MASK	
111	BE	r0,r18,_WAIT_SEND_DONE	
112	ANDR	r0,r0,r0	
113			
114	LDW	r17,r18,UART_STATUS_OFFSET	
115	XORI	r18,r18,UART_TX_INTR_MASK	
116	STW	r17,r18,UART_STATUS_OFFSET	;Transmit Interrupt bitをクリア
117			
118	JMP	r31	;呼び出し元に戻る
119	ANDR	r0,r0,r0	;NOP

SEND_CHAR

104 行目から 105 行目で、r16 に UART 制御 レジスタのベースアドレスである 0x6000000 を格納しています。106 行目で UART Data レジスタに r16 の値を書き込みま す。108 行目から 111 行目で、UART Status レジスタの Transmit Interrupt ビットの値 を確認し、0 の場合はラベル_WAIT_SEND_DONE に戻ります。1 の場合は、114 行目か ら 116 行目で UART Status レジスタの Transmit Interrupt ビットを 0 クリアします。そ の後、118 行目で SEND_CHAR サブルーチンの呼び出し元に戻ります。

COLUMN

第3章

サブルーチン

サブルーチンとは、まとまった処理をモジュール化して、別のルーチンから呼び出せるように したものです。繰り返し行われる処理をサブルーチンにすることで、同じ処理を何度も記述する 手間が省け、プログラムの可読性も上がります。繰り返し行われる処理でなくても、意味的なま とまりを示すために処理をサブルーチン化することもあります。

AZ Processor では、サブルーチンコールに CALL 命令を使用します。CALL 命令ではサブ ルーチンへ命令の実行が移ると同時に CALL 命令の 2 つ先のアドレスを r31 に格納します。サ ブルーチンでの処理が終わり、元のプログラムに戻るときは、JMP r31 と記述します。この命 令によって r31 に格納されていたアドレスに命令の実行が移ります。CALL 命令には遅延スロ ットがあるので実行しても問題ないように注意が必要です。図 3-78 にサブルーチンコールの流 れを示します。



3.3

ASCII コード

ASCII コードとは、アメリカ規格協会の定めた文字コードです。7 ビットで表現され、128 種類のアルファベット、数字、記号、制御コードで構成されています。**リスト 3-2** のプログラ ム中で 'H' のようにシングルクォートで囲んだ文字は、アセンブラによって整数に変換されてい ます。文字の代わりに同じ意味を表す整数をプログラムに書いても、同じ文字が表示されます。 ASCII コード表を**表 3-10** に掲載します。

▼表3-10 AS	CII コード表
-----------	----------

10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字
0	0×0	NUL	32	0x20	SPACE	66	0x40	@	99	0×60	`
1	0×1	SOH	33	0x21	!	67	0x41	А	100	0x61	а
2	0x2	STX	34	0x22	"	68	0x42	В	101	0x62	b
3	0x3	ETX	35	0x23	#	69	0x43	С	102	0x63	с
4	0x4	EOT	36	0x24	\$	70	0x44	D	103	0x64	d
5	0x5	ENQ	37	0x25	%	71	0x45	E	104	0×65	е
6	0×6	ACK	38	0x26	&	72	0x46	F	105	0×66	f
7	0x7	BEL	39	0x27		73	0x47	G	106	0x67	g
8	0x8	BS	40	0x28	(74	0x48	Н	107	0×68	h
9	0x9	ΗT	42	0x29)	75	0x49	I	108	0×69	i
10	0xA	LF	43	0x2A	*	76	0x4A	J	109	0x6A	j
11	0xB	VT	44	0x2B	+	77	0x4B	К	110	0x6B	k
12	0×C	FF	45	0x2C	,	78	0x4C	L	111	0x6C	I
13	0xD	CR	46	0x2D	-	79	0x4D	М	112	0x6D	m
14	0×E	SO	47	0x2E		80	0x4E	N	113	0×6E	n
15	0×F	SI	48	0x2F	/	81	0x4F	0	114	0×6F	0
16	0x10	DLE	49	0x30	0	82	0x50	Р	115	0x70	p
17	0x11	DC1	50	0x31	1	83	0x51	Q	116	0x71	q
18	0x12	DC2	51	0x32	2	84	0x52	R	117	0x72	r
19	0x13	DC3	52	0x33	3	85	0x53	S	118	0x73	s
20	0x14	DC4	53	0x34	4	86	0x54	Т	119	0x74	t
21	0x15	NAK	54	0x35	5	87	0x55	U	120	0x75	u
22	0x16	SYN	55	0x36	6	88	0x56	V	121	0x76	V
23	0x17	ETB	56	0x37	7	89	0x57	W	122	0x77	w
24	0x18	CAN	57	0x38	8	90	0x58	Х	123	0x78	×
25	0x19	EM	58	0x39	9	91	0x59	Y	124	0x79	У
26	0x1A	SUB	59	0x3A	:	92	0x5A	Z	125	0x7A	z
27	0x1B	ESC	60	0x3B	;	93	0x5B]	126	0x7B	{
28	0x1C	FS	61	0x3C	<	94	0x5C		127	0x7C	1
29	0x1D	GS	62	0x3D	=	95	0x5D]	125	0x7D	}
30	0x1E	RS	63	0x3E	>	96	0x5E	^	126	0x7E	~
31	0×1F	US	65	0x3F	?	98	0x5F	-	127	0x7F	DEL

3.3.3 プログラムの実行

プログラムの動作確認を行います。まずは、3.2.6 項と同様にソースコードを作成し、 SVF ファイルの作成までを行います。次に、AZPR EvBoard の電源を入れて、USB ケー ブルでパソコンと接続します。パソコンにデバイスが認識されたら、Tera Term を起動 します。

Tera Term を起動すると、「新しい接続」ダイアログが開きます。ここでシリアルポートを選択します。USB シリアル変換デバイスの FT2232 によるシリアルポートは、COM ポートの値の異なる 2 つの「USB Serial Port」が存在します。数字の小さいほうの COM ポートは、FPGA のコンフィギュレーション用のポートなので、ここでは数字が大きい ほうの COM ポートを選択して、[OK] ボタンをクリックします。図 3-79 に「新しい接続」 ダイアログを示します。

Tera Term: 新し	い接続	X
© TCP/ĮP	ホスト(丁): myhost.example.com 「ビヒストリ(Q) サービス: O Telnet TCPボート#(P): 22 ③ SSH SSHバージョン(V): SSH2 ● その他 プロトコル(C): UNSPEC	•
● シリアル・	ボート(<u>R</u>): COM4: USB Serial Port (COM4)	•
	OK キャンセル ヘルプ(H)	

▲ 図 3-79 新しい接続

メニューバーより [設定] → [シリアルポート] を選択して、シリアルポートの設定 を行います。この設定値は AZPR SoC の UART の仕様により設定値が決まります。図 **3-80** にシリアルポート設定を示します。

Tera Term: シリアルポー	ト設定	x
ボート(<u>P</u>):	COM4 -	ОК
ボー・レート(B):	38400 -	
データ(<u>D</u>):	8 bit 🔹	キャンセル
バリティ(<u>A</u>):	none 🔻	
ストップ(<u>s</u>):	1 bit 🔹	ヘルプ(円)
フロー制御(<u>F</u>):	none 🔻	
送信遅延 0 ミリ利)/字(<u>C</u>) 0 ミ	リ秒/行(_)

▲ 図 3-80 シリアルポート設定

UrJTAG でコンフィギュレーションを行い、リセットボタンを押すと、図 3-81 のように、 Tera Term の画面に「Hello,world.」の文字が出力されます。

COM4:38400baud - Tera Term VT	
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)	
Hello,world.	

▲図 3-81 「Hello,world.」の出力

3.4 プログラムローダ

本節では、プログラムローダについて説明します。

ここまでのプログラムの実行手順では、プログラムの作成や修正をするたびに論理合成 や配置配線などを行なって、FPGA のコンフィギュレーションをしなければいけません でした。本節では、AZPR Processorの RAM 領域であるスクラッチパッドメモリにプロ グラムを転送して実行するプログラムローダについて説明します。プログラムローダを ROM に格納し、プログラムの作成や修正をしたときには、プログラムの転送のみを行う ことで、作業が短縮され、効率よく開発が行えるようになります。

本節で解説するプログラムローダは、パソコンからシリアル通信でデータを受け取り、 スクラッチパッドメモリに展開します。すべてのデータを書き込んだ後、スクラッチパッ ドメモリの先頭アドレスからプログラムを実行します。

データ通信を行うためには、通信をするための決まり事(プロトコル)が必要になります。 実装が比較的簡単であることと、Tera Term で利用可能であることから、ここではデー タ転送プロトコルに XMODEM を利用します。まず XMODEM の仕様について説明しま す。次に、AZ PRocessor で実行するローダとロード対象のプログラムについて説明します。 最後に、プログラムをロードするまでの手順を説明します。

3.4.1 XMODEM 解説

XMODEM にもいくつか種類がありますが、ここでは基本となる XMODEM-SUM について説明します。これ以降、XMODEM と記した場合は、XMODEM-SUM を指します。 XMODEM はデータブロック単位でデータを送信し、制御コードを使用して通信制御を 行います。通信制御で使われる制御コードを**表 3-11** に示します。

略号	意味	16 進数	送出元
SOH	Start Of Heading	0x01	送信側
EOT	End Of Transmission	0x04	送信側
CAN	CANcel	0x18	双方
ACK	ACKnowlege	0x06	受信側
NAK	Negative AcKnowlege	0x15	受信側

▼ 表 3-11 XMODEM で使う制御コード

SOH は送信データブロックの先頭に入ります。ACK は、受信側が正しくブロックを受

3

プログラミング

信したことを送信側に伝えます。NAK は送られてきたブロックに誤りがあったなどこと を送信側に伝え、送信側は NAK を受信すると直前に送ったブロックを再送します。EOT はファイル転送の終了を意味します。これを受けた側は、ACK を返してファイル転送を 終了します。CAN は何らかの理由でファイル転送を続行できなくなった場合に送信側、 あるいは受信側から送信します。これを受けた側は、ACK を返すことなく処理をエラー 終了します。図 **3-82** に送信データのブロックの構成を示します。



▲ 図 3-82 送信データのブロック構成

ブロックの先頭には SOH が入ります。BN はブロック番号が入り、0x01 からはじまり ます。BN が 0xFF に到達した場合、次は 0x00 になります。BNC はブロック番号をビッ ト反転した値になります。DATA には 128 バイトのデータが入ります。データが 128 バ イトに満たないときは、0x1A で埋めて 128 バイトにします。CS は 1 バイトのチェック サムで、1 ブロックのデータをすべて足した下位 8 ビットになります。

次に、図 3-83 を参照して、データの転送手順を説明します。



▲ 図 3-83 XMODEM のデータのやりとり

第3章 プログラミング

- (1) 受信側から NAK を送信します。
- (2) 送信側は、NAK を受信すると1つ目のブロックを送信します。
- (3) 受信側では、先頭の1バイト(ヘッダ)が SOH となっていることを確認し、エ ラーチェックを行います。エラーチェックは、BN と BNC がビット反転の関係にあ るか(1の補数となっているか)を確認した後、チェックサムを確認します。すべ て正しければ ACK、エラーがあれば NAK を送信します。
- (4)送信側は、ACKを受信すると次のブロックを送信します。NAKを受信すると、 直前に送信したブロックを再送します。送信側は、データをすべて送り終わると EOTを送信します。
- (5) 受信側は、EOT を受信すると、ACK を送信します。送信側ではこの ACK を受信してファイル転送を終了します。

3.4.2 プログラムローダの作成

プログラムローダは、XMODEM の受信側の処理を行ってデータを受信し、スクラッ チパッドメモリにデータを書き込みます。すべてのデータを書き込んだ後、スクラッチパ ッドメモリの先頭アドレスからプログラムを実行します。プログラムローダのフローチャ ートを図 3-84 に示します。

プログラムローダは UART のバッファクリアを行い、プッシュスイッチが押されるの を待ちます。プッシュスイッチが押されると、NAK を送信し、ヘッダを受信します。ヘ ッダが SOH のときは BN、BNC、DATA を受信します。DATA は1バイトずつ受信し、 4バイトのデータに組み立て、スクラッチパッドメモリに書きこんで行きます。これを 128 バイト分繰り返します。

DATA 受信完了後、CS を受信します。その後、2つのエラーチェックを行います。1 つ目のエラーチェックとして BN と BNE を足して 0xFF と比較します。0xFF でなけれ ばエラーなので、NAK を送信します。BN と BNE のチェックでエラーとなっていなけれ ば、2つ目のエラーチェックとしてチェックサムを確認します。このチェックサムによる、 データの誤り検出は、受信した前データをバイト単位で加算して、CS の値と比較します。 値が同じ場合、データは正常であり、異なる値となった場合は、エラーとなります。チェ ックサムでもエラーが発生した場合は、NAK を送信します。エラーが発生しなかった場 合は ACK を送信し、DATA 受信で書き込むスクラッチパッドメモリの領域を、次の領 域に変更します。

そして次のヘッダを受信します。SOH を受信した場合は、再度 BN、BNC、DATA を



受信します。EOT を受信した場合は、ACK を送信して命令の実行をスクラッチパッドメ モリに移します。

▲ 図 3-84 プログラムローダのフローチャート



■プログラムローダのリスト

プログラムローダを**リスト 3-5** に示します。

▼ リスト 3-5 プログラムローダ (loader.asm)

1	;;;; \$	シンボルの	D定義			
2	UART	BASE_AD	DR_H	EQU	0×6000	;UART Base Address High
3	UART	STATUS	OFFSET	EQU	0x0	;UART Status Register Offset
4	UART DATA OFFSET		EQU	0×4	;UART Data Register Offset	
5	UART	RX_INTF	_MASK	EQU	0×1	;UART Receive Interrupt
6	UART	TX_INTF	_MASK	EQU	0x2	;UART Receive Interrupt
7						
8	GPI0_	BASE_AD	DR_H	EQU	0×8000	;GPIO Base Address High
9	GPI0_	IN_OFFS	SET	EQU	0x0	;GPIO Input Port Register Offset
10	GPI0_	_OUT_OFF	SET	EQU	0×4	;GPIO Output Port Register Offset
11						
12	SPM_E	BASE_ADD	DR_H	EQU	0x2000	;SPM Base Address High
13						
14	XMODE	EM_SOH		EQU	0x1	;Start Of Heading
15	XMODE	EM_EOT		EQU	0x4	;End Of Transmission
16	XMODE	EM_ACK		EQU	0×6	;ACKnowlege
17	XMODE	EM_NAK		EQU	0×15	;Negative AcKnowlege
18	XMODE	M_DATA	SIZE	EQU	128	
19						
20						
21	>	KORR	r0,r0,r	9		
22						
23	(ORI	r0,r1,h	igh(CLEA	R_BUFFER)	;ラベルCLEAR_BUFFERの上位16ビットをr1にセット
24	9	SHLLI	r1,r1,1	6		
25	(ORI	r1,r1,l	ow(CLEAR	_BUFFER)	;ラベルCLEAR_BUFFERの下位16ビットをr1にセット
26						
27	(ORI	r0,r2,h	igh(SEND	_BYTE)	;ラベルSEND_BYTEの上位16ビットをr2にセット
28	5	SHLLI	r2,r2,1	6		
29	(ORI	r2,r2,l	ow(SEND_I	BYTE)	;ラベルSEND_BYTEの下位16ビットをr2にセット
30						
31	(ORI	r0,r3,h	igh(RECV	_BYTE)	;ラベルRECV_BYTEの上位16ビットをr3にセット
32	5	SHLLI	r3,r3,1	6		
33	(DRI	r3,r3,l	ow(RECV_I	BYTE)	;ラベルRECV_BYTEの下位16ビットをr3にセット
34						
35	(ORI	r0,r4,h	igh(WAIT	_PUSH_SW)	;ラベルWAIT_PUSH_SWの上位16ビットをr4にセット
36	5	SHLLI	r4,r4,1	6		
37	(ORI	r4,r4,l	ow(WAIT_	PUSH_SW)	;ラベルWAIT_PUSH_SWの下位16ビットをr4にセット
38						
39	;;; l	JARTのバ	ッファクリ	リア		
40	(CALL	r1			;CLEAR_BUFFER呼び出し
41	A	ANDR	r0,r0,r	9		;NOP
42						

プログラムローダ 3.4

43 44 45 46 47 48 49		ORI SHLLI ORI SHLLI ORI STW	r0,r20,GPI0_BASE_ADDR_H r20,r20,16 r0,r21,0x3 r21,r21,16 r21,r21,0xFFFF r20,r21,GPI0_OUT_OFFSET	;GPI0 Base Address上位16ビットをr20にセット ;16ビット左シフト ;出力データを上位16ビットをr21にセット ;16ビット左シフト ;出力データを下位16ビットをr21にセット ;GPI0 Output Portに出力データを書き込む
50	;;	Wait Push	n Switch	
51		CALL	r4	
52		ANDR	r0, r0, r0	
53 54		NAKi关信		
55	,,	ORI	r0.r16.XMODEM NAK	:r16にNAKをセット
56		CALL	r2	;SEND BYTE呼び出し
57		ANDR	r0,r0,r0	;NOP
58				
59		XORR	r5, r5, r5	
60 61	;;	ノロックの	元現を受信9 る	
62	,, RF(
63		CALL	r3	:RECV BYTE呼び出し
64		ANDR	r0, r0, r0	;NOP
65				
66	;;	受信データ		
67		ORI	r0, r6, XMODEM_SOH	;r6にS0Hをセット
68		BE	r16,r6,RECV_SOH	NOD
09 70		ANDK	r0,r0,r0	;NUP
71	::	EOT		
72	;;	ACK送信		
73		ORI	r0,r16,XMODEM_ACK	;r16にACKをセット
74		CALL	r2	;SEND_BYTE呼び出し
75		ANDR	r0,r0,r0	;NOP
76				
// 70	;;	jump to s	FOM FASE ADDD H	·SPM Base Address H位16ビットたr6にセット
79		SHLLT	r6.r6.16	, SFI Dase AddressIOC / Net Orce / N
80		0		
81		JMP	r6	;SPMのプログラムを実行する
82		ANDR	r0,r0,r0	;NOP
83				
84	;;	SOH		
00 96	RE(_V_SUΠ: BN函信		
87	, ,		r3	:RECV BYTE呼び出し
88		ANDR	r0, r0, r0	;NOP
89		ORR	r0,r16,r7	;r7に受信データBNをセット
90				

3 プログラミング

続く➡

91 ;; BNC受信 92 CALL r3 ;RECV BYTE呼び出し 93 ANDR r0,r0,r0 ;NOP 94 ORR r0,r16,r8 ;r8に受信データBNCをセット 95 96 ORI r0,r9,XMODEM DATA SIZE 97 XORR r10,r10,r10 ;r10をクリア 98 XORR r11, r11, r11 ;r11をクリア 99 100 ;; 1ブロック受信 101 ; byte0 102 READ BYTE0: 103 CALL r3 ;RECV_BYTE呼び出し 104 ANDR r0,r0,r0 ;NOP 105 ADDUR r11,r16,r11 106 SHLLI r16,r16,24 ;24bit左シフト 107 ORR r0,r16,r12 108 109 ; bytel 110 CALL r3 ;RECV BYTE呼び出し 111 ANDR r0,r0,r0 ;NOP 112 ADDUR r11,r16,r11 113 SHLLI r16,r16,16 ;16bit左シフト 114 ORR r12,r16,r12 115 116 ; byte2 117 CALL r3 ;RECV BYTE呼び出し 118 ORR r0,r0,r0 ;NOP 119 ADDUR r11,r16,r11 120 SHLLI r16,r16,8 ;8bit左シフト 0RR 121 r12, r16, r12 122 123 ; byte3 124 CALL r3 ;RECV BYTE呼び出し 125 r0,r0,r0 ORR ;NOP 126 ADDUR r11, r16, r11 127 0RR r12, r16, r12 128 129 ; write memory ;SPM Base Address上位16ビットをr13にセット 130 ORI r0,r13,SPM BASE ADDR H 131 SHLLI r13,r13,16 132 133 SHLLI r5, r14,7 134 ADDUR r14,r10,r14 135 ADDUR r14, r13, r13 136 STW r13,r12,0 137 138 ADDUI r10,r10,4

プログラムローダ 3.4

139 140 141	BNE ANDR	r10,r9,READ_BYTE0 r0,r0,r0	;NOP
142	;; CS受信		
143	CALL	r3	;RECV_BYTE呼び出し
144	ANDR	r0,r0,r0	;NOP
145 146	URR	r0,r16,r12	
147	;; Error Che	eck	
148	ADDUR	r7,r8,r7	
149	ORI	r0,r13,0xFF	;r13に0xFFをセット
150	BNE	r7,r13,SEND_NAK	;BN+BNCが0xFFでなければNAK送信
151	ANDR	r0,r0,r0	;NOP
152			
153	ANDI	r11,r11,0xFF	;r11に0xFFをセット
154	BNE	r12,r11,SEND_NAK	;check sumが正しいか
155	ANDR	r0,r0,r0	;NOP
156			
157	;; ACK送信		
158	SEND_ACK:		
159	ORI	r0,r16,XMODEM_ACK	;r16にACKをセット
160	CALL	r2	;SEND_BYTE呼び出し
161	ANDR	r0,r0,r0	;NOP
162	ADDUI		
163	BINE	r0,r0,RETURN_RECV_HEADER	NOD
164	ANDR	10,10,10	;NOP
166	•• NAK注行		
167			
168		r0 r16 XMODEM NAK	·r16にNAKをセット
169		r2	·SEND BYTE呼び出し
170	ANDR	r0, r0, r0	NOP
171	,		,
172	:: RECV HEAD	DERに戻る	
173	RETURN RECV	HEADER:	
174	BE	r0,r0,RECV HEADER	
175	ANDR	r0,r0,r0	;NOP
176			

■シンボルの定義

プログラム中でアクセスする UART と GPIO の制御レジスタのベースアドレスと各レ ジスタのオフセット、XMODEM で使う制御コード、スクラッチパッドメモリのベース アドレス、データサイズを定義しています。

■サブルーチンコールの設定

r1 にはラベル CLEAR_BUFFER の値を、r2 にはラベル SEND_BYTE の値を、r3 には ラベル RECV_BYTE の値を、r4 にはラベル WAIT_PUSH_SW の値を格納しています。

■UART バッファクリア

シリアル通信のプログラムと同じように、CLEAR_BUFFER を呼び出して、UART の バッファ内が空になるまで、データの読み出しを行っています。CLEAR_BUFFER サブ ルーチンについては 3.3.2 項で説明していますので、そちらを参照してください。

■LED 制御

43 行目から 48 行目で、LED 制御を行っています。3.2.6 項で説明した LED 制御とほぼ 同じなのですが、GPIO Output Port レジスタの LED に割り当てられているビットをす べて1にして、すべての LED を消灯しています。このプログラムは、主に UART での データのやりとりとなるため、転送が始まるまではプログラムが動作しているかがわかり ません。AZPR EvBoard の LED はデフォルトの設定ではすべての LED が点灯するため、 この LED 制御を行うことでプログラムが動作しているかどうかを AZPR EvBoard 上の LED で確認することができます。

■プッシュスイッチの入力待ち

WAIT_PUSH_SW を呼びだし、プッシュスイッチが押されるのを待ちます。WAIT_ PUSH_SW サブルーチンについては後述します。

■NAK 送信

r16 に送信データとして NAK を格納し、CALL 命令で SEND_BYTE サブルーチンを 呼んでいます。これによって、NAK を送信します。

■ヘッダ受信

RECV_BYTE サブルーチンを呼び出し、1 バイト受信しています。受信データが SOH となっているかを確認し、SOH の場合は、ラベル RECV_SOH に分岐し、BN 受信を行 います。SOH ではない場合は、受信データが EOT であるとして ACK 送信を行います。 RECV_BYTE サブルーチンについては後述します。

■ACK 送信

NAK 送信と同様に、r16 に送信データとして ACK を格納し、CALL 命令で SEND_ BYTE サブルーチンを呼び出しています。これによって、ACK を送信します。ここで XMODEM による転送が完了します。78 行目から 79 行目で、r6 にスクラッチパッドメモ リのベースアドレスである 0x20000000 を格納し、81 行目の JMP 命令でスクラッチパッ ドメモリに命令の実行が移ります。その後、スクラッチパッドメモリに書き込まれたプロ グラムを実行します。

■BN 受信

RECV_BYTE サブルーチンを呼び出し、1 バイト受信しています。受信データをr7 に セットしています。

■BNC 受信

RECV_BYTE サブルーチンを呼び出し、1 バイト受信しています。受信データを r8 に セットしています。

■DATA 受信とスクラッチパッドメモリへの書き込み

96 行目では r9 に 1 ブロックの DATA 領域のサイズである 128 を格納しています。r10 は 1 ブロック中に受信した DATA 領域のサイズを記録するカウンタとして使用し、r11 はチェックサムのために使用します。r10 と r11 は、それぞれ 98 行目、99 行目で 0 クリ アをしています。

DATA 領域は、byte0、byte1、byte2、byte3 に分けて RECV_BYTE サブルーチンを 呼び出し、1 バイトずつデータを受信しています。データを受信後、r12 に受信データを 格納しています。byte0 は 24 ビット分、byte1 は 16 ビット分、byte0 は 8 ビット分左シ フトしてから格納することで、r12 は図 3-85 で示すような 4 バイトのデータに組立てて います。



▲図3-85 4バイトのデータ

第3章 プログラミング

105 行目、112 行目、119 行目、126 行目では、受信データを加算して rl1 に格納し、チェックサムを計算しています。

130 行目から 135 行目で、4バイトのデータを書き込むアドレスを計算しています。 130 行目から 131 行目では、r13 にスクラッチパッドメモリのベースアドレスである 0x20000000 を格納しています。133 行目では、128 倍と同じ結果が得られる7 ビットの左 シフトを受信したブロック数を格納する r5 に対して行い、r14 に格納しています。134 行 目から 135 行目で、r10 と r13 と r14 を加算し、r13 に格納しています。つまり、以下の 結果を r13 に格納していることになります。

r13 + r5 × 128 + r10 r5 :受信したブロック数 r10:1 ブロック中に受信した DATA 領域のサイズ r13:スクラッチバッドメモリのベースアドレス

136 行目で、上記で計算した r13 で示すアドレスに r16 の値を書き込みます。138 行目 から 129 行目で、カウンタに 4 を加算し、受信データが 128 バイトに到達したかどうかを 確認しています。到達していない場合は、ラベル READ_BYTE0 に分岐し、DATA 受信 を行います。到達した場合は、CS 受信を行います。

■CS 受信

RECV_BYTE サブルーチンを呼び出し、1 バイト受信しています。受信データを r12 にセットしています。

エラーチェック

148 行目から 149 行目で、BN の値が格納されている r7 と BNE の値が格納されている r8 を足して、0xFF となっているか確認しています。0xFF となっていない場合は、ラベル SEND_NAK に分岐し、NAK 送信を行います。0xFF となっている場合は、153 行目 から 154 行目でチェックサムを格納している r11 の値が 0xFF となっているかを確認しま す。0xFF となっていない場合は、ラベル SEND_NAK に分岐し、NAK 送信を行います。0xFF となっているばあいは ACK 送信を行います。

■ACK 送信

r16 に送信データとして ACK を格納し、CALL 命令で SEND_BYTE サブルーチンを

呼び出し、ACK を送信します。162 行目で、受信したブロック数を格納している r5 に1 を加算します。その後、163 行目でラベル RETURN_RECV_HEADER に分岐します。

■NAK 送信

r16 に送信データとして NACK を格納し、CALL 命令で SEND_BYTE サブルーチンを 呼び出し、NAK を送信します。

■ヘッダ受信に戻る

ラベル RECV_HEADER に戻り、再度ヘッダ受信を行います。

■RECV_BYTE サブルーチン

RECV_BYTE サブルーチンを**リスト 3-6** に示します。このサブルーチンは、UART の 受信データを r16 に格納します。

▼ リスト 3-6 RECV_BYTE サブルーチン (loader.asm)

220	RECV BYTE:		
221	ORI	r0,r17,UART_BASE_ADDR_H	;UART Base Address上位16ビットを r17にセット
222	SHLLI	r17,r17,16	
223			
224	LDW	r17,r18,UART_STATUS_OFFSET	; STATUSを取得
225	ANDI	r18,r18,UART_RX_INTR_MASK	
226	BE	r0,r18,RECV_BYTE	;Receive Interrupt bitが立ってい ればRECV_BYTEを実行
227	ANDR	r0,r0,r0	;NOP
228			
229	LDW	r17,r16,UART_DATA_OFFSET	;受信データを読む
230			
231	LDW	r17,r18,UART_STATUS_OFFSET	;STATUSを取得
232	XORI	r18,r18,UART_RX_INTR_MASK	
233	STW	r17,r18,UART_STATUS_OFFSET	;Receive Interrupt bitをクリア
234			
235	JMP	r31	;呼び出し元に戻る
236	ANDR	r0,r0,r0	;NOP

RECV_BYTE

224 行目で、UART Status レジスタを r18 に格納し、225 行目から 226 行目で Receive Interrupt ビットが0かどうかを確認しています。0 となっている場合は、ラベル RECV_ BYTE に戻り、再度 UART Status レジスタを読み出します。1 となった場合は、UART

第3章 プログラミング

Data レジスタで受信データを rl6 に格納します。231 行目から 233 行目で、Receive Interrupt レジスタをクリアしてサブルーチンを終了しています。

■WAIT_PUSH_SW サブルーチン

WAIT_PUSH_SW サブルーチンをリスト 3-7 に示します。このサブルーチンでは、 SW1 から SW4 のいずれかのプッシュスイッチが押されたときに、呼び出し元に戻ります。

```
▼ リスト 3-7 WAIT_PUSH_SW サブルーチン (loader.asm)
```

238	WAIT_PUSH_S	N :	
239	ORI	r0,r16,GPI0_BASE_ADDR_H	
240	SHLLI	r16,r16,16	
241	_WAIT_PUSH_S	SW_ON:	
242	LDW	r16,r17,GPI0_IN_OFFSET	
243	BE	r0,r17,_WAIT_PUSH_SW_ON	
244	ANDR	r0,r0,r0	;NOP
245	_WAIT_PUSH_S	SW_OFF:	
246	LDW	r16,r17,GPI0_IN_OFFSET	
247	BNE	r0,r17,_WAIT_PUSH_SW_OFF	
248	ANDR	r0,r0,r0	;NOP
249	_WAIT_PUSH_S	SW_RETURN:	
250	JMP	r31	
251	ANDR	r0,r0,r0	;NOP

WAIT_PUSH_SW

241 行目から 243 行目で、GPIO Input Port レジスタに値がセットされていないか確 認し、プッシュスイッチが ON になるまでラベル _WAIT_PUSH_SW_ON に戻り続けま す。次に、プッシュスイッチが OFF に戻ることを確認します。245 行目から 247 行目で、 再度 GPIO Input Port レジスタを確認し、プッシュスイッチが OFF になるまでラベル _WAIT_PUSH_SW_OFF に戻り続けます。

プログラムローダ 3.4

3.4.3 ロード対象のプログラムの作成

次はロード対象のプログラムについて説明します。ロード対象のプログラムの簡単な例 をリスト 3-8 のプログラムで示します。

▼ リスト 3-8 ロード対象のプログラムの例 (prog.asm)

1 2 3	;;; ロケーションアドレスの設定 2 LOCATE 0x20000000						
4	;;; シンボルの定義						
5	GPIO BASE ADDR H EQU 0x8000 ;GPIO Base Address High						
6 7	GPI	0_0UT_0FF	SET	EQU 0x4	;GPIO Output Port Register Offset		
8	;;;	LED点灯					
9		XORR	r0,r0,r	9			
10		ORI	r0,r1,G	PIO_BASE_ADDR_H	;GPIO Base Address上位16ビットをr1にセット		
11		SHLLI	r1,r1,1	б	;16 ビット左シフト		
12		ORI	r0,r2,0	x2	;出力データを上位16ビットをr2にセット		
13		SHLLI	r2,r2,1	6	;16 ビット左シフト		
14		ORI	r2,r2,0	xFFFF	;出力データを下位16ビットをr2にセット		
15		STW	r1,r2,G	PI0_OUT_OFFSET	;GPIO Output Portに出力データを書き込む		
16							
17	7 ;;; 無限ループ						
18	LOOF	P:					
19		BE	r0,r0,L0	00P	;L00Pに戻る		
20		ANDR	r0,r0,r	0	;NOP		

このプログラムは、**リスト 3-1** の LED を点灯させるためのプログラムとほぼ同じで、 動作としては LED の点灯を行います。**リスト 3-1** のプログラムと異なる点は、2 行目の LOCATE の記述です。

LOCATE でアドレスを指定することにより、プログラムの先頭番地を指定します。ア ドレス 0x20000000 番地は AZ Processor のスクラッチパッドメモリのベースアドレスと なっているため、**リスト 3-8** はスクラッチパッドメモリに配置するためのプログラムとな ります。

3.4.4 プログラムの実行

プログラムローダの動作確認を行います。まず、プログラムローダのソースコード 「loader.asm」を作成し、アセンブラでバイナリファイルに変換します。その後、このバ イナリファイルに対して 3.2.6 項と同様に BIT ファイルを作成します。

プログラムローダはこの後の節でも使うことになるので、AZPR EvBoard の電源を入 れた時に自動的に FPGA にコンフィギュレーションが行われるように、コンフィギュレ ーション ROM を使用することをおすすめします。コンフィギュレーション ROM に書き 込むために必要な MCS ファイルを作成する手順は、3.2.3 項の「MCS ファイル作成」を 参照してください。その後、SVF ファイルを作成します。MCS ファイルから SVF ファ イルを作成する手順は、3.2.3 項の「SVF ファイル作成」を参照してください。

UrJTAGでSVFファイルを再生し、コンフィギュレーション ROM に対して書き込み を行います。AZPR EvBoardの電源を入れて、USB ケーブルでパソコンを接続します。 パソコンにデバイスが認識されたら、UrJTAG を起動し、以下のコマンドを実行します。 上記手順で作成した SVF ファイルを「loader.svf」としています。

jtag>	cable jtagkey	
jtag>	detect	
jtag>	part 1	
jtag>	svf loader.svf progress	

次に、ロード対象のプログラムのソースコード「prog.asm」を作成し、アセンブラで バイナリファイル「prog.bin」に変換します。その後、Tera Term を起動します。3.3.3 項と同様に「新しい接続」ダイアログで、シリアルポートを選択します。ここで選択する シリアルポートも、COM ポートの異なる2つの「USB Serial Port」のうち数字の大きい ほうになります。

Tera Term のウィンドウが表示されたら、メニューバーより [ファイル] → [転送] → [XMODEM] → [送信] を選択します。メニューバーから選択している画面を図 3-86 に示します。

1	COM4:38400baud - Tera Ter	m VT			
ファ	イル(F) 編集(E) 設定(S)	コントロール(0)	ウィンドウ(W)	へレプ(H)	
	新しい接続(N)	Alt+N			*
	セッションの複製(U)	Alt+D			
	Cygwin接続(G)	Alt+G			
	ログ(L) ログにコメントを付加(0) ログを表示(V) ログダイアログを表示(W)				
	ファイル送信(S)		14 million th		
		•	XMODEM	▶ <u>– – – – – – – – – – – – – – – – – – –</u>	
			XMODEM	又信(N)…	
	リイレクトリを変更(C) ログを再生(R)		ZMODEM	· <u>Ka(3)</u>	
	印刷(P)	Alt+P	B-Plus Ouick-VAN	- F	
	接続断(D)	Alt+I			
	終了(X)	Alt+Q			
					-

▲ 図 3-86 XMODEM による送信の選択

図 **3-87** で示すようにファイル選択ダイアログが表示されるので、ロード対象のプログ ラムである「prog.bin」を選択します。このとき、オプションでは「チェックサム」が選 択されていることを確認してください。

👑 Tera Term: XM0	DDEM送信		x
ファイルの場所(I):	🕕 program 👻	GØB	°▼
g prog.bin			
ファイル名(<u>N</u>):			開(0)
ファイルの種類(工):	(ฮฺ๙ฺๅ๔๗ファイル(**)	•	キャンセル ヘルプ(H)
オプション	© <u>C</u> RC		

▲ 図 3-87 送信のファイル選択ダイアログ

ロード対象のプログラムを選択すると、Tera Term は転送待ちの状態となり、図 3-88 で示すウィンドウが表示されます。



▲ 図 3-88 転送待ちの状態で表示されるウィンドウ

ここまでで、準備は完了です。AZPR EvBoard のリセットボタンを押し、プログラム ローダを再実行します。AZPR EvBoard 上の SW1 から SW4 までのいずれかのプッシュ スイッチを押すと、Tera Term で指定したファイルの転送が開始します。転送中は転送 済容量がプログレスバーで表示され、転送が完了するとウィンドウは自動的に閉じます。 ロード対象のプログラムが動作すると、LED1 が点灯します。

ロード対象のプログラムを変更した場合は、アセンブラでバイナリファイルに変換し、 Tera Term の操作からやり直します。このように、プログラムローダを使うと、プログ ラムを転送するだけで動作確認をすることができます。

次の節からは、プログラムローダを使う前提でプログラムを作成します。

3.8 おわりに

本章では、AZ Processor のプログラミングについて解説しました。

始めに、AZ Processorの開発環境として、ISE WebPACK、UrJTAG、アセンブラの インストールと使い方を説明しました。その後、LED、シリアル通信、XMODEM、割り 込み、例外、7 セグメント LED を動作させるためのサンプルプログラムの説明を行いま した。最後に、AZPR EvBoard に搭載されている様々な周辺回路を動作させる応用プロ グラムとして、キッチンタイマーを作成しました。

謝辞

本書を出版するに際して、大変多くの方々からご助力を賜りました。末筆ではありますが、 そうした全ての方々に対して、感謝の意を述べたいと思います。

「CPU 自作入門」の書籍化にあたり、担当編集者の林也寸夫様に、深く感謝いたします。 彼のマネジメント、アドバイス、そして忍耐なしには、本書が完成する日は来なかったでしょ う。そして、編集長の加藤博様にも、厚く御礼申し上げます。

第1章の執筆にあたり、ツールの掲載を快諾して下さった IcarusVerilog 作者の Stephen Williams 様、GTKWave 作者の Tony Bybell 様に深く感謝いたします。

第2章の執筆にあたり、基板製作のアドバイスや機材の貸し出しをして下さったサンハヤ ト株式会社様、Eagleの使用方法についてアドバイスをして下さった有限会社サーキットボー ドサービス様、基板製造についてアドバイスをして下さった株式会社ピーバンドットコム様、 3D 部品ライブラリを提供して下さった株式会社図研 ePartFinder 運営事務局様に深く感謝 いたします。

第3章の執筆にあたり、ツールの不明点についての問い合わせに快く応じて下さったザイ リンクステクニカルサポートセンター様、ツールの掲載を快諾して下さった cblsvr-0.1_ft2232 作者の fenrir 様に深く感謝いたします。

本書の内容について、下記の方々から多くのご助言、ご指導を賜りました。深く感謝いた します。

株式会社 東芝 セミコンダクター&ストレージ社武田 瑛 様株式会社 東芝 セミコンダクター&ストレージ社真垣 郁男 様パナソニック株式会社西川 由理 様某情報サービス事業会社藤井 啓 様

また、本書の執筆にあたって、家族の理解と協力に感謝いたします。

最後に、本書を手に取って下さった全ての方々に、心より感謝いたします。

CPU 自作入門 おわりに

本書ではオリジナルのコンピュータシステムを作成しました。第1章では CPU である AZ Processor を中心に、I/O、メモリ、バスから構成される AZPR SoC の設計と実装について 説明しました。第2章では AZPR SoC を実機にて動作させるための基板を設計し、製作する 手順を説明しました。第3章では AZPR SoC 上で動作するプログラムを作成し、実機にて動 作を確認する手順を説明しました。

本書の原点は、執筆者らの同人活動です。執筆者の3人は同人サークル「れすぽん」(http:// respon.org/)において、CPUを独自に設計・実装し、その過程を同人誌にまとめて頒布して います。本書の前身は、れすぽんにて執筆した同人誌「CPU 自作入門」です。2007 年 8 月 の発足以来、同人誌をコミックマーケットなどで頒布していたところ、技術評論社の林様か ら書籍化の話を頂き、このたび出版に至りました。

書籍化にあたり、CPU、基板、ソフトウェアを新たに設計しました。より多くの人に読ん でもらうため、前提知識の敷居を低く設定し、本文も新たに書き下ろしました。限られた紙 面の中で、CPUの設計と実装、基板の設計と製作、プログラミングに至るまでを網羅的に説 明すると、どうしても内容が一足飛びになってしまいますが、本書では「ものをつくる」とい うことに重点を置いて執筆しました。コンピュータの設計と製作について説明した数少ない 書籍として、お役に立てれば幸いです。

2012 年 9 月 執筆者一同

■著者略歴

■ 第 1 章著者

水頭 一壽 (すいとう かずとし)

1986年3月福岡県生まれ。慶應義塾大学理工学研究科前期博士課程を修了。現在同大学の後期博士課程に在学。 組込みリアルタイムシステム用システムLSIの研究開発に従事。趣味は音楽、写真、自転車など。サークルれす ぼんでは論理設計を担当。

第2章著者

米澤 遼(よねざわ りょう)

1984年10月東京都生まれ。慶應義塾大学理工学研究科前期博士課程を修了後、株式会社東芝セミコンダクター &ストレージ社に勤務。高速シリアルインタフェースIPの開発に従事。趣味は電子工作、自宅サーバ管理など。サークルれすぼんでは基板設計と表紙絵を担当。

第3章著者

藤田 裕士(ふじた ゆうじ)

1985年3月愛知県生まれ。慶應義塾大学理工学研究科前期博士課程を修了後、日本電気株式会社に勤務。ファームウェア開発に従事。趣味は音楽鑑賞、ギター演奏など。サークルれすぽんではソフトウェア設計を担当。

んしちている。 ろんりせっけい きばんせいさく ろんりせっけい きばんせいさく マート ろんりせっけい きばんせいさく マート 日日による論理設計・基板製作・プログラミング~

2012年11月25日 初版 第1刷発行

	すいとう	かずとし	よねざわ	りょう	ふじた	ゆうじ
著者	水頭	一壽・	米澤	遼・	藤田	裕士

- 発行者 片岡 巌
- 発行所 株式会社技術評論社

東京都新宿区市谷左内町 21-13

電話 03-3513-6150 販売促進部 03-3513-6166 書籍第2編集部

印刷/製本 株式会社 加藤文明社

定価はカバーに印刷してあります

本書の一部または全部を著作権法の定める範囲を超え、無断で複写、複製、 転載、テーブ化、ファイルに落とすことを禁じます。

造本には細心の注意を払っておりますが、万一、乱丁(ページの乱れ)や落 丁(ページの抜け)がございましたら、小社販売促進部までお送りください。 送料小社負担にてお取り替えいたします。

© 2012 水頭 一壽・米澤 遼・藤田 裕士

ISBN978-4-7741-5338-4 C3055 Printed in Japan ■カバーデザイン

有限会社釣巻デザイン室
 ■本文デザイン・DTP
 有限会社スタジオ・キャロット
 ■編集担当
 林 也寸夫

■お問い合わせについて

本書に関するご質問は記載内容につい てのみとさせていただきます。本書の 内容以外のご質問には一切応じられま せんので、あらかじめご了承ください。 なお、お電話でのご質問は受け付けて おりませんので、書面またはFAX、小 社 Web サイトのお問い合わせフォー ムをご利用ください。

〒 162-0846

東京都新宿区市谷左内町 21-13 株式会社技術評論社 書籍第2編集部 「CPU 自作入門 ~ HDL による論理設 計・基板製作・プログラミング~」係 FAX:03-3513-6167 URL: http://gihyo.jp/ (技術評論社 Web サイト)



本コンテンツは, Gihyo Digital Publishing において, 電子化データとして再発行されたものです。

初回配信日:2012 年 10 月 20 日 最新配信日:2012 年 10 月 20 日 電子版 ISBN978-4-7741-5381-0

本コンテンツは、電子化データの一部分をお試し版とし て再発行したものです。

正誤表などは以下をご覧ください。 http://gihyo.jp/book/2012/978-4-7741-5338-4 電子版の詳細は以下をご覧ください。 https://gihyo.jp/dp/ebook/2012/978-4-7741-5381-0

初回配信日:2017年10月27日 最新配信日:2017年10月27日