

SpaceWire/SpaceCube

Tutorial

Takayuki Yuasa and SpaceWire User Group Japan
Version : 2010/06/18

目次

はじめに / Introduction	4
全体像 / Overview	4
言葉の説明	5
より詳しいドキュメント	10
重要なお知らせ	10
更新履歴	10
連絡先	10
SpaceCubeのセットアップ / Setup a SpaceCube	11
全体像 / Overview	11
EclipseをもちいたSpaceWire/RMAP Libraryとおまけソフトのビルト / Building SpaceWire/RMAP Library and appendix softwares using Eclipse.	19
SpaceWire I/Fボードを接続 / Connecting SpaceWire I/F board	22
全体像 / Overview	22
SpaceWire I/Fボードの構造 / Structure of the SpaceWire I/F board	22
物品の準備 / Preparation of materials	23
SpaceWire FPGAに焼き込まれているSpaceWire IPコアの確認 / Check SpaceWire IP core installed in SpaceWire FPGA	24
コマンドライン版rmaphongoを用いてボード上LEDを点灯させる / Light-on LEDs on the board using command-line rmaphongo.	24
コマンドライン版rmaphongoを用いてボード上のSDRAMにアクセス / Accessing SDRAM on a board using command-line rmaphongo	27
SpaceWire I/Fを利用するプログラムの開発 / Development of programs using SpaceWire I/F	28
全体像 / Overview	28

UserFPGAの作成	34
全体像	34
開発環境の整備	34
UserFPGA_templateについて	35
もっと詳しいVHDL	38
UserFPGA_templateの書き込み	40
UserFPGA_templateの内容説明と動作確認	40
UserFPGA_templateから先へ進むには	43
SDRAMの利用について	43
UserFPGA_templateをもちいた例題(VHDLの勉強用)	44
SpaceWire/RMAP Libraryの応用 / Further usage of SpaceWire/RMAP Library	45
このセクションについて / About this section	45
SpaceWire to TCP/IP Bridge On SpaceCube1	45
RMAP Target Simulator on SpaceCube1	46
参考文献 / References	48

はじめに / Introduction

全体像 / Overview

このチュートリアルでは、SpaceWireやSpaceCubeの概念的なところはどこに置いておいて、「どういう手順で作業を行えばSpaceWireやSpaceCubeを用いたデータ通信ができるか」や、「それをどのように実際の実験で使うのか」をまとめます。このチュートリアルを読むと、以下のことができるようになります。

- SpaceCubeを起動して、基本的なコマンドを実行する
- C++言語で記述したプログラムを、PCやMac上に構築する「クロスコンパイル環境」を用いて、SpaceCube向けにコンパイルする
- 作成したSpaceCube用プログラムをSpaceCubeに転送して、実行する
- SpaceCubeとSpaceWire I/Fボードを接続し、サンプルプログラムを用いてRMAPによる通信を行う
- RMAPを行うプログラムを自分で開発してみる
- SpaceWire I/Fボード上のUserFPGA用ロジックを自分で開発してみる

このチュートリアルを読み終わって、もっと詳しい内容が知りたくなった場合は、SpaceWire/RMAPの仕様書[1,2]、「SpaceWire/RMAP Library」のドキュメント[15]や、各種SpaceWire I/Fボードの回路図などを参照してください。

This tutorial summarizes "How and with what kind of work we can make data transfer by SpaceWire and SpaceCube" or "How to use the program in actual measurements". This tutorial does not explain the concept of SpaceWire and SpaceCube. By learning the tutorial, the following things will be made available:

- To start up a SpaceCube and run basic commands
- To compile programs written by C++ for SpaceCube using "cross-compiling environment" on a PC or a Mac.
- To transfer the programs for SpaceCube to a SpaceCube and execute them.
- To connect a SpaceCube and a SpaceWire I/F board and make RMAP communication by sample programs
- To develop programs using RMAP by yourself.
- To develop UserFPGA logics on SpaceWire I/F board by yourself.

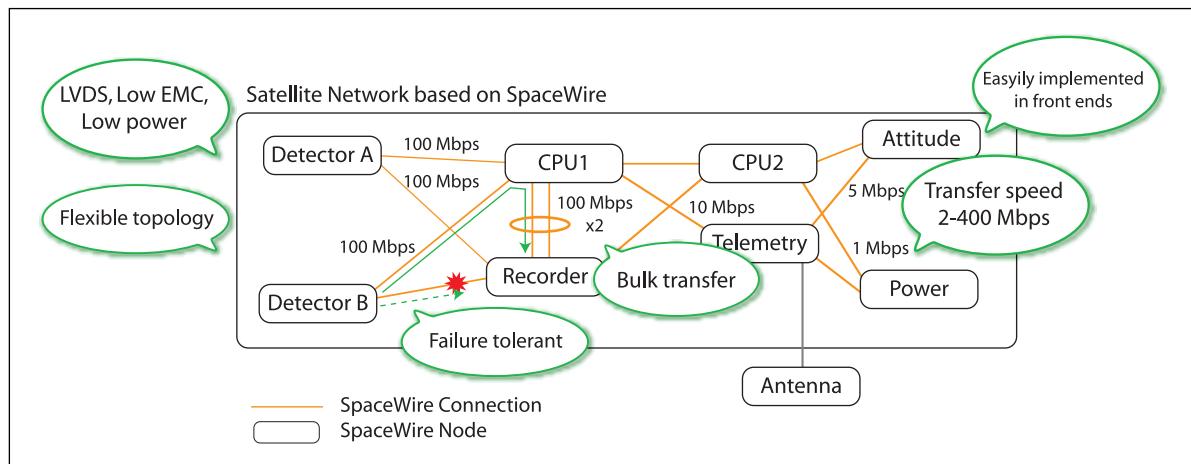
If you want to learn more after finishing learning this tutorial, see also SpaceWire/RMAP specification [1,2], "SpaceWire/RMAP Library" document [15] and various SpaceWire I/F board circuit diagrams.

言葉の説明

とりあえずSpaceCubeとSpaceWireを使ってみるにあたって、知ておくと便利な言葉をまとめておきます。より詳しい説明は、後述の他のドキュメントを参照してください。

SpaceWire

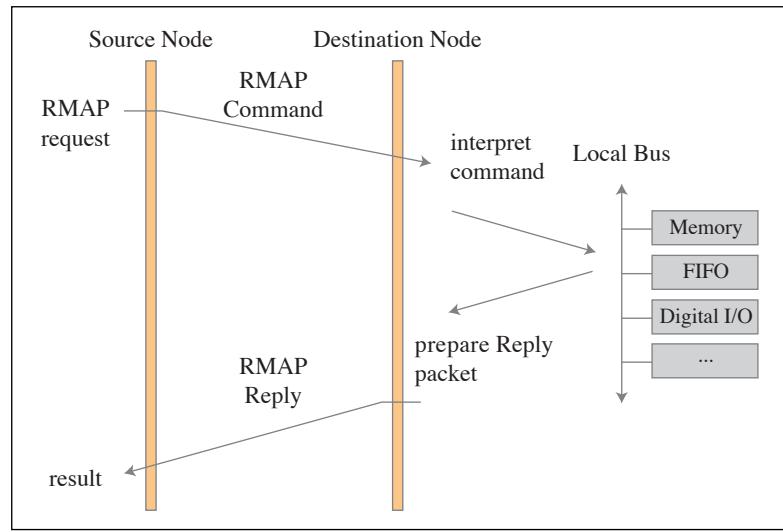
宇宙機用データ通信インターフェースの仕様。コネクタ形状、信号の電気特性、ケーブル、パケット構造、ルーティング方法など、ネットワークの複数のレイヤーに渡って仕様を定めています。簡単なプロトコル、幅広い通信速度(2~400Mbps)、冗長系が組みやすい、などが主な特長です。たとえば、パーソナルコンピュータの世界では、USBやEthernetなどのインターフェースは、コンピュータの製造メーカーが異なっても同じコネクタでいろんな周辺機器が利用できます。衛星搭載の装置も、共通のインターフェースを定義して、みんなで使うことで、信頼性向上や技術の蓄積、コスト低減に役立てようという流れから、ESA、JAXA、NASA、ROSCOSMOS(ロシア宇宙機関)などが参加して、標準化作業が進められてきました。



RMAP

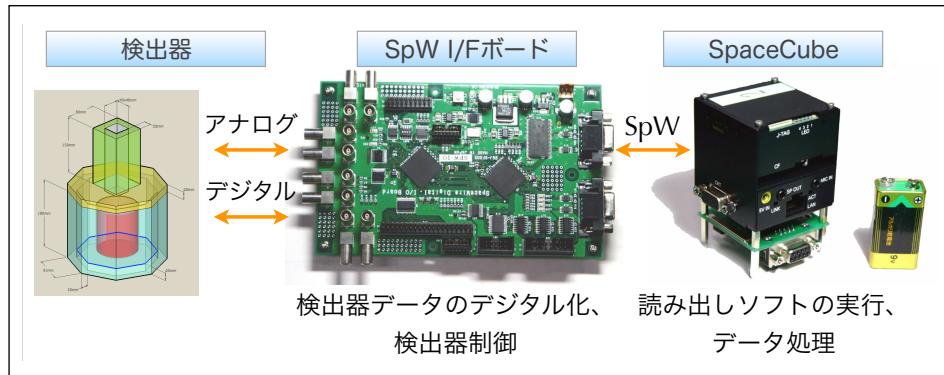
Remote Memory Access Protocolの略。SpaceWireネットワーク上で、データ通信を行うためのプロトコル。SpaceWireをEthernetだと思うと、RMAPはTCP/IPなどのレイヤーに相当。データ通信の親(Source)と、子(Destination)の間で、コマンドパケット Command Packetやリプライパケット Reply Packetをやり取りして、データの転送を行います。

SpaceWire I/Fボードに接続された検出器のデータをSpaceCubeに転送するときも、RMAPを使いますが、コマンドパケットの送信や、リプライパケットの受信、パケットの生成・解釈は、後述のSpaceWire/RMAP Libraryが自動的に行ってくれるので、ユーザアプリケーションのレイヤではとくに難しいことを考える必要はありません。



SpaceWire/SpaceCube-based Data Acquisition System (もしくは、SpaceWire/SpaceCubeを用いたデータ取得システム)

SpaceWireを検出器データの読み出しにも利用して、地上での検出器開発段階から、衛星に搭載してフライ特品のSpaceCubeでデータを読み出すまで、シームレスに移行できるよになればよいのではないか、という考えのもとに開発・利用されているデータ取得系(DAQ)です。構成は下図のように、検出器、検出器データをデジタル化しSpaceWireネットワークに流す「SpaceWire I/Fボード」、データ取得やデータを用いた計算を行なうコンピュータ「SpaceCube」の3つの部分からなっています。このドキュメントの目的のひとつは、そのような読み出し系を各サイトで立ち上げて、自分たちの実験をSpaceWireベースで行えるようにすることです。



SpaceCubeアーキテクチャ

SpaceWire I/Fを搭載し、リアルタイムOSによって動作するコンピュータの総称。衛星搭載のあつきには、SpaceWire経由で検出器のデータを読み出したり、検出器を制御したり、読み出したデータを処理したりするマシンとしてはたらきます。地上試験では主にSpaceCube1(シマフジ電機/JAXA)のことを指します。このドキュメントでも「SpaceCube」は「SpaceCube1」と読み替えてください。衛星搭載用には、SpaceCube2(NTSpace)、SpaceCard(MHI)などが開発されています。

T-Kernel

コンピュータのOSの一つ。その中でも、処理時間の管理や資源(メモリ、I/Fなど)の管理を細かく調整できる、リアルタイムOSという種類のOSです。東京大学大学院 情報学環教授の坂村先生が提唱しているTRONの流れをくんでいます。

SpaceCube1およびSpaceCube2はT-Kernelで動作しています。TRONはOSの名前ではなくて、アイデアの名前なので、その「実装(後述)」には様々なものがあり、TOPPERS、超漢字などがその例です。

T-Kernelおよびそれと関係が深いμTRON仕様は、組み込みデバイス向けのOS仕様なので、普通のPCとは考え方がことなる部分が多くあります。また、関数コールの名前の付け方がUNIXでよく使っているPOSIX標準やWindowsのWIN32APIなどと全く異なるので、たとえばUNIX/Linux用のプログラムはそのままでは全く動作しません(コンパイルできません)。

後述のSpaceWire/RMAP Libraryでは、ハードウェアやOSが変わってもユーザアプリケーションのレイヤでの変更はほとんどなくて済むように、OS間の違いを吸収するためのラッパ wrapper レイヤーを用意しています。

SpaceCube 1 (シマフジ電機製)

SpaceWireの開発を行うために、JAXAとシマフジ電機が協力して開発した小型コンピュータ。T-Kernelが動作するプラットフォーム仕様であるT-Engineとして開発されていたteacubeという、cubicなコンピュータをもとに、SpaceWire I/Fを追加する形で開発されました。あまりにも本体が小型化されているために、SpaceWire I/F用のD-Subコネクタが内蔵できないので、現状では外付け基板でコネクタを引き出しています。

SpaceCube by Shimafuji & JAXA

CPU : VR5701(64bit MIPS) 200MHz

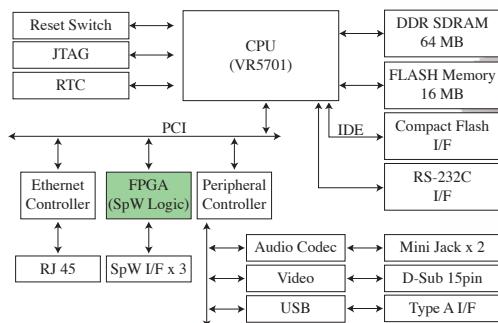
RAM : 64MB

OS : T-Kernel(TRON) or Linux

I/F : SpaceWire x 3, Fast Ethernet,
Serial, USB, VGA, CF

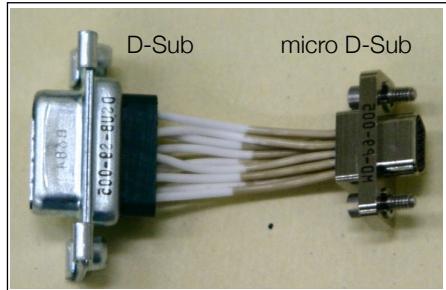
Power : DC +5V

Size : 52x52x55 mm³



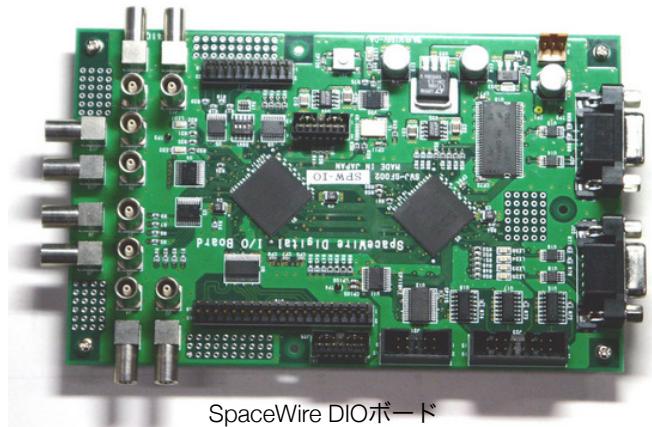
SpaceWire I/F

SpaceWireの通信インターフェースのこと。たとえばSpaceCube1には3つのSpaceWire I/Fがあります。衛星搭載のホンモノでは、SpaceWireの仕様に則ってmicro D-Subというコネクタが利用されますが、地上試験の段階では、ハンドリングが簡単でより安価なD-Subコネクタが多く利用されています(micro D-Subは抜き挿し回数に制限があったり、ねじ回しのときにトルク管理しないといけないなど、気軽に使うにくい)。



SpaceWire I/Fボード

SpaceWire I/Fを搭載しているアナログ・デジタルボードのことです。とくに、SpaceWireユーザ会の枠組みのなかで開発されている各種SpaceWire I/Fボードは、SpaceWire I/F以外に、UserFPGAやSpaceWire FPGAなどいくつかの共通のコンポーネントを搭載しています。地上試験では、シマフジ電機製のSpaceWire DIOボードやSpaceWire ADC/DACボード、ClearPulse製SpaceWire ADC Box、また、MHI製のSpaceWire Universal I/Oボードなどがよく使われています。



SpaceWire DIOボード

SpaceWire/RMAP Library (RMAPHongo Library)

SpaceCube上で、SpaceWireやRMAPを使ったデータ通信を行うプログラムを簡単に開発するためのソフトウェアライブラリです。C++言語で記述されており、オブジェクト指向の徹底、Eclipseを用いた効率的な開発、ドキュメンテーションの整備を大事な柱と位置づけて開発が行われています。このライブラリの元になった、ひとつ前のバージョンのライブラリ名を引き継いで、RMAPHongoライブラリと呼ばれることがあります。

FPGA

Field Programmable Gate Arrayの略。フリップフロップをたくさん(10^5 ~ 10^7 個)並べたような構造をもち、それらの間の配線を切り替えることで様々な論理回路を実現することができる電子部品です。インストラクションを一つ一つ実行していくCPUとは違い、複数の信号を同時に判定したり、複数のサブロジックを同時に動作させたりできます(もちろん、FPGA内部に、ロジックを組んでCPUを構築することもできる。こういうのをソフトコアCPUと呼ぶ)。

SpaceWire I/Fボードには、ユーザが利用できるFPGA「UserFPGA」が搭載されており、ここに実験固有の処理(トリガ判定やデジタルフィルタリング、イベントバッファリングなど)を記述して利用します。

VHDL/Verilog-HDL

二つとも、ロジックを記述するためのコンピュータ言語で、FPGAやASIC用のロジックを記述するときに利用します。どちらを用いても同等のロジックが記述できますが、発祥が異なるので、記述のしやすさ、可読性などが少し違います。とくにVerilog-HDLは、ロジックのシミュレーションに強い、と言われることがあります。

ロジック記述で大事なのは、シミュレーションで動作を詳細にテストする(テストベンチを記述することで、シミュレーションを行わずにFPGAやASICにロジックを組み込むことは、原則的にやってはいけません。ロジックのモジュールを記述したら、必ずその機能を全て試験するようなテストベンチを記述しましょう。ポリシーによっては、モジュールの実装を行うまえに、まずテストベンチの記述から始めて、そのテストベンチを正しくクリアすることを目標にして実際のロジック記述を開始する、という手法もあります。

IPコア

Intellectual Property Coreの略です。なんらかの「機能」を、FPGAやASIC用にコンパイルできるように、なんらかの記述言語を使って書き下した(実装した)もの。たとえば、SpaceWireの機能をつかさどるロジックの実装を、SpaceWire IPコアと呼んだりします。

実装

絵に描いた餅ではなくて、実際に機能やアイデアを形にしたもの。ソフトウェアやハードウェアの開発で、事前に決めておいた仕様にそって、モノをつくる作業のことを指します。また、その作業で完成したモノ自体のことも指します。

より詳しいドキュメント

SpaceWireの仕様やSpaceCubeのコンセプト、SpaceWireを用いたデータ取得系の詳しいことに関しては「SpaceWire/RMAP Library」のドキュメントを参照してください。また、各種SpaceWire I/Fボードに搭載されているUserFPGA上にロジックを作成するための方法は、「UserFPGA on SpaceWire I/F Board」のドキュメントを参照してください。

また、宇宙研でホスティングされているSpaceWire Wiki[C]にも、詳しい情報が載っているかもしれません。

重要なお知らせ

このチュートリアルはOdaka, Yuasa et al.の「SpaceWireのつなげかた」というドキュメントを一部ベースにしています。このチュートリアルの、とくにSpaceWireとRMAPの解説の部分は、[1]と[4]を参考にして記述されています。このチュートリアルで利用している画像や写真の一部は、[14]から引用しています。

このチュートリアルやSpaceWire/RMAP Library、UserFPGA_templateには、まだまだ不十分なところが多くあります。みなさんのフィードバックや、修正・加筆、デバグなどのコントリビューションが期待されています。このチュートリアルは、PDF版だけでなく、Mac用のワードプロセッサPagesで編集可能なファイルも配布していますから、修正・加筆は遠慮なく行っていただき、SpaceWire MLやSpaceWire Wikiで連絡いただけますでしょうか。できるだけ情報を共有し、ドキュメントとして残していくことで、たくさんの方が生産的に作業を進められるようにしましょう。

更新履歴

このドキュメントは以下のように改訂されてきました。改訂した方は追記してください。

2008-07-04 作成開始(湯浅)

2008-07-23 公開開始(湯浅)

2008-11-14 アップデート(湯浅)

連絡先

このドキュメントは下記の人たちによって作成されました。改訂作業などをした場合は連絡先を追記してください。

湯浅孝行 (yuasa at amalthea.phys.s.u-tokyo.ac.jp)

また、問題の報告や解決策、議論は、SpaceWire MLやSpaceWire Wiki[C]で行われています。簡単な質問でもかまいませんので、なるべく多くの人たちで問題を共有して、解決できればより良いので、困ったときやすばらしい開発成果ができたときは、ぜひMLに連絡してください。MLメンバーに登録するには、各機関のスタッフの方々を経由して、国分さん(JAXA/ISAS)経由で渡辺さん(同)に連絡してください(2008年07月現在)。

SpaceCubeのセットアップ / Setup a SpaceCube

全体像 / Overview

このセクションでは、次のことを行います。

- SpaceCubeのセットアップ(起動ディスク作成、ネットワークの設定)
- SpaceCube内のSpaceWire FPGAに書き込まれているSpaceWire IPコアの種類の確認
- 開発環境のセットアップ(Linux、 Mac)
- PCとシリアルケーブルで接続してコンソールとして利用
- ファイルの転送(FTP経由)
- SpaceWire/RMAP Libraryのおまけソフトを、Eclipseでビルド

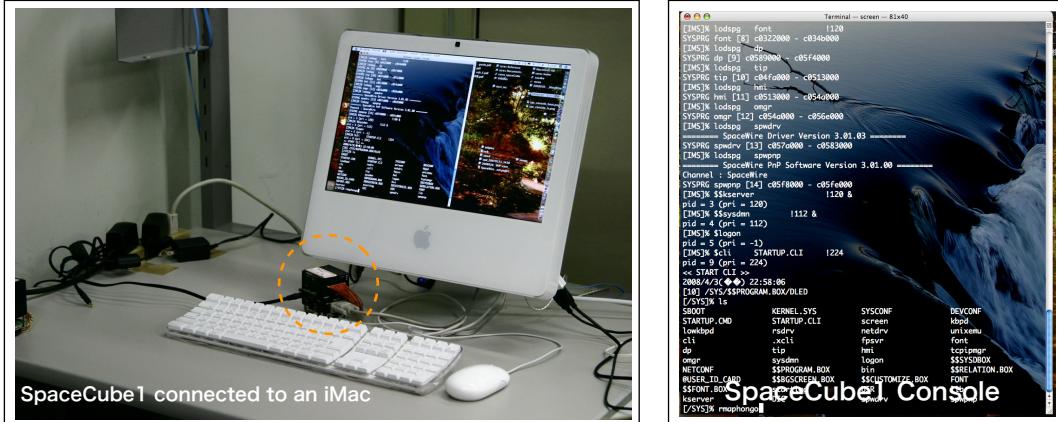
The following items will be covered in this section.

- To set up a SpaceCube (to create of a start-up disk and to configure network).
- To check the type of SpaceWire IP core on SpaceWire FPGA of the SpaceCube.
- To set up the development environment (on a Linux or a Mac)
- To use a SpaceCube as a console on a PC by connecting the SpaceCube by a serial cable.
- To transfer files via FTP.
- To build appendix softwares in SpaceWire/RMAP Library using Eclipse.

SpaceCube本体と開発環境+Eclipse CDTのセットアップ / Set up SpaceCube, development environment+Eclipse CDT

SpaceCubeは、それ自体にディスプレイやキーボードを接続して、単体のマシンとして利用することができますが、通常は、下の図のように、MacやLinux PCとシリアルケーブルで接続し、シリアルコンソール（コマンドラインのようなもの）を経由して、プログラムを実行したり、設定の変更を行ったりします。以下では、SpaceCubeをこのような状態で利用できるようにセットアップを行います。

SpaceCube is usually connected to a Mac or a Linux PC by a serial cable and a serial console (like a command-line terminal) is used to execute programs or to change settings, although SpaceCube may be used with a display and/or keyboard connected to itself. We set up the SpaceCube to be used under serial cable connection.



SpaceCubeのセットの確認、物品の準備 / Check of the SpaceCube set and preparation of materials.

シマフジ電機さんからSpaceCube1を購入すると、SpaceCube本体、CFカード、ACアダプタ、SpaceWire I/F分配ボード、分配ボード接続用ケーブル、開発環境やマニュアルが入ったCD-ROMが送られてきます。

A SpaceCube1 set, that can be bought from Shima Fuji, contains the A spaceCube, a CF card, an AC adaptor, a SpaceWire I/F distribution board, a cable to connect the distribution board, a CD-ROM including development environment and manuals.

SpaceCubeを利用するためには、他に必要なものとしては、以下のものが挙げられます。

- **Mac or Linux PC** SpaceCubeの開発を行う際の「親機」の役割をするコンピュータを用意してください。このドキュメントでは、ところどころMacの利用を仮定して説明を行っていきます
- **SpaceWireケーブル** 市販のストレートやクロスのシリアルケーブルとは内部配線が異なるので、代用できません。シマフジ電機さんからSpaceCubeとあわせて購入してください
- **クロス配線のシリアルケーブル** SpaceCubeと開発環境のMacやLinuxマシンを接続するためのケーブルで、市販のものを購入してください。大学の場合は、生協購買部で手に入るはずです。「ストレート配線」のものは使えないでの、クロスであることを確認して購入・使用してください
- **USB-シリアル変換ケーブル** 最近のパーソナルコンピュータにはRS-232Cタイプのシリアルポートは装備されていません。USBポートに接続して、シリアルケーブルを接続できるようにする製品が、各社から発売されているので、シリアルポートがないPCを親機として使う場合には購入する必要があります。Macユーザにお勧めなのは、RATOCのREX-USB60Fや、IO-DATAのUSB-RSAQ5という製品で、大学の場合は生協で簡単に手に入るようです
- **LANケーブル** SpaceCubeでTCP/IPネットワークを利用するときに必要です。後述しますが、FTPの利用などで、あったほうが便利なのでぜひ利用しましょう

The following materials are also necessary to use the SpaceCube.

- **Mac or Linux PC** It's a master computer to develop for a SpaceCube. This document often assumes you are using a Mac.
- **SpaceWire cable** Buy cables by Shima Fuji. The connection is different from standard straight or cross serial cables.
- **Serial cable (cross)** A cable to connect a SpaceCube and a Mac or Linux PC that is used as the development environment. Buy a commercial one. Make sure it is "cross"-type, not a straight type.

- **USB-Serial adaptor cable SB-シリアル変換ケーブル** Recent computers does not have any RS-232C serial port. Buy an adaptor cable to connect a serial cable to a USB port. For Mac users, RATO REX-USB60F or IO-DATA USB-RSAQ5 are recommended.
- **Ethernet cable** It is necessary to use a TCP/IP protocol on the SpaceCube. It is used for e.g., FTP.

CD-ROM内のドキュメントを親機(Mac, Linux)にコピー / To copy documents from CD-ROM to Mac/Linux.

付属CD-ROMには、SpaceCubeとその開発環境のマニュアルが、teacube/man/以下におさめられています。このドキュメントは頻繁に参照することになると思うので、開発環境用のマシンにコピーしておいてください。ドキュメントのファイル名と中身をまとめると、以下のようになっています。一番最初は、manual.pdfを読むと、だいたいの流れが掴めるのではないかと思います。

- **develop.pdf** SpaceCube用の開発環境(コンパイラ・リンカー)の使い方の説明
- **driver.pdf** SpaceCube用の各種ハードウェアドライバの説明
- **ext_man.pdf** T-Kernelの上位レイヤーであるT-Kernel/Standard Extentionのライブラリ関数の説明
- **hardware.pdf** SpaceCubeのハードウェア構成の説明
- **library.pdf** T-Kernelレイヤーのライブラリ関数の説明
- **manual.pdf** SpaceCubeのセットアップ、開発環境のインストールなどの説明
- **tsh_guide.pdf** T-Kernelの上で動作する、TCP/IPスタックやGUIライブラリなどの説明

The attached CD-ROM contains in teacube/man/* manuals of the SpaceCube and its development environment. You'd better copy the documents to the computer for development (Mac or Linux) because they are often referred. The summary is as follows. Overview is shown in manual.pdf and it is recommended to read first the document. However, they are all written in Japanese.

- **develop.pdf** Showing how to use the compiler and linker for a SpaceCube.
- **driver.pdf** Showing hardware drivers for a SpaceCube.
- **ext_man.pdf** Showing library functions of T-Kernel/Standard Extension, which is a higher layer of T-Kernel.
- **hardware.pdf** Showing a hardware configuration of a SpaceCube.
- **library.pdf** Showing library functions of T-Kernel layer.
- **manual.pdf** Showing how to set up SpaceCube and how to install its development environment.
- **tsh_guide.pdf** Showing TCP/IP stack and GUI libraries on T-Kernel.

開発環境(コンパイラ・リンカー)とEclipse CDTを親機にインストール / To install development environment (compiler and linker) and Eclipse CDT in the master computer.

SpaceCubeは、セルフコンパイル環境を持たないので、ユーザが記述したプログラムは、親機の上でSpaceCube用の実行形式としてコンパイルする必要があります(これをクロスコンパイル、といいます)。SpaceCubeは、MIPSというタイプのCPUを利用しているので、それに対応したコンパイラ(実際には、gccのmipsオプションを有効にしたもの)が必要になるので、親機側でそのインストールを行います。

SpaceCube does not have self-compiling environment. So the program written by a user must be compiled on the master PC as an executable file on SpaceCube. This is called cross-compiling. Since the CPU of the SpaceCube is MIPS type, a compiler that supports MIPS should be used (this is done by mips option of gcc). Here we install files in the master PC.

SpaceCube用の実行形式を生成してくれるコンパイラやリンカーは、teacube/soft/以下におさめられています。Linuxを親機として利用する場合は、親機に/usr/local/teというディレクトリを作成して、そこへCygwinという名前が入ったファイル以外全部のtar.gz形式のファイルをコピーし、その場で展開してください。すでにコンパイルされたコンパイラやリ

ンカーの実行形式も展開されるので、インストールはこれで完了です。後述の、shellの環境変数の設定を行えば、クロスコンパイル環境を利用できるようになります。

The compiler and linker to create executables for a SpaceWire is in teacube/soft. If you use a Linux PC, make a directory /usr/local/te and copy all tar.gz files except those containing "Cygwin" in the filename, and then extract them. It also installs a pre-compiled compiler and linter. Once you set the shell environmental variables, the cross-compiling environment will be available.

Macを親機として利用する人も、Linuxの場合と同様に、まずはCygwinという名前が入ったファイル以外全部のtar.gz形式のファイルをコピーし、その場で展開してください。続いて、SpaceWire Wiki[C]にアクセスして「SpaceCube用コンパイラのMacintosh用バイナリ」というページに置いてある、Mac用に再コンパイルしたバージョンの実行形式をダウンロードしてください（[直接トップページからリンクが張られていないので、一覧から探して下さい](#)）。ダウンロードしたアーカイブを展開すると、usr/local/te/tools/unknown/...というフォルダが生成されます。unknown/以下に、コンパイル済みのgccやnm、arなどがおさめられているので、このディレクトリを、/usr/local/te/に移動してください。たとえば、デスクトップでアーカイブを展開した場合は、以下のようにコマンドを入力することになると思います。

If you use a Mac, first copy all tar.gz files except those containing "Cygwin" in the filename, and then extract them. Then access to SpaceWire Wiki [C] and download pre-compiled executables, which are on "SpaceCube用コンパイラのMacintosh用バイナリ". Folders called user/local/te/tools/unknown/... are created when you extract the archive. The pre-compiled gcc, nm, ar, etc., are under unknown folder. Move the directories to /user/local/te. If you extract the archive on the Desktop, you will use the following commands.

```
cd ~/Desktop  
mv usr/local/te/tool/unknown /usr/local/te/tool
```

コンパイラなどのインストールが終わったら、あとはshellの環境変数の設定を行います。bashを使っている場合は、~/.bash_profileなど、起動時に自動的に読み込まれるファイルに、以下の記述を追加してください(環境によっては、~/.bashrcが読まれる場合もあります)。tcshなどを使っている人は、これを機会にbashに乗り換えましょう:-)。

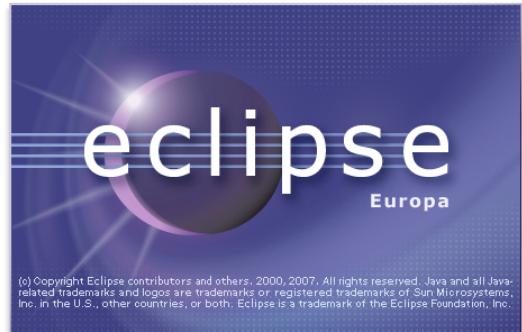
After installation of the compiler, the shell environmental variables should be set. If you are using bash, add the following lines in ~/.bash_profile (or other files that are read at startup).

```
# Linux PC版  
export BD=/usr/local/te  
export GNUs=/usr  
export GNU_BD=$BD/tool/Linux-i686  
export GNUMips=$GNU_BD/mips64el-unknown-tkernel  
export GCC_EXEC_PREFIX=$GNU_BD/lib/gcc-lib/  
export PATH=$PATH:${GNU_BD}/etc  
  
# Mac版  
export BD=/usr/local/te  
export GNUs=/usr  
export GNU_BD=$BD/tool/unknown  
export GNUMips=$GNU_BD/mips64el-unknown-tkernel  
export GCC_EXEC_PREFIX=$GNU_BD/lib/gcc-lib/  
export PATH=$PATH:${GNU_BD}/etc
```

また、SpaceCubeのプログラム開発を行うにあたって、テキストエディタとターミナルでの手動コンパイル、printfを多用したデバッグ作業、手動のファイル管理などといったこれまでのスタイルから、より効率的な、現代的な開発を行うために、このドキュメントでは、親機の開発環境として、統合開発環境Eclipse IDE[G]の使用を前提とします。
Eclipseは、オープンソースのIDEで、もともとはJavaのプログラムを開発するために構築されましたが、現在ではC/C++、Ruby、HTMLなどさまざまな言語の記述・コンパイル・デバッグ環境として世界的にひろく利用されています。とくにSpaceCubeの開発では、EclipseにC++開発環境のプラグインを加えた、Eclipse CDTというバージョンを利用します。

We assume that you use Integrated Development Environment Eclipse IDE [G] as a development environment on the master PC.

<http://www.eclipse.org/downloads/>にアクセスして、「Eclipse IDE for C/C++ Developers」からパッケージをダウンロードし、インストールしてください(「Eclipse IDE for Java Developers」とは異なるので注意してください)。ありがたいことに、Googleで「Eclipse CDT インストール」と検索すると、たくさんの例が見つかります。Eclipseはとても強力で、膨大な機能を備えているので、具体的な利用方法はwebや書籍[dなど]を参照していただくことになりますが、プロジェクトの作成、コンパイル、デバッグの簡単なwalkthroughは、このドキュメントの後の方で述べます。



Access to <http://www.eclipse.org/downloads/> to download and install "Eclipse IDE for C/C++ Developers". Note that this is different from "Eclipse IDE for Java Developers".

ケーブルの接続、電源投入 / Connecting cable, power on.

まずは、CFを入れずに、SpaceCubeの内蔵メモリに書き込まれているT-Kernelを利用して起動し、親機との接続を確けめます。シリアルケーブル(クロス)をSpaceCubeと親機PC/Macに接続してください(親機側でUSB-シリアル変換ケーブルを利用している場合は、その製品のドライバをインストールしておいてください)。

First, turn on the SpaceCube using T-Kernel written in the internal memory, without CF connected, in order to check the connection to the master computer. Connect the serial cable (cross) in advance, between the SpaceCube and the master PC/Mac (If you use USB-Serial adaptor on the master PC, install in advance the driver of the adaptor).

親機側で、ターミナル(Macなら「ターミナル.app」、Fedoraなら「ターミナル」)を開いて、gterm*やscreenといった、ターミナルプログラムを通してSpaceCubeと接続します。その際、シリアルポートデバイスにアクセスするので、OSによってはシリアルポートデバイスのパーミッションを変更する必要があるかもしれません。以下のようにして、ターミナルソフトを起動します。

Opne a terminal (Terminal.app on Mac; Terminal on Fedora) and use gterm (Linux) or screen (Mac). Hereafter we assume you are using Mac. For some situation, the permission of serial port device needs to be changed. Examples of using terminal software are shown below.

*gtermは、SpaceCubeの開発環境に同梱されているプログラムなので、利用するためには「開発環境(コンパイラ・ linker)とEclipse CDTを親機にインストール」の手順をふんで、開発環境をインストールしてある必要があります。

```

# gtermを使う場合(さきほど設定した環境変数が有効になっている必要あり; environmental variable must be set in
advance)
gterm -X -b -B -l/dev/ttys0    <== シリアルポートとして/dev/ttys0を利用する場合 (in the case of using
/dev/ttys0 as the serial port).

# Mac版(screenという便利なソフトが標準で入っているのでそれを使いましょう; use "screen", which is installed
as a part of OS)
screen "/dev/tty.I-O DATA USB-RSAQ5" 115200
    <== シリアルポートとして、変換ケーブル(USB-RSAQ5)を利用する場合 (in the case of using USB-RSAQ5 as the
USB-Serial adaptor).

```

これらのソフトの終了方法は結構特殊なので、念のため書いておくと、gtermが「Ctrl-C Ctrl-C .q」で、screenが「Ctrl-A Ctrl-¥」です。screenに関しては「Ctrl-A ?」と入力すると、ヘルプがでます。

To quit the software, use "Ctrl-C Ctrl-C .q" for gterm and "Ctrl-A Ctrl-\" for screen. Help is available for screen by "Ctrl-A ?".

さて、ターミナルソフトを起動した状態で、SpaceCubeに電源ケーブルを接続して、起動してみてください。以下のような起動メッセージが画面に表示されるはずです。表示されない場合は、まわりのスタッフやシニアなメンバーに、ケーブルの接続、親機のパーミッションの設定、USB-シリアル変換ケーブルのドライバのインストール、ケーブルの接続などを、もう一度確認してもらってください。どうしてもこの段階でSpaceCubeとうまくつながらない場合は、SpaceWire MLなどに問い合わせてください。

Start the SpaceCube by connecting the power cable, with the terminal software running. You will see the following startup message. If not, check the cable connection, permission of the master PC, device driver of the USB-serial adaptor.

PMC T-Kernel/VR5701 Spacecube + Extension Version 1.0.00 Copyright (C) 2005 by Personal Media Corporation ** SegmentMgr OK ** ProcessMgr OK ** MemoryMgr OK ** MessageMgr OK ** TaskCommMgr OK ** GNameMgr OK ** DeviceMgr OK ** ClockMgr OK ** EventMgr OK ** FileMgr OK ** FPUEmu OK ** UsbMgr OK ** ClockDrv OK ** SysDiskDrv OK <<< IMS >>> [IMS]% lodspg screen !35 SYSPRG screen [1] c0209000 - c0210000 [IMS]% lodspg kbd !30 SYSPRG kbd [2] c01fa000 - c0206000 [IMS]% lodspg lowkbpd !28 SYSPRG lowkbpd [3] c021f000 - c0227000 [IMS]% lodspg rsdrv !26 SYSPRG rsdrv [4] c0215000 - c021c000 [IMS]% lodspg netdrv !23 SYSPRG netdrv [5] c023c000 - c0245000 [IMS]% lodspg tcipmgrp	SYSRPG tcipmgrp [6] c028a000 - c02c4000 [IMS]% lodspg unixemu SYSRPG unixemu [7] c0247000 - c0282000 [IMS]% lodspg font !120 SYSRPG font [8] c0320000 - c0349000 [IMS]% lodspg dp SYSRPG dp [9] c0587000 - c05f2000 [IMS]% lodspg tip SYSRPG tip [10] c04f8000 - c0511000 [IMS]% lodspg hmi SYSRPG hmi [11] c0511000 - c0548000 [IMS]% lodspg omgr SYSRPG omgr [12] c0548000 - c056c000 [IMS]% \$kserver !120 & Can't find /SYS/kserver [IMS]% \$\$sysdmn !112 & pid = 3 (pri = 112) [IMS]% \$logon pid = 4 (pri = -1) dictionary setup err : -458752 [IMS]% \$cli STARTUP.CLI !224 pid = 8 (pri = 224) << START CLI >> 2008/7/16(øå) 23:00:46 [9] /SYS/\$\$PROGRAM.BOX/DLED [/SYS]%
---	---

この状態で何か文字を入力すると、通常のコマンドラインと同じように動作します。これでとりあえず電源が入って、親機と接続できることは確認できたので、いったんSpaceCubeの電源を落とします。SpaceCubeはソフトウェア的に電源を落とすしきみがないので、通常、電源ケーブルを抜いて電源OFFにします。

If you type some character, the screen works as a normal command-line terminal. Now you've checked the start-up and the connection to the master PC. So turn off the SpaceCube, by disconnecting the power cable.

CFのフォーマット(起動ディスクの作成) / Formatting CF to create a start-up disk → [manual.pdfの7ページ参照](#)

さきほどは、SpaceCubeの内蔵ROMに書き込まれているT-Kernelを用いてSpaceCubeを起動しましたが、内蔵ROMは64MBしか容量がなく、ユーザプログラムなどからデータを書き込むことができないので、代わりに、外部のCFにシステムを書き込んで、そのCFをメインのディスクとして利用し、プログラムやデータの保存を行います。シマフジ電機さんからSpaceCubeを購入したときに同梱されているCFには、起動用のシステム(T-Kernel)が書き込まれていないので、附属CD-ROMに入っているmanual.pdfの7ページを参照して、起動ディスクを作成してください。起動ディスク作成後は、ONにした本体のDIPスイッチをOFFに戻して、常時CFから起動するようにしておいてください。

2010年6月現在、シマフジさんから納入されたCFには起動ディスクがあらかじめ書かれているようです。

OBSOLETE: this is already done when the SpaceCube is purchased. No need to do this.

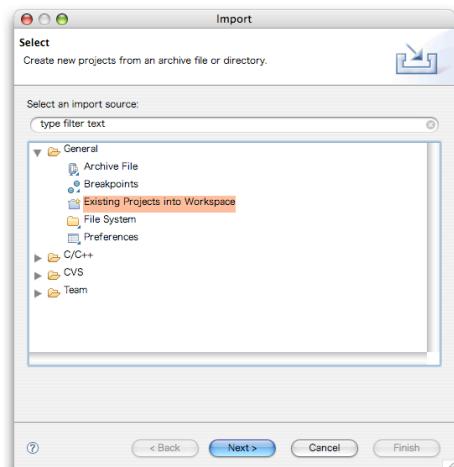
SpaceWire FPGAに焼き込まれているIPコアの種類の確認 / Check IP core

version of SpaceWire FPGA.

SpaceCube内のSpaceWire FPGAに書き込まれているSpaceWire IPコアが、どのメーカーさん製の、どういったバージョンのものか、によって、使用するドライバが異なるので、いま使っているSpaceCubeが何バージョンか、購入時などに確認する必要があります。

2008年8月から、SpaceWireユーザ会のJAXA/大学側の標準IPコアは、NEC Soft製のものから、シマフジ電機製のものに変更されました。これにより、転送速度が少なくとも10倍以上に向上しています。IPコアのソースコードはシマフジ電機さんのwebsiteで公開されており、SpaceCubeへの書き込み作業は、SpaceCubeをシマフジ電機さんに送付すれば、実費で行ってくれます。

We have no way to check the IP core of Shimafuji boards. Ask ISAS.



シマフジ電機製SpaceWire IP Coreの利用方法 / How to use Shimafuji SpaceWire IP core.

シマフジ電機製IPコアを利用する場合は、SpaceWire/RMAP Libraryをビルド(後述)すると、SpaceCube用ドライバも自動的に生成されます。ライブラリを、/usr/local/teという標準の場所にインストールして開発を行っている場合は、Eclipseでのビルド後、/usr/local/te/SpaceWireRMAPLibrary/driver/spc1_shimafuji3/spc_driver/spwというドライバモジュールの実行系式ができます。これを、SpaceCubeに転送して、lodspgコマンドを実行することでシステムに組み込んでSpaceWire I/Fを利用することができます。

If you use Shimafuji IP core (As of June 2010, Shimafuji IP core is used on SpC), SpaceWire driver for SpaceCube is created when SpaceWire/RMAP library is built (see following sections). If you install the SpaceWire/RMAP library to /usr/local/te (the standard location), an executable of the driver module /usr/local/te/SpaceWireRMAPLibrary/driver/spc1_shimafuji3/spc_driver/spw is created at building by Eclipse. By transferring it to SpaceCube and using lodspg command, SpaceWire I/F will be available on SpaceCube.

```
[/SYS]% fget himawari:/usr/local/te/SpaceWireRMALibrary/driver/spc1_shimafuji3/spc_driver/spw
UserName: yuasa
PassWord: (パスワード入力)
220 himawari.phys.s.u-tokyo.ac.jp FTP server (tnftpd 20061217) ready.
.....(転送中).....
[/SYS]% lodspg spw (リターン)
SYSPRG spw [13] c060b000 - c0613000 <= ドライバをロードしたアドレスが表示される
[/SYS)%
```

さらに詳しい説明は、documets/HowToUseShimafujiSpaceWirePCore.textを参照してください(どなたか、もう少し詳しい説明書を作成していただけますと助かります)。

You can find more detailed information in documets/HowToUseShimafujiSpaceWirePCore.text

SpaceCubeのネットワーク設定 / Network setting of the SpaceCube

SpaceCubeにEthernetケーブルを接続して起動すると、デフォルトではDHCPでIPアドレスを取得します。固有のIPアドレスを名乗らせたい場合には、netconfコマンドを「-c」オプション付きで実行して、設定を変更します。設定情報を確認する場合は、netconfコマンドを引数なしで実行してください。

If a SpaceCube is turned on with a Ethernet cable connected, it gets an IP address from a DHCP server. To specify an address explicitly, use netconf command with an option -c. To check the current network information, use netconf without option.

```
[/SYS]% netconf -c (リターン)
hostname = ? momi <= 以下、必要事項を入力していく (input required information)
host ip = 0.0.0.0 ? 133.11.165.83
dns1name = ? 133.11.1.1
dns1 ip = 0.0.0.0 ? 133.11.1.1
dns2name = ? (リターン)
dns2 ip = 0.0.0.0 ? (リターン)
domain = ? phys.s.u-tokyo.ac.jp
gateway ip = 0.0.0.0 ? 133.11.165.126
subnetmask = 255.255.255.0 ? 255.255.255.128
wlan = none (n/a/i)? n

[/SYS]% netconf
hostname = momi
host ip = 133.11.165.83
(以下、設定した情報が表示される; current setting is shown)
```

FTPの利用方法 / How to use FTP

シリアルケーブルでの実行形式の転送は、かなりの時間がかかるため、バグの修正と実機での実行を繰り返すデバッグの段階では、相当な時間ロスになりかねません。SpaceCube1のおまけコマンドとして提供されている、fgetというプログラムを使うと、FTP経由でLAN内のサーバからファイルをダウンロードすることができ、こちらの方がシリアル通信よりも高速なので、利用をおすすめします。とくに、UNIXやMacintoshで開発を行っている場合は、そのマシン単体で簡単にFTPサーバーデーモンを実行できます。

By using fget command, files can be downloaded from a FTP server in the LAN to a SpaceCube. Using FTP is recommended because it is much faster than file transfer based on serial connection. If your master computer is either UNIX, Linux or Mac, FTP daemon is easily available.

例えばMacなら、「システム環境設定」→「共有」→「ファイル共有」のオプションの中の「FTPを使用してファイルやフォルダの共有」のチェックボックスをチェックするだけです。また、Fedoraの場合には、「システム」メニュー→「管理」→「サービス」を開いて、vsftpdというサーバにチェックを付けて「開始」するとFTPサービスが稼働します(イ

ンストールされていない場合は、「アプリケーション」メニュー→「ソフトウェアの追加と削除」から、「サーバー」→「FTPサーバー」を追加インストールすればOKです)。

If you are using a Mac, Go to "System preferences" -> "Sharing" -> "File Sharing" and go to "Options" -> "Share files and folders using FTP".

開発環境のマシンでFTPサーバを起動したあとで、SpaceCube1のコマンドラインから、

```
[/SYS]% fget (サーバ名):(ファイルパス) (保存する名前)
```

と入力し、ユーザ名とパスワードを入れれば転送が行われます。保存する名前は、ファイル名を変更しないときは省略できます。下の例では、main_rmaphongoという実行形式(コマンドライン版のRMAPHongo)を、SpaceCubeに転送する際に、rmaphongoという風に名前を省略しています。

After a FTP daemon starts up on the master PC, type on SpaceCube command line terminal

```
[/SYS]% fget (server name):(file path) (filename on SpaceCube)
```

then by a correct username and password, the file will be transferred. If you dont change the file name,"filename on SpaceCube" can be omitted. In the example below, "main_rmaphongo" is renamed as "rmaphongo" when transferred.

```
[/SYS]% fget himawari:/usr/local/te/SpaceWireRMAPLibrary/build/t-kernel/main_rmaphongo rmaphongo
UserName: yuasa
PassWord: (パスワード入力)
220 himawari.phys.s.u-tokyo.ac.jp FTP server (tnftpd 20061217) ready.
.....(転送中).....
.....
.....
.....
2,866,208 bytes transferd. (409,458 Bytes/sec)
[/SYS]%
```

注：FTPでは、ユーザ名やパスワードが平文でネットワークを流れるので、各機関のネットワークポリシーによっては、別の対応(例えばSpaceCubeと開発コンピュータだけで、他から切り離されたネットワークにする、など)が必要になかも知れません。

Note: the username and password is not encrypted in FTP.

EclipseをもちいたSpaceWire/RMAP Libraryとおまけソフトのビルド / Building SpaceWire/RMAP Library and appendix softwares using Eclipse.

SpaceWire/RMAPを行って、データ取得をする場合には、SpaceWire/RMAP Library (RMAPHongo)に含まれている、SpaceWire I/Fをカプセル化したクラスや、RMAPパケットの生成・解釈を自動化してくれるクラスを利用することになります。SpaceWire/RMAP Libraryには、ライブラリとして利用できるクラス群以外に、それらを用いて作成したおまけソフトとして、コマンドラインからRMAPを利用できる「rmaphongo」や、SpaceWire/RMAPのデータ転送速度を計測する「speedtest」、SpaceWire I/FをTCP/IP経由で利用可能にするプロトコルブリッジ「sthongo」などが含まれています。次のセクションでは、そのおまけソフトを使って、SpaceCubeからSpaceWire I/FボードとSpaceWire/RMAPを用いた通信を行うので、ここで事前にライブラリ本体とおまけソフトをビルド(コンパイル)しておきます。

SpaceWire/RMAP Library (RMAPHongo) includes classes that encapsulates SpaceWire I/F and classes that automates creation and interpretation of RMAP packets. Data acquisition by SpaceWire/RMAP is utilized these classes. Some appendix softwares that are also provided as a part of SpaceWire/RMAP Library, besides classes that can be used as library. The softwares contains rmaphongo, which enables to use RMAP by command line, speedtest, which measures the data

transfer speed of SpaceWire/RMAP, and sthongo, which is a protocol bridge that enables SpaceWire I/F via TCP/IP. Since we will use the appendix softwares in the next section for SpaceWire/RMAP communication from SpaceCube to SpaceWire I/F board, we here build (compile) the software as well.

なお、このドキュメントでは、SpaceWire/RMAP Libraryの構造や使い方については、大まかにしか説明しません。詳細な情報は、「SpaceWire/RMAP Library」ドキュメント[15]や、SpaceWire/RMAP Library Online API Reference[H]を参照してください。

See also "SpaceWire/RMAP Library" document [15] or SpaceWire/RMAP Library Online API Reference[H] for more information.

プロジェクトパッケージのダウンロードとEclipseへのインポート / Downloading project package and importing it to Eclipse

SpaceWire/RMAP Libraryは、SpaceWire Wiki[C]などから最新版がダウンロードできるので、ダウンロード後、zip圧縮を展開してください。展開後生成されるフォルダは、Eclipseの「プロジェクト」になっているので、Eclipse側から「インポート」することで、簡単にEclipse上で作業ができるようになります。Eclipseの「File」→「Import」とたどると出てくるウインドウの中で、インポートする形式が聞かれるので、「General」→「Existing Projects into Workspace」を選択し、「Next」ボタンをクリックします。次の画面では、どのフォルダをプロジェクトとして取り込むか、が聞かれるので、「Browse」ボタンをクリックしてフォルダ選択ダイアログを開き、さきほど展開しておいた「SpaceWireRMAPLibrary」のフォルダを選択します。あとは、関連するプロジェクトも読み込むかなどが聞かれますが、デフォルトの設定のままOKをクリックしていけば、プロジェクトがWorkspaceにインポートされます。

The latest version of SpaceWire/RMAP Library can be downloaded from, e.g., SpaceWire Wiki[C]. The zip file is a compressed folder that is "Project" of Eclipse. So once it is "imported" to Eclipse, all the building work easily becomes available on Eclipse. Click "File" -> "Import" in Eclipse, and the file format of the imported file is asked. Select "General" -> "Existing Projects into Workspace" and click "Next". The folder to be imported is asked in the next window. Click "Browse" to open a file selection window and select the folder "SpaceWireRMAPLibrary" that has been extracted. You will also be asked whether related projects should be imported. Just click OK with the default selections, and the project will be imported on Workspace.

環境変数の設定とビルトの実行 / Setting environment variables and building

WorkspaceにインポートされたSpaceWireRMAPLibraryプロジェクトは、「コマンド+B」(Mac)もしくは「Ctrl + B」で、コンパイル・リンクなどの一連のビルト作業が自動的に行われます。最初のビルトには、ちょっと時間がかかるかもしれません。もし、SpaceCube用の開発環境を/usr/local/te以外の場所にインストールしている場合は、Workspace内のSpaceWireRMAPLibrary/build/t-kernel/Makefileをダブルクリックしてエディタで開き、6行目の「BD = /usr/local/te」の部分を、適切なパスに変更しておいてください。なお、SPWLIB/build/Makefile の SPACEWIRERMAPLIBRARY_PATH が /usr/local/te でハードコードされているなど、/usr/local/te にインストールされていることが想定されている箇所があるので、他の場所にインストールした場合は変更が必要です。また、Perl の path が /usr/local/bin/perl にあることも仮定されているので、その場所に perl がない場合は、必要に応じてファイルを書き換えるなり /usr/local/bin/perl に symbolic link を貼るなり対応して下さい。

SpaceWireRMAPLibrary project is built, i.e., automatically compiled and linked, by "Command+B (Mac)" or "Ctrl+B". The first build may take a long time. If development environment for SpaceCube is installed somewhere other than /usr/local/te, modify "BD = /usr/local/te" accordingly, in the sixth line of SpaceWireRMAPLibrary/build/t-kernel/Makefile using an editor (double-clicking of the file). Some other source codes also assume the environment is installed in /usr/local/te, such as "SPACEWIRERMAPLIBRARY_PATH" in SPWLIB/build/Makefile. Some files assume Perl is installed in /usr/local/bin/perl.

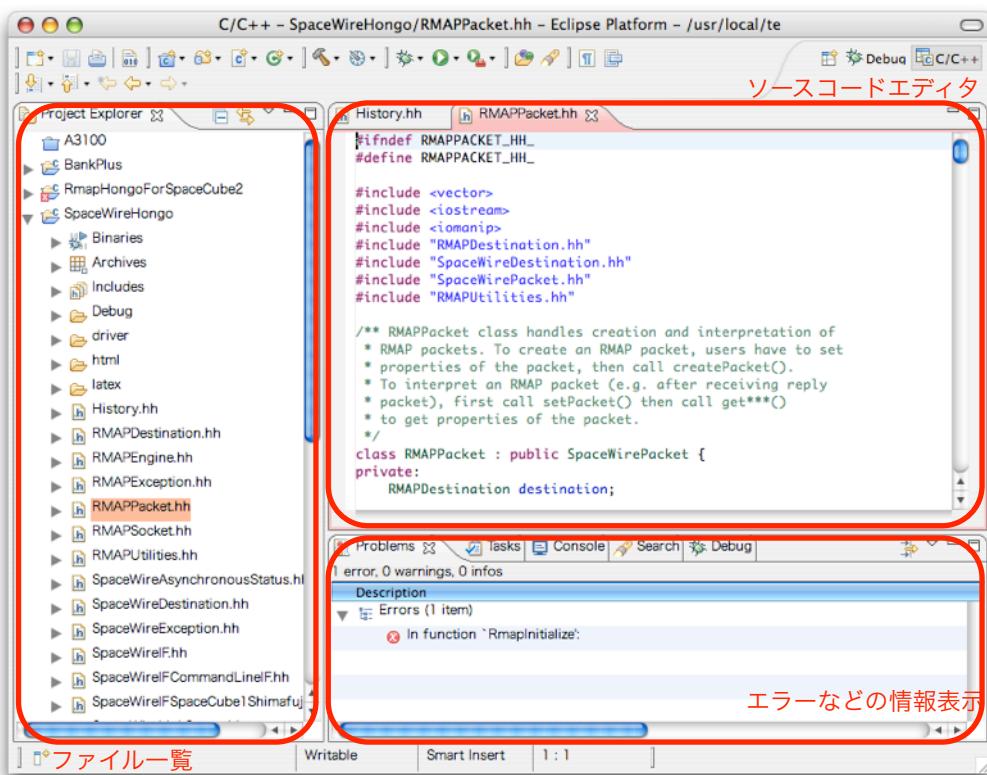
ビルトがエラーなく完了すると、SpaceWireRMAPLibrary/executables/t-kernel/以下に、おまけソフトや例題プログラムのSpaceCube用実行形式が配置されます。ちなみに、t-kernelだけでなく、posixというフォルダもありますが、まったく同じソースコードから、posix、つまり、標準的なLinuxやMacで動作する実行形式も生成されている、ということです。

地上実験から衛星搭載までシームレスに開発を行いたいので、SpaceWire/RMAP Libraryでは、ハードウェアやOSに依存する部分をなるべく減らし、移植性 portabilityを高めるということが、重要と考えています。

When the build is finished without errors, the executables of appendix and sample programs for SpaceCube is stored under SpaceWireRMAPLibrary/executables/t-kernel/. In posix folder, executables in posix, i.e., standard Linux or Mac, are also stored. We consider the portability as important, because the development should be done seamlessly from on-ground development to flight software. So hardware- or OS- dependency is excluded as much as possible.

自分でSpaceWire/RMAPの機能を使ったユーザプログラムを記述した場合も、「コマンド+B」と入力するだけで簡単にビルドができます。エラーが発生した場合にも、エラー箇所へのジャンプやハイライト表示、エラーの原因のポップアップなどを利用すれば、すぐに間違いを見つけ出して修正することができます。自分でユーザプログラムを作成する方法は、「SpaceWire I/Fを使ったプログラムの開発」のセクションで説明します。

When you write a your own user program using SpaceWire/RMAP function, it can be easily built by "Command-B". Modification will be easily done by utilizing jumping to the error location, highlighting, popping-up of error reasons. More information for writing your own user program is found in the section "Development of programs using SpaceWire I/F".



SpaceWire I/Fボードを接続 / Connecting SpaceWire I/F board

全体像 / Overview

このセクションでは、次のことを行います。

- SpaceWire I/Fボードの構造を理解する
- SpaceWire I/Fボードに必要なものを準備する(電源装置、電源ケーブル、SpaceWireケーブル)
- SpaceWire FPGAに焼き込まれているIPコアを確認する
- SpaceCubeと接続し、コマンドライン版rmaphongoを用いて、SpaceWire/RMAP通信経由でLEDを点滅させる
- SDRAMのread/writeを試験する

The following items will be covered in this section.

- To understand the structure of SpaceWire I/F board.
- To prepare required materials for SpaceWire I/F baord (power supply, power cable, SpaceWire cable).
- To check IP core written in SpaceWire FPGA.
- To communicate with SpaceCube by command-line program rmaphongo, and to blink LEDs via SpaceWire/RMAP.
- To test reading/writing of SDRAM.

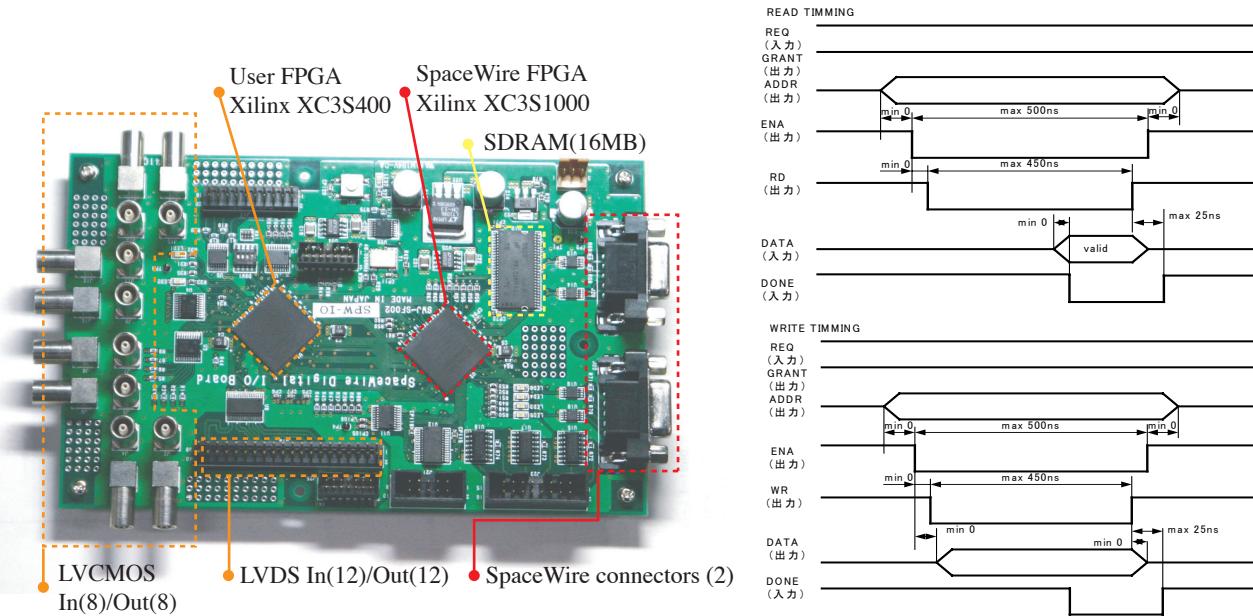
SpaceWire I/Fボードの構造 / Structure of the SpaceWire I/F board

SpaceWire I/Fボードは、検出器などからのアナログ信号や、独自の形式のデジタル信号を受け取って、SpaceWireネットワークからアクセス可能な形式に変換や処理を行うのが役割です。ユーザが各自の実験でフレキシブルに利用できるように、ユーザが書き換え可能なFPGA(UserFPGAと呼びます)と、データを保持しておくためのSDRAMが搭載されています。また、SpaceWire I/Fを提供するためのSpaceWire FPGAには、各メーカーさん製のSpaceWire IPコアが搭載されています。

A SpaceWire board is to receive analog signals from e.g., detectors or arbitrary format digital signals and to convert them to format that is accessible from SpaceWire network. It is equipped with a FPGA that is programmable by a user (called UserFPGA) and a SDRAM, in order for users to use it suitable for their own measurements. Another FPGA, which provides SpaceWire I/F (called SpaceWire FPGA), is not programmable. SpaceWire IP core is installed on it.

UserFPGAとSpaceWire FPGAの間は、「External Bus」という名前のシンプルなデータバスで接続されています。SpaceCubeなどからのRMAP Read/Writeの要求は、SpaceWire FPGAにおいてExternal Busへのアクセスに変換され、UserFPGAに伝達されます。UserFPGAは、例えばReadアクセスの場合には、アクセスされたアドレスに対応するデータを、External Bus経由で返答することで、SpaceCubeにデータを返すことができます。以下の写真では、シマフジ電機製SpaceWire DIOボードの構成を示しています。また、External BusのRead/Writeアクセス時のタイミングチャートも掲載しておきます(シマフジ電機, SpaceWire DIOボードマニュアルより引用)。

UserFPGA and SpaceWire FPGA is connected by a simple data bus called "External Bus". RMAP Read/Write request from e.g., SpaceCube is converted to an access to External Bus by the SpaceWire FPGA and transferred to UserFPGA. UserFPGA replies to SpaceCube the data at the specified address in the case of RMAP read access, through the External Bus. The photo shows the structure of DIO board. Timing charts are also shown.



補足：2008年7月時点では、External Busの仕様はメーカーさんごとに異なっているようです。バスアクセスの信号のタイミングなどが違う場合、UserFPGA内のバスアダプタを書き換える必要があるかもしれません。

以下の作業では、SpaceWireケーブルを接続するコネクタ番号などは、シマフジ電機製SpaceWire AD/DAボード、もしくはDIOボードを仮定して話をすすめます。他のボードを使用している場合は、ボード購入時に、電源電圧やアドレスマップ(何番地にアクセスすればUserFPGAに到達し、何番地から何番地までがSDRAMに割り当てられているか、など)が掲載された、ある程度のマニュアルが同梱されているはずなので、それらも参照しながら読みすすめてください。

The following procedure assumes you use Shimafuji AD/dA board or DIO1 board. If not, you must check power voltage, address map (the address of UserFPGA, SDRAM, etc.) from a manual.

物品の準備 / Preparation of materials

以下の物品を準備してください。

- SpaceWire I/Fボード** NTSpaceさん、シマフジ電機さんやClear Pulseさん、MHIさんなどから入手できるものと思われます。ミッションごとに、使用するメーカーさんが異なるかもしれません、このチュートリアルではシマフジ電機製ボードを仮定します
- ボードのマニュアル** 購入時に添付される(はず)の、ボードのマニュアルを用意してください。回路図、アドレスマップは、開発の際に必ず必要になります
- 電源装置** 一部のボードは、電源投入直後のFPGAのコンフィグレーションの際に、2A以上の電流を引き込むものがあります。3A程度の電流を供給できる電源装置を使用してください。また、FPGAに間違ったロジックを構築してショートが発生した場合に、電流を制限するために、カレントリミッタが装備された電源装置の使用をおすすめします。湯浅の個人的なオススメは、TEXIO(Kenwood)のPW-Aシリーズです。プリセットの記憶個数が5個もあったり、出力電圧設定のインターフェースが使いやすいです

- **電源ケーブル** シマフジ電機製SpaceWire I/Fボードなどでは、購入時に電源ケーブルが同梱されているので、それを使用してください。電源装置によっては、コネクタ部分をバナナプラグやY字プラグなど適切な形に自分で改造する必要があるかもしれません(たとえば上記のTEXIOの電源だと、そのままの状態で接続できます)
- **SpaceWireケーブル** 市販のシリアルケーブルでは代用できません。内部配線が異なります。また、LVDS信号を通すので、等長配線、インピーダンス整合など、厳密にいうと細かい規定がたくさんあるため、シマフジ電機さんなどからちゃんとしたものを購入することをお勧めします。とはいっても、実験室で使うレベルの信頼性でよければ、SpaceWireの仕様書の配線図を見ながら、手作りすることも可能です

Prepare the followings.

- **SpaceWire I/F board** It can be purchased from Shimafuji, NTSpace, Clear Pulse, MHI, etc. In this tutorial, we assume you are using a Shimafuji board.
- **Manual of the board** Prepare a manual attached in the delivered package. Circuit diagram, address map are necessary for development.
- **Power supply** Some boards dissipates more than 2 A at start-up. Prepare power supply that can provide more than 3 A. One with a current limiter is recommended in case an incorrect logic of FPGA causes a short circuit. TEXIO (Kenwood) PW-A series are good.
- **Power cable** Use attached power cable if you use Shimafuji board. You may need to prepare an adaptor or connectors for some power supply.
- **SpaceWire cable** It is not a commercial serial cable because the wiring is different.

SpaceWire FPGAに焼き込まれているSpaceWire IPコアの確認 / Check SpaceWire IP core installed in SpaceWire FPGA

SpaceWire I/FボードのSpaceWire FPGAも、SpaceCubeのそれと同様に、いくつかのメーカーさんによって提供されており、機能が違うことがあります。しかし、IPコアごとにドライバの差し替えが必要なSpaceCubeの場合と違って、SpaceWire I/Fボードの方は、必ずRMAPのターゲット(正しい言葉でいうとRMAP Destination)として、アクセスされる側にしかならないので、IPコアの違いは大きな問題にはなりません。基本的には、ボードのマニュアルに記載されているアドレスマップがわかれば、IPコアはなんでもよいはずです。また、SpaceCubeと違って、SpaceWireレイヤーからRMAPレイヤーまでの処理を全てハードウェアロジックだけで(ソフトウェアを介入させずに)行うので、SpaceCube上のRMAP Libraryと比べるとSpaceWire/RMAPによるデータ転送速度は圧倒的に高速である、ということを補足しておきます。以下のプログラムを使うには、**SpaceWire I/F board の logical address, destination key, CRC の方式を知っている必要があります。**あらかじめ調べておいて下さい。たとえば、Shimafuji DIO2 board では dip switch で logical address, destination key を変更できますが、のデフォルトでは logical address = 0x30, destination key = 0x02, CRC は draft F です。

SpaceWire IP core in SpaceWire FPGA on SpaceWire I/F board may be from various companies. However, since SpaceWire I/F board is always RMAP destination, the difference does not matter. As long as you know the address map, the program do not need to change. The logical address, destination key and CRC version must be known for programs shown below. The logical address and destination key of Shimafuji DIO2 boards can be set by dip switches. The default values are logical address = 0x30, destination key = 0x02 and CRC = draft F.

コマンドライン版rmaphongoを用いてボード上LEDを点灯させる / Light-on LEDs on the board using command-line rmaphongo.

シマフジ電機製の各種SpaceWire I/Fボードには、SpaceWire FPGAにも、UserFPGAにも、動作確認用のLEDが搭載されており、SpaceWire/RMAP経由でアクセスして、LEDに対応するアドレスのレジスタの値を変更することで、点灯・消灯

が行えるようになっています。他社製ボードでも、アドレスが違うだけで、同じような動きをするレジスタが搭載されているはずです。ここでは、SpaceCubeとI/Fボードを接続して、SpaceCubeのコマンドラインから、SpaceWire/RMAP経由でLEDを点滅させてみます。

Shimafuji SpaceWire I/F board or DIO1 board has LEDs for checking purpose for both SpaceWire FPGA and UserFPGA. They can be turned on/off by changing the register values of according addresses by SpaceWireRMAP. Note: Shimafuji DIO2 board does not have registers for LEDs. If you use Shimafuji DIO2 board, access to SDRAM for checking.

SpaceCubeとボードを接続して電源投入 / Power on SpaceCube and Board after connecting

SpaceCubeの3つあるSpaceWire I/Fのうち、普段はCN2番コネクタを利用します。ボードは、AD/DAボードの場合、J1コネクタを、DIOボードの場合はJ20コネクタを利用します。ボードとSpaceCubeをSpaceWireケーブルで接続したら、SpaceCube、ボードの電源を投入してください(電源投入の順番はどちらが先でも大丈夫です)。

Use CN2 connector among the three SpaceWire I/F of SpaceCube. Use J1 connector (AD/DA board) or J20 connector (DIO board) as the board side. After connecting SpaceCube and the board by a SpaceWire cable, turn-on power of SpaceCube and the board. The order of turn-on does not matter.

rmaphongoをSpaceCubeに転送・実行 / Transferring rmaphongo to SpaceCube and executing

FTPなどを用いて、前のセクションでビルドしたSpaceWire/RMAP Libraryに含まれるおまけソフト「rmaphongo」をSpaceCubeに転送してください。ビルド直後は名前がmain_rmaphongoとなっていて長いので、転送のときに「rmaphongo」に変更してしまいましょう。

Transfer to SpaceCube by FTP rmaphongo, which is included in the SpaceWire/RMAP Library as an appendix software, and is built in the previous section. Change the name to rmaphongo, because original name main_rmaphongo is rather long.

```
[/SYS]% fget (ホスト名):/usr/local/te/SpaceWireHongo/build/t-kernel/main_rmaphongo rmaphongo  
User: xxx  
Password: xxx
```

以下の例では、AD/DAボードのSpaceWire FPGAのLEDレジスタ(0x0100_0000番地)にアクセスして、LEDの点灯状況を変更しています。DIOボードの場合もアドレスは同じです。「Writing...」の表示ができるところで、LEDレジスタの値をWriteするためのRMAPコマンドパケットがプログラム内部で生成され、ボードに送られて、ボードの内部でLEDレジスタの状態が変わり、LEDが指定された状態で点灯(消灯)します。ボードからは、RMAP Writeアクセスが成功したのか失敗したのかを示す、RMAPリプライパケットが返送されてくるので、プログラムはそれを受け取って成功/失敗を判定し、以下の例では成功していたので「done」を表示しています。**なお、RMAP コマンドを送る前に、SpaceCube1 の IP core と port、ボードの IP core の設定を選択しています。これらを正しく入力しないと通信できません。**

In the following example, the LED status of an AD/DA board SpaceWire FPGA is changed by accessing a register (0x0100_0000). The address is same for DIO1 board. A RMAP command packet is created in the program during "Writing..." is displayed. Then it is sent to the board to change the LED status in the board and the LED is turned on or off as specified. The program judge whether it was successful by receiving an RMAP reply packet. In the following example, "done" is displayed to show it was successful. IP core and port of SpaceCube1 and your board should be appropriately input. Otherwise the communication does not work.

```

[ /SYS ]% rmaphongo <== 実行
Which SpaceWireIF implementation:
  Shimafuji IP Core      [2]
  Shimafuji Router IP Core [3]
select > 2                                     <== Choose 2 for SpaceCube1
Which port? [1,2,3] > 1                         <== Choose 1 for CN2 port
SpaceWireIFSpaceCube1ShimafujiIPcore : initialize
spw0 opened
RX TimeOut : 1000 ms
TX TimeOut : 1000 ms
fpga ver. = 0x1000
SpaceWireIFSpaceCube1ShimafujiIPcore : open (19)
connecting ...
link0 on (0x00000003f)
status = 0x00000003f
connected.
SpaceWireIFSpaceCube1ShimafujiIPcore : connect (0)
#####
### SpaceWire sample program : rmaphongo #####
###           ver 2.1 (20080624)               #####
###                                              #####
### This program is totally based on       #####
### SpaceWire/RMAP 'Hongo' Library.        #####
#####

Please set Destination information
Select :
  1 : Input new RMAP Destination
  2 : Reset to default RMAPDestination
  3 : Dump Current RMAP Destination
  4 : Quit RMAPDestination Setting Mode
### Frequently Used Settings ###
  10 : Old Shimafuji SpaceWire IF Boards
        (Logical Address=0x01, Draft E CRC)
  11 : Shimafuji SpaceWire IF Boards
        (Logical Address=0xFE, Draft E CRC)
  12 : Shimafuji SpaceWire IF Boards
        (Logical Address=0xFE, Draft F CRC)
  13 : Shimafuji SpaceCube Router Configuration Port
        (Logical Address=0xFE, Draft E CRC, Destination Key=0x02)
  14 : Shimafuji SpaceWire IF Boards (Shimafuji SpaceWire/RMAP IP)
        (Logical Address=0x30, Draft E CRC, Destination Key=0x02)
  15 : Shimafuji SpaceWire IF Boards (Shimafuji SpaceWire/RMAP IP)
        (Logical Address=0x30, Draft F CRC, Destination Key=0x02)
  20 : SpaceWireRouter by NEC
  30 : SpaceWire Universal IO Board by MHI
> 15                                              <== DIO-II board J20 defalut dip switch setting

RMAPDestination::dump() invoked
Destination
  Logical Address   : 30
  Path Address     : none
  Destination Key  : 02
  CRC Version      : Draft F
  Word Width       : 1bytes-1Word

Source
  Logical Address   : fe
  Path Address     : none
  Path Address Len. : 00

Menu :
  Read(2bytes)          [1]
  Write(2bytes)         [2]
  Read(general)         [3]
  Write(general)         [4]
  Receive a raw SpW packet [5]
  Send a raw SpW packet  [6]
  Set Destination Info   [8]
  Toggle Debug Mode [Off] [0]
  Quit                  [9]

select > 2                                     <== RMAP Write (2Bytes)を選択
Write (2bytes)
address in hex > 01000000 <== アドレス0x0100-0000を入力
data for 0000-0010 > 0002 <== Writeデータの入力
Writing...done                                     <== Writeの実行中 (LED表示がかわるはず)

Menu : (以下繰り返し)

```

ちゃんと動かない！そんなときは プログラム起動直後にエラーが表示される場合は、SpaceCubeで利用しているドライバが正しくロードされているか、ビルド時に用いたドライバラッパーのバージョンがIPコアと対応しているかなどを確認してください。Writing...のところで表示が止まってしまう場合も、ドライバやIPコアのところでバージョンの不整合が起きている可能性があります。Writing...の直後にエラーになる場合は、ボードの電源やSpaceWireケーブルの接続を確認してください。**また、定義されていないアドレスにアクセスを試みた場合も、FPGA がロックし、Writing... で表時が止まってしまいます。その場合、電源を切るまでその後のコマンドを受け付けなくなるので、アドレスの指定には気をつけて下さい。**

Not working! Then, If program returns an error right after starting the program, check that the driver is correctly loaded in the SpaceCube and that the driver wrapper version used during the build is same as that of IP core. If the display is stopped after "Writing...", inconsistency of version in the driver or IP core, is suspected. Check the power of board and connection of cable, if an error is returned after "Writing...". When you try to access to the addresses of which destination is not defined, FPGA could be locked. Then, the display stops at "Writing..." and the FPGA cannot accept any commands unless you turn-off the power.

補足：2008年7月現在のrmaphongoは、シマフジ電機製SpaceWire I/Fボード(NEC Soft IPコア版)を想定したアドレス指定(具体的には、ロジカルアドレスが32番以下になっている)になっているので、他社製ボードで利用する場合は、メニューの8番から、接続しているボードの情報(ロジカルアドレス、パスアドレス、Destination Keyなど)を設定しなおしてください。RMAP仕様では、ロジカルアドレスのデフォルトは0xFE(254)です。

コマンドライン版rmaphongoを用いてボード上のSDRAMにアクセス / Accessing SDRAM on a board using command-line rmaphongo

LEDレジスタと同様に、SDRAMのアドレスにアクセスすると、SpaceCubeからデータを書き込んだり、そのデータを読み出したりできます。AD/DAボードやDIOボードでは、0x0000_0000～0x00FF_FFFF番地までの16MBの空間がSDRAMに割り当てられているので、上の例を参考に、適当な場所にアクセスしてみてください(2008年7月現在のNEC Soft製IPコア搭載のSpaceWire I/Fボードでは、2バイト単位のアクセスが仮定されているので、偶数番地のアドレスにアクセスしてください)。実際の実験では、UserFPGAが取得した検出器データをSDRAMに書き込んで、それをSpaceCubeが読み出す、という形になります。

Data of the SDRAM can be written or read by SpaceCube, in the same way as we did for LED registers. The address of SDRAM is assigned from 0x0000_0000 to 0x00FF_FFFF for an AD/DA board, a DIO board and a DIO2 board. Use the example above to access the SDRAM. In the actual measurement, UserFPGA writes data from a detector on SDRAM, and SpaceCube reads it.

以上で、ボードを接続して、SpaceCubeとSpaceWire/RMAPで通信することができるようになりました。次のセクションでは、SpaceWire/RMAP Libraryを使って、独自のユーザプログラムを作成してみます。

Now the communication between a SpaceCube and a board using SpaceWire/RMAP is established.

SpaceWire I/Fを 利用するプログラムの開発 / Development of programs using SpaceWire I/F

全体像 / Overview

このセクションでは、次のことを行います。

- UserProgramのスケルトンを利用して新規プログラムを作成
- その中でLEDを指定回数点滅させてみる
- Condition.ccを用いて、適当な長さの待ち(wait)もかけてみる

The following items will be covered in this section.

- To create a new program from a skelton of UserProgram.
- To blink LEDs as many times as specified.
- To put a waiting time using Condion.cc

UserProgramスケルトンから新規プログラム作成 / Creating a new program from UserProgram skelton

新規ファイル作成 / creating new file → [Copy source_Executables/main_example_00.cc](#)

SpaceWire/RMAP Libraryには、**source_UserPrograms userprograms** というフォルダがあり、ユーザプログラムの開発はその中で行います。そのフォルダの中に入っているmain_UserProgram_Template.ccは、ユーザプログラムを作成するためのスケルトン(ひな形)です。source_Executables/main_example00.ccは、そのスケルトンを用いて作成した、SpaceWire/RMAP通信の使い方を示した簡単な例題です。ここでは、この例題00を少し書き換えて、一定時間ごとにボード上のLEDを点滅させるプログラムを作成してみます。例題00については、「SpaceWire/RMAP Library」のドキュメント[15]に、より詳しい解説が載っているので、そちらも参照してください。

Development of user programs should be done in a folder "userprograams" in the SpaceWireRMAP Library project. A skelton (template) of a user program is found as main_UserProgram_Template.cc. And an example of a user program is found as source_Executables/main_example00.cc. In this section, we modify this example00 and create a program to blink LEDs on the board for a certain time. More information about example00 can be found in a document of "SpaceWire/RMAP Library" [15].

新しいプログラムを作成するときは、**source_UserPrograms userprograms** フォルダのアイコン上で右クリックし「New」→「Source File」として、新規ファイルを作成します。実行形式を記述しているファイルであることがわかりや

すいように、main_XXXX.ccという名前をつけてください。新規ファイルが画面右のエディタ部分で開くので、source_Executables/main_example00.ccの中身をコピー&ペーストで流し込んでください。

When you create a new user program, click on the folder icon of "userprograms" and then select "New" -> "Source file". Then a new file is created. To clearly indicate it is supposed to be compiled to a executable, the file name should be main_XXXX.cc (XXXX should be changed as you like). Then a new file is opened in the editor window at the right side of the Eclipse window. Copy and paste the contents of source_Executables/main_example00.cc.

ユーザは、Mainクラスのrun()メソッドの中で、SpaceWire/RMAP通信や、データ処理、データ保存などの行いたい処理を記述します。このドキュメントでは説明しませんが、multi threadingをしたい場合も、child threadのクラスのインスタンス化とstart()を、ここで行ってください。また、オブジェクト指向の考え方沿って、機能やデータの表現をクラスとして分割し、見通しのよいプログラムになるよう、配慮して開発を行ってください。

A user will describe functions (SpaceWire/RMAP communication, data reduction, data writing, etc) in run() method in the Main class. If you use multi threading, instantiation and start() of child threads must be done in run() method. It is recommended to follow the object oriented programming so that functions and data are represented as classes to make the program clear to understand.

SpaceWire/RMAP通信を行う流れ / Overview of SpaceWire/RMAP communication

SpaceWire/RMAP Libraryを用いたSpaceWire/RMAP通信は、次のような流れで利用可能になります。検出器データの取得などの際は、初期化のときに1-4を実行し、データ取得中は5番を繰り返し行う形になります。

1. **SpaceWire I/Fクラスの初期化、オープン** 抽象化されたSpaceWire I/Fの実体(シマフジIPなのか、NEC Soft IPなのか)を指定してインスタンス化。ハードウェアに依存するのはこの一行だけ
2. **RMAPEngineクラスの初期化、スタート** RMAPの複数トランザクションを管理するためのレイヤをスタートさせ、RMAPパケットの送受信ができるようにする
3. **RMAPDestinationクラスを用いた接続先の指定** アクセスしたいRMAPターゲットが、何というロジカルアドレスをもち、どのようなパスアドレスで辿っていけばアクセスできるのか、CRCの計算方法のバージョンはDraft EなのかFなのかなど、Destinationの情報を保持
4. **RMAPSocketのオープン** 指定したRMAPDestinationをRMAPEngineに渡し、そのDestinationと仮想的につながったようなソケットを作成
5. **RMAPSocketインスタンスに対するread/write** 作成されたソケットに対して、read/writeを行うことで、内部でRMAPパケットの生成・送受信・解釈が自動的に行われる

SpaceWire/RMAP communication with SpaceWire/RMAP Library become available as follows. Items 1-4 are used as initialization and item 5 is repeatedly used for e.g., data acquisition.

1. **Initializing and opening SpaceWire I/F class** Creating an instance of SpaceWire I/F, by specifying which IP core is used (e.g., Shimafuji or NEC). This line is the only line which consider Hardware dependency.
2. **Initializing and starting RMAPEngine class** Starting a layer to manage multiple RMAP transactions and making sending/receiving of RMAP packet available.
3. **Defining the destination by RMAPDestination class** Input the logical address, path address routing, CRC calculation version (Draft E or F, etc) to define the information of the destination.
4. **Opening RMAPSocket** Pass the RMAPDestination to RMAPEngine to open a socket virtually connected to the destination.

5. **Reading/Writing to RMAPSocket** By reading/writing to the created socket, creation, sending/receiving and interpretation of the RMAP packet is automatically done.

SpaceWire I/Fは、SpaceWireIF.ccというabstract base classで、send()やreceive()など、SpaceWireレイヤーの操作に必要なメソッドのスケルトンを定義し、個々の実装はそれを継承したchild classの中で、パケット送受信などの具体的な処理を記述しています。たとえばSpaceCube1用NEC Soft製IPコアならSpaceWireIFSpaceCube1NECSoftIPcore.ccの中で、NEC Softさんから提供されているlibspw.a内の送受信関数をコールし、SpaceCube1用シマフジ製IPコアの場合はSpaceWireIFSpaceCube1ShimafujilPcore.ccの中で、spw_if.cで定義されている送受信関数をコールしています。

SpaceWire I/F is an abstract base class written in SpaceWireIF.cc. It defines skeletons of methods necessary for controlling SpaceWire layers. Actual coding of various functions like sending/receiving packets is described in child classes inheriting SpaceWireIF. For instance, SpaceWireIFSpaceCube1NECSoftIPcore.cc, which is for NEC Soft IP core of SpaceCube1, contains a function to call a sending/receiving function defined in libspw_a provided by NEC Soft, while for Shimafuji IP core, of SpaceCube1, SpaceWireIFSpaceCube1ShimafujilPcore.cc is used to call a sending/receiving function defined in spw_if.c.

使用するハードウェアが変わることは、このSpaceWireIFクラスのインスタンス化の一行を書き換えて再コンパイルすれば、新しいSpaceWireIFの実装に対応したプログラムに作り替えることができます。**2010 年 6 月現在、ハードウェアはインタラクティブに選択するようになっています。**

Once the program is compiled with modification of the line creating an instance of SpaceWireIF class for another hardware, the program can be used with the new SpaceWireIF for the hardware. As of June 2010, the hardware is selected interactively.

```

SpaceWireIF* spacewireif=
    SpaceWireIFIImplementations::selectInstanceFromCommandLineMenu();
// SpaceWireIF* spacewireif=new SpaceWireIFSpaceCube1NECSoftIPcore();

spacewireif->initialize();                                1. SpaceWireIFのインスタンス化
spacewireif->open();

//create an instance of RMAPEngine
RMAPEngine* rmapengine=new RMAPEngine(spacewireif);        2. RMAPEngineのインスタンス化
rmapengine->start();

//create an instance of RMAPDestination
//set properties for Shimafuji SpaceWire DIO Board
RMAPDestination rmapdestination;
rmapdestination.setDefaultDestination();
rmapdestination.setDestinationLogicalAddress(0xFE);
rmapdestination.setSourceLogicalAddress(0xFE);
rmapdestination.setDestinationKey(0x02);
rmapdestination.setDraftFCRC();
vector<unsigned char> path;
path.clear();
rmapdestination.setDestinationPathAddress(path);
rmapdestination.setSourcePathAddress(path);
rmapdestination.setWordWidth(1);                            3. RMAPDestinationの設定

//open RMAPSocket to "rmapdestination"                    4. RMAPSocketのオープン
RMAPSocket* rmapsocket=rmapengine->openRMAPSocketTo(rmapdestination);

//prepare data vector for write access
unsigned int writelength=16;
vector<unsigned char> writedata;
for(unsigned int i=0;i<writelength;i++){
    writedata.push_back((unsigned char)i);
}

//execute RMAP Write to Shimafuji DIO Board's SDRAM
unsigned int writeaddress=0x00000000; //SDRAM Address
try{
    cout << "Writing...";
    rmapsocket->write(writeaddress,&writedata);           5. RMAP Writeの実行
    cout << "Done" << endl;
} catch(RMAPEXception e){
    //if an exception occurs
    cout << "Exception in Write Access" << endl;
    //dump the exception
    e.dump();
}

```

使用するSpaceWire I/Fクラスを自分の環境に合わせる / Use an appropriate SpaceWire I/F instance for your environment

2行目にコメントアウトしているような、IP core のハードコードによる指定を行う場合は、必要に応じて適切なインスタンスを生成する必要があります。ここでは、NEC Soft製SpaceWire IPコアに対応したSpaceWireIFクラスを利用しています。シマフジ製IPコアを焼き込んだSpaceCubeを利用している場合は以下のように、インスタンス化の行を書き換える必要があります。その際、includeファイルも、SpaceWireIFSpaceCube1ShimafujiIPcore3.hhなど、対応したものを選択してください。

The IP core may be specified NOT interactively but by specifying it in the code. See the comment-outed line in the example above. In this example, a SpaceWire IP core by NEC Soft is assumed. If you use a SpaceCube1 with Shimafuji IP core, modify the line for instantiation as follows. The header file should also be the corresponding one like SpaceWireIFSpaceCube1ShimafujiIPcore3.hh.

```
SpaceWireIF* spacewireif=new SpaceWireIFSpaceCube1ShimafujiIPcore();
```

LEDを点滅させるように、処理を書き換える / Modify the file to make LEDs blinking

example00では、RMAP WriteとReadを一度ずつ、SpaceWire I/FボードのSDRAMに対して行って、書き込んだデータと読み出したデータが一致しているか(RAMに書いて、同じアドレスを読んでいるので、同じデータでないとまずい!)を調べています。今回は、アクセス先アドレスをボードのSpaceWire FPGAに接続されたLEDを制御するレジスタのアドレスに変更し、writeを複数回繰り返すようにすることで、LEDを点滅させるサンプルに書き換えてみてください。**注意: DIO2 board には LED 制御レジスタがありません。**

In example00, one RMAP Write and one Read was executed to the SDRAM on the SpaceWire I/F board. The written data and the read data should be same, because the same address is accessed. Change the accessed address to LED-controlling address connected to the SpaceWire FPGA and write several times to make LEDs blinking. LEDs of DIO2 board is not able to be controlled.

ヒント1 : SpaceWire AD/DAボードやDIOボードでは、LEDレジスタのアドレスは0x0100_0000です。

ヒント2 : 繰り返しwriteを行うときに、writeするデータ(vectorクラスで表現します)を変更することで、LEDのON/OFFを制御します。シマフジさん製のSpaceWire I/Fボードでは、LEDレジスタのビットが1のときにLEDはOFF、0のときにONになります。vectorインスタンスのデータを初期化するメソッドはclear()、追加するメソッドはpush_back()です。vectorインスタンスのn番目の要素を参照したいときは、at(n)メソッドを使います(Googleで「c++ vector」で検索すると、リファレンスはたくさん出てきます)。

ヒント3 : 単純なfor-loopでwriteを実行すると、ON/OFFの切り替わりが速すぎて、目で見えないかもしれません。でも、ビギループ busy loopは使ってはいけません。一定時間の「待ち」を入れたいときは、SpaceWire/RMAP Libraryに付属している、ThreadLibrary¹内のCondition.ccクラスを使います。詳しい方法は、「SpaceWire/RMAP Library」ドキュメント[15]を参照することになりますが、以下のように記述することで、ミリ秒単位の待ち時間をしています。busy loopではないので、このプログラム(スレッド)がwaitしている間も、他のプロセスやスレッドは処理を続行します(busy loopではCPUパワーがbusy loopの処理にとられるので、他のスレッドに割り当てられるCPU時間がwaitの場合に比べて大幅に減ります)。

Hint 1 : LED register address is 0x0100_0000 for SpaceWire AD/DA board and DIO1 board.

Hint 2 : For multiple writing, the data to write (described as a vector class) should be changed to control LED ON/OFF. LED is OFF when the LED register bit is 1, while it is ON when the bit is 0. To initialize the vector instance, use clean(), to add a data value, use push_back(), and to refer nth element, use at(n). By searching with Google, you may find more references.

Hint 3 : If you just use a for-loop, ON/OFF switching may be too fast for eyes to check. However, busy-loop should not be used. To wait a certain time, use Condition.cc class included in ThreadLibrary of SpaceWire/RMAP Library, as described below. See SpaceWire/RMAP Library document [15] for details. While the program (thread) is waiting, other processes or threads continue.

```
#include "Condition.hh" を冒頭に追加  
  
Condition condition;  
condition.wait(500); //wait for 500ms
```

¹ マルチスレッドや非同期動作を、複数のOS環境でサポートするクラスライブラリ。JavaのThreadと、POSIXのpthreadを参考にして作成した。T-Kernelの、可読性の低いシステムコールをユーザプログラムであらわに記述しなくてよくなる。

答え合わせ / Answer

湯浅が作成したサンプルをhttp://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/spc/20080717/mainTutorial_example00.cc に置いておいたので、参考にしてみてください。

See an answer at http://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/spc/20080717/mainTutorial_example00.cc.

実際の検出器読み出しプログラムの作成では

検出器読み出しプログラムは、検出器パラメタの設定、測定開始、測定データ読み出し、データ処理・保存、測定終了という一連の流れがだいたい決まっています。まだSpaceWire/RMAP Libraryの中では、読み出しプログラムのスケルトンは提供できません。できれば、これに関しても、移植性の高い、コンパクトな、可読性の高いスケルトンを作成し、ドキュメンテーションを整えたいと思っていますが、human resourceの問題で未了です。

どなたか、そういったフレームワークを作成されたら、フィードバックいただけませんでしょうか。SpaceWireRMAPLibraryフォルダの中に、source_DaqLibraryみたいなフォルダを作って、ソースを管理してもらえると、たくさんの人々に役立つ、大きな貢献になります。

参考文献(というかソフトウェア)としては、NOVA、Unidaq、MIDAS、Kinoko、ANL+、RTMiddlewareなどが挙げられると思います。aceやboostなどportableなライブラリ、Javaのような可読性の高いソースコードも、とても参考になるはずです。

UserFPGAの作成

全体像

このセクションでは、次のことを行います。

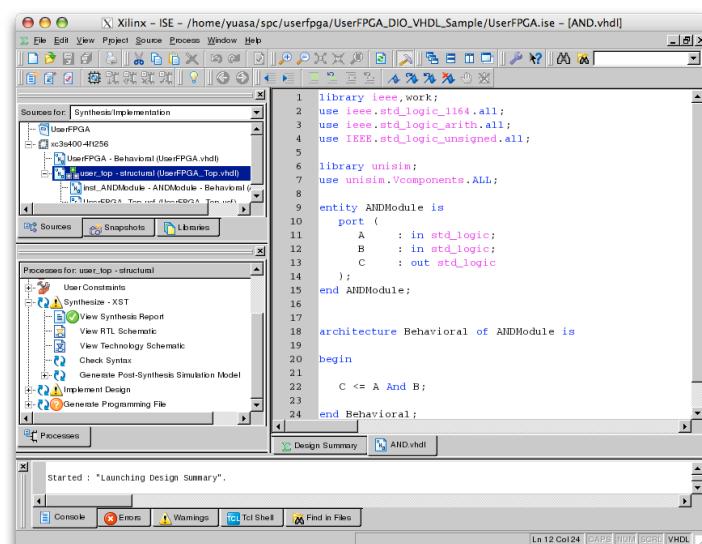
- FPGA開発ツール「Xilinx ISE」のインストール
- UserFPGA_templateのダウンロード
- UserFPGA_templateのサンプルをコンパイル
- UserFPGA_templateのサンプルをUserFPGAに書き込んで動作させる
- サンプルの改変(レジスタ追加、ステートマシン作成など)

開発環境の整備

シマフジ電機製SpaceWire I/Fボードには、主にXilinx製のFPGAが搭載されています。FPGAは製造メーカーごとに独自の開発ツールが存在しており、FPGAメーカーの選定を変更すると、ツールの使い方をまた一から覚えなければいけないというのが現状です。このドキュメントでは、Xilinxの開発環境を例に話を進めます。

XilinxのFPGA開発統合環境はISE(Integrated Software Environment)と呼ばれており、WebPackという名前の無料版がwebサイト(http://japan.xilinx.com/ise/logic_design_prod/webpack.htm)からダウンロードできます。かなりサイズが大きいですが、ダウンロードしてインストールしてください。半年に一度くらいの頻度で大きなアップデートがあり、そのたびに機能が追加され、処理性能が体感できるほど向上(コンパイルにかかる時間がすごく短縮される、とか)するので、いつも最新版を使うようにしましょう。

Xilinx ISEは2008年7月現在、WindowsとLinuxにしか対応していません。Macで利用する場合は、VMWare FUSIONやParallelsなどの仮想化環境を利用することになります(仮想化環境でも、十分な速度で動作します(実際、湯浅はVMWare FUSION上のWindowsで開発を行っています))。また、研究室にワークステーションのLinuxマシンがある場合は、そちらにインストールしてX環境でリモートログインして利用する、という方法もあります。ビルド時間はCPUパワーにほとんど反比例するので、この方法もおすすめです。



また、開発したFPGA用ロジックを、実際にFPGA(や、それに付随するEEPROM)に書き込むためには「ダウンロードケーブル」という名前のUSB機器が必要になります。Xilinx製品を取り扱っている代理店から購入できます(～15000円くらい)。本郷では東京エレクトロン デバイス株式会社を利用しました。ISEをインストールするとき、ダウンロードケーブルのデバイスドライバも一緒にインストールするか聞かれるので、インストールしておいてください。

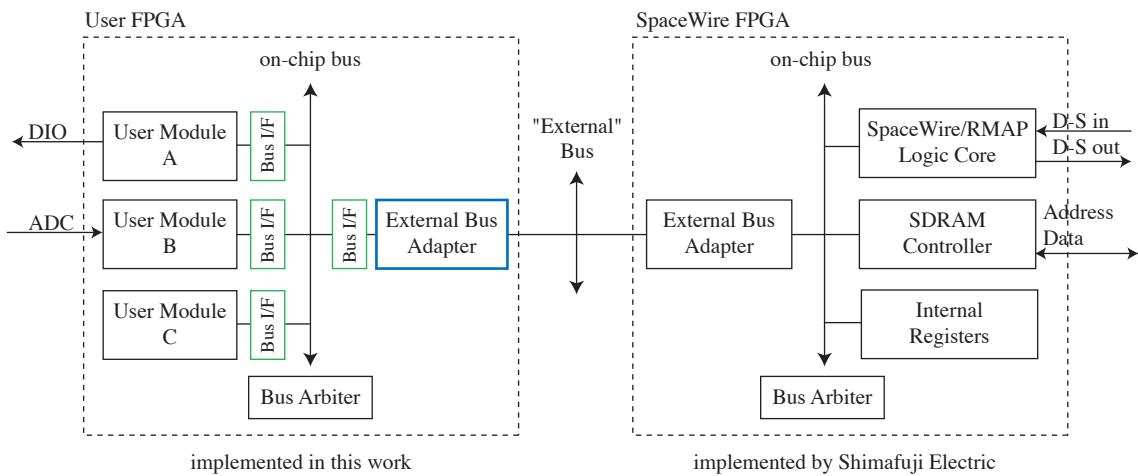
UserFPGA_templateについて

UserFPGAの典型的な内部構造を以下に示します。検出器からの信号を処理したり、検出器を制御したり、取得データを処理するための「ユーザモジュール UserModule」は、各実験で望みの動作が異なるので、ユーザごとに実装する必要があります。

一方で、SpaceWire FPGA側から「External Bus」経由で伝えられるRMAPコマンド(Write/Read)を処理するための「バスアダプタ」や、ユーザモジュール間でデータを流すon-chip busの仕組み(External Busに対してInternal Busと呼ばれています)、SDRAM管理モジュール(SDRAM Controller)などは、どの実験でも共通に必要になるもので、個々のユーザが記述する必要はなさそうです。

また、個々のユーザモジュールもRMAP経由でパラメタ(たとえばtrigger thresholdや、サンプル点数)を変更するなど、SpaceWire/RMAPで到達できるようになっていたほうが便利ですが、そのような機能も、ユーザごとに車輪の再発明をするのは非効率的です。

そういうわけで、共通に必要なモジュール群と、RMAPでアクセス可能なユーザモジュールのひな形(スケルトン)を提供するために、UserFPGA_template (Odaka et al[9], Yuasa et al.[10])というVHDLで記述されたモジュールのパッケージが用意されています。以下では、UserFPGA_templateを利用して、SpaceWire AD/DAボードのUserFPGAを開発する例を紹介します。



Yuasa 2008[14]より引用

VHDLに関する簡単な解説

VHDLを用いたFPGA用のロジック開発は、大きく分けて「抽象的な信号線名を用いたロジックモジュールの記述」、「モジュールをインスタンス化して、FPGA全体の構造を記述するトップファイルの作成」、「シミュレーション用のテストベンチ記述」、「シミュレーションによる動作試験」という流れをたどります。シミュレーションで動作が確認できたら、「個々のFPGAに依存する、ピン番号と信号線名の対応付け」用ファイルの作成と「論理合成・配置配線・EEPROM書き込み用ビットストリームファイル作成」という一連のビルト作業を行ってから、「完成したロジックをFPGA(もしくはEEPROM)に書き込んで試験・使用」となります。

各VHDLの機能モジュールに対応するファイルは、大きく分けて2つの部分から構成されます。

1. **entity declaration part** そのモジュールのI/F信号の名前と方向(そのモジュールからみたin/out)の定義。C/C++のプロトタイプ宣言のようなもの
2. **behavioral part** モジュールの実装を記述する部分。内部でさらに「実装の記述で使用する信号線や、利用するモジュールの宣言」と「実装の記述」のふたつのパートに分かれています

VHDLモジュールの例

```
library ieee,work;
```

```
use ieee.std_logic_1164.all;
```

```
entity SampleModule is
  port(
    inputA : in std_logic;
    inputB : in std_logic;
    output : out std_logic;
    clock : in std_logic;
    globalreset : in std_logic;
  );
end SampleModule;
```

1. entityの宣言部分

```
architecture behavioral of SampleModule is
```

```
  component SubModule
```

```
    port(
      subin : in std_logic;
      subout : out std_logic;
    );
  end component;
```

```
  signal counter : integer := 0;
  signal internalsignal : std_logic;
```

2. behaviorの記述

使用する信号線やモジュールの宣言

```
begin
```

実装の記述

```
  instanceOfSubModule : SubModule
    port map(
      subin <= inputA,
      subout <= internalsignal;
    );
```

```
  process(clock,globalreset)
  begin
    if(globalreset='0')then
      counter <= 0;
      output <= 0;
    elsif(clock'event and clock='1')then
      if(inputA='1' and inputB='1')then
        if(counter<1000)then
          counter <= counter + 1;
        else
          counter <= 0;
          output <= not output;
        end if;
      end if;
    end if;
  end process;
end behavioral;
```

パーソナルコンピュータに搭載されているCPUは、メモリに書かれたインストラクションを先頭から順番に解釈しながら処理を進めます。一方VHDLでは、ある信号の値に応じて、出力信号がどうあるべきかを記述することで、処理を進めます。「組み合わせ回路」と「クロック同期回路」の2種類があり、前者の一番簡単な例のひとつは、2つの入力信号の論理和(AND)をとる回路で、

```
--信号線定義部
signal ina : std_logic;
signal inb : std_logic;
signal out : std_logic;

--動作記述部
out <= ina and inb;
```

のように記述できます。内部的にも、FPGAのlook up tableという要素を用いて、入力信号に対応する出力信号の組み合わせをマトリックスとして表現します。out信号が変化するタイミングは、inaやinbが変化するタイミングであり、外部のクロックとは関係なく(非同期で)動作することになります。

後者では、「ある信号」の変化に同期して、出力信号を決定します。内部的には、「ある信号」(多くの場合はクロック信号)で駆動されたflip flopの組み合わせで構成されます。よく使われるクロック同期回路の例は、「ステートマシン」で、組み合わせ回路よりも複雑な、分岐や繰り返しを伴う処理を、可読性を保ったままで記述できます。ステートマシンは、複数の状態(ステート)と、その状態の間の遷移条件で記述される計算モデルで、遷移条件を適当に設定することで、任意の条件分岐や繰り返しを実装できます。ステートマシンはシーケンサとも呼ばれます。

以下に、クロック信号(Clock)が $0 \Rightarrow 1$ と変化するたびに、入力信号の状態に応じて1ステップずつ処理を進めていくステートマシンの例を示します。

```
--信号線定義部
type StateMachine_State is
    (Initialize, Idle, Wait500ms, State2, Finalize);
signal state : iBus_Receive_StateMachine_State := Initialize;

--動作記述部
SampleStateMachine : process (Clock,GlobalReset)
begin
    --is this process invoked with GlobalReset?
    if (GlobalReset='0') then
        --Initialize StateMachine's state
        state <= Initialize;
    --is this process invoked with Clock Event?
    elsif (Clock'Event and Clock='1') then
        case state is
            when Initialize =>
                --turn off LED
                LEDRegister <= x"0000";
                UserModule2BusIF.ReceiveEnable <= '0';
                --move to next state
                state <= idle;
            when Idle =>
                if (LVDSIn='1') then
                    --turn on LED
                    LEDRegister <= x"ffff";
                    --initialize counter
                    counter <= 0;
                    --move to next state
                    state <= Wait500ms;
                end if;
            when Wait500ms =>
                --wait
                if (counter>10000000) then
```

```

        --move to next state
        state  <= Finalize;
    else
        counter <= counter + 1;
    end if;
when Finalize =>
    --turn off LED
    LEDRegister <= x"0000";
    --move to next state
    state  <= Initialize;
when others =>
    --move to next state
    state  <= Initialize;
end case;
end if;
end process;

```

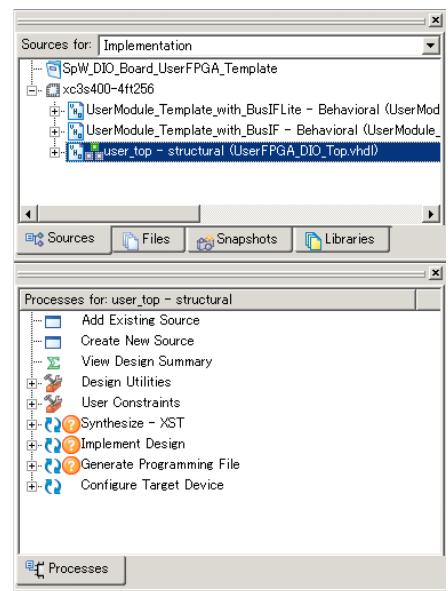
もっと詳しいVHDL

VHDLや、ハードウェア記述言語全般に関するもっと詳しい情報は、インターネットや書籍[e,f]で得てください。相当奥が深いので、勉強してみて、よりよいドキュメントが作成できたら、こちらにもフィードバックをお願いします。

UserFPGA_templateのインポートとビルト

UserFPGA_templateには、SpaceWire AD/DAボードと、SpaceWire DIOボード用のISEプロジェクトファイルが含まれています。これらの中で異なる部分は、トップファイルとUCF(User Constraint File; ピンと信号線の対応、タイミング制約などが記述されたファイル)だけです。内部で使用している機能モジュールは同じものです。UserFPGA_templateのアーカイブをSpaceWire Wikiなどからダウンロードし、展開してください。ISEは、ファイルパスに英数字以外の文字が含まれていると、正しくファイルを読み込めないので、Windows環境で作業する場合は「デスクトップ」に配置せず、「C:\UserFPGA」などのフォルダを作成して、その中に配置してください。展開されたフォルダの中に入っている.ise形式のファイルがISEのプロジェクトファイルです。使用するボードにあったものを選択してダブルクリックしてください。ISEが開いて、自動的にプロジェクトに関連したファイルが読み込まれるはずです(ISEのバージョンが新しくなった場合には、「プロジェクトファイルをアップデートするか」旨のダイアログが出るので、「はい」をクリックしてアップデートを行ってください)。

とりあえず、UserFPGA_templateのサンプルをそのままコンパイルして、動作を確認してみましょう。プロジェクトをビルトするには、画面左上のファイル一覧で「userfpga_dio_top - structural (UserFPGA_XXX_Top.vhd)」をクリックして選択状態にしておき、画面左下のProcessesタブ内の「Generate Programming File」をダブルクリックしてください。「Synthesize(論理合成)」、「Implementat Design(実装; Translate, Map, Place and Route(配置配線)」が自動的に行われていきます。論理合成や、合成した論理回路を実際のFPGAの組み合わせ回路やflip flopの配線に落とし込んでいく計算は、かなり複雑なので、使用するCPUによっては数分から10分程度時間がかかるかもしれません。また、ビルト中には、山のようにWarningが表示されますが、とりあえずは気にしなくて大丈夫です(VHDLやISEに詳しくなった人は、どうすればWarningが出なくなるか教えてください)。不幸にしてビルト途中でエラーが出る(Processの画面で、赤色の×印が出てしまう)場合は、近くのスタッフか研究室のシニアなメンバー、もしくはSpaceWire MLのメンバーに助けを求めてください。ISEで出てくるエラーには、いろんな原因があり得て、環境(ISEのバージョン、そのほかいろいろな状況)によって個別に対処

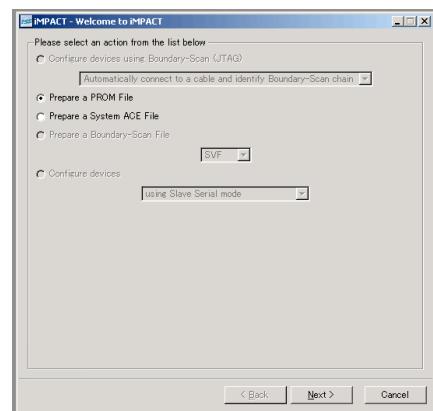


法が異なるので、ここでは詳しく説明できません(すいません)。

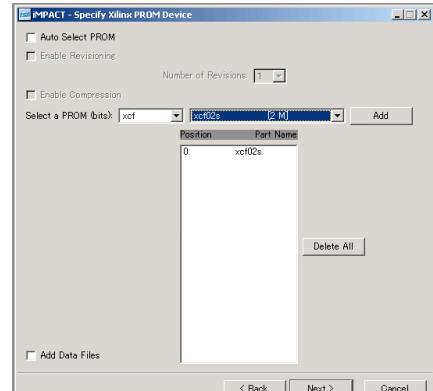
ビルドに成功すると、FPGAに直接書き込むためのビットストリームファイルが生成されています(UserFPGA_Top_XX.bit)。FPGAの内部配線はSRAMで作られていて、一度コンフィグレーションしても(bitファイルを書き込んでも)、電源を切るとその内容は消えてしまいます。電源投入時には、毎回、外付けのEEPROM(基板上でFPGAの近くに配置されているLSI)から、コンフィグレーション用データを読み込んで、SRAMによる内部配線を接続していきます。そのEEPROMに書き込むためには、生成されたbitファイルを、ISEの一部である「iMPACT」という書き込み/変換ツールを使ってEEPROMのデータ形式に変換します。

(以下、ISE10.1をもとに話を進めますが、バージョンによっては表記が変わっているかもしれません)ビルドの際と同様に、「Processes」タブから「Configure Target Device」ツリーを開いた中にある「Generate Taret PROM/ACE File」という項目をダブルクリックします。「iMPACT(FPGA書き込みツール)用のファイルがないので生成してよいか」という旨のダイアログが表示されるのでOKをクリックします。つづいて、bitファイルから変換して生成するファイルの形式を聞かれるので、「Prepare a PROM File」を選択して「Next」をクリックします。

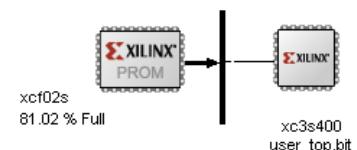
続いて、生成するPROM用ファイルの保存先とファイル名を聞かれるので、「Untitled」になっているファイル名を、「UserFPGA_DIO.mcs」や「UserFPGA_ADDA.mcs」などわかりやすいものに変更します。PROM File Formatは、「MCS」をチェックしておいてください。



次の画面では、FPGAとEEPROMの接続モードを聞かれますが、「I am using a Xilinx PROM in Serial Mode」を選択して、「Next」をクリックします。



さらに、次の画面では使用しているEEPROMの型番を選択するよう求められるので、右の画面のように、「Select a PROM (bits)」の項目のところを「xcf」 「xcf02s (2M)」にした状態で「Add」をクリックして、画面中央のリストの部分にxcf02sが追加された状態にします。この型番は、ボードで使用しているEEPROMの種類によって異なるので、DIOボードとAD/DAボード以外の場合は、確認が必要です。たとえばClear Pulse製SpaceWire ADC Boxでは、UserFPGAにSpartan-3のXC3S1500(DIO・AD/DAボードではXC3S400)を使っているため、EEPROMも容量の大きいxcf04sになっています。次の画面は設定の確認だけなので、そのまま「Finish」をクリックしてください。



画面が切り替わって、「xcf02s 0%」というメッセージと「Add Device」这样一个ダイアログが表示されます。「OK」をクリックして表示されるファイル選択ダイアログから、変換元のbitファイル(userfpga_xxx_top.bit)を選択します。さらに「Would you like to add another device file to Data Stream : 0」と聞かれますが、いまは一つのbitファイルしか変換しないので、「No」をクリックしてください。右のような画面になったら、左下の「Configuration Operations」タブから「Generate File...」をダブルクリックすると、MCSファイルが生成され、成功すると青い囲みで「PROM File Generation Succeeded」と表示されます。

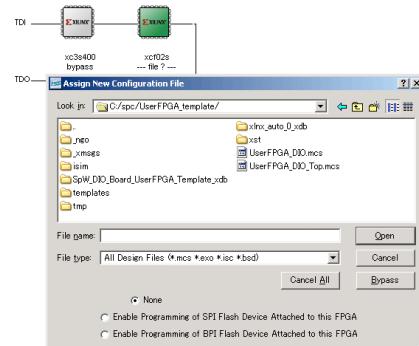
UserFPGA_templateの書き込み

以上でボード上のUserFPGA(のEEPROM)に、UserFPGA_templateのロジックを書き込む準備ができました。実際に書き込むには、「USBダウンロードケーブル」を開発PC/Macに接続し、ダウンローダー本体から伸びているJTAGケーブルを、ボード上のUserFPGA用のJTAGコネクタに接続します。SpaceWire FPGAとUserFPGAが搭載されているボード(DIOやAD/DA、ADC Box)では、SpaceWire FPGA用のJTAGコネクタも近くに配置されているので、間違えないようにしてください(実際は、接続を間違えてたとしても、EEPROMの形式が違うので書き込みのときにエラーになります)。UserFPGA用のJTAGコネクタは、DIOボードならJ19、AD/DAボードならJ9です。

PC/Mac側で、ダウンロードケーブルが認識されてUSBから給電が行われていると、ランプがオレンジに点灯します。ボードの電源が入っていないとEEPROMに書き込めない(JTAGチェーン[F]に接続できない)ので、ボードの電源を投入すると、ランプは緑に代わり、書き込みが行えるようになります。

iMPACTの画面の左上、「Configuration」タブから、「Boundary Scan」をダブルクリックして、JTAGチェーンにどういうLSIが接続されているかを表示する画面に移行します。当初は何も表示されないと思うので、「Operations」メニューの「Initialize Chain」を選択するか「Ctrl+I」と入力して、チェーンを自動的に識別させます(USBダウンロードケーブルのドラバがロードされていなかったり、ボードの電源が入っていないとエラーになります)。

JTAGチェーンの識別が終わると、ファイル選択ダイアログが2回開いて、FPGAとEEPROMのそれぞれに書き込むファイルを選択するよう促されます。デバッグの段階で、EEPROMに書き込みますに、試験的にFPGAに書き込みたい場合は、一つ目のファイル選択ダイアログでbitファイルを選択してください。EEPROMに直接書き込んで、電源を切っても内容が保持されるようにしたい場合は、1つ目のダイアログは「Bypass」ボタンをクリックして、何も選択せず、2つ目のダイアログで先ほど作成したmcsファイルを選択して「OK」をクリックします。

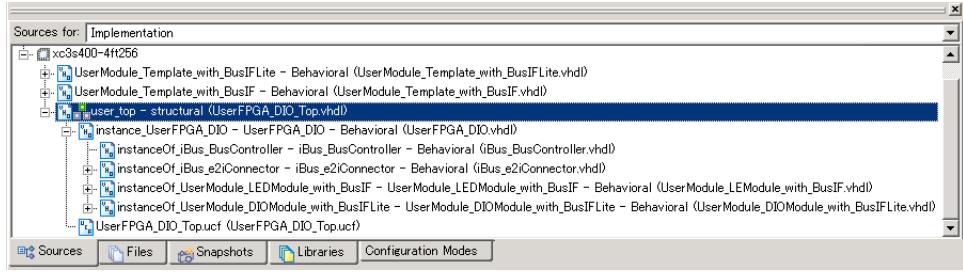


「Device Programming Properties」というウインドウが開きますが、デフォルトの設定で大丈夫なので、そのまま「OK」をクリックしてください。JTAGチェーンのアイコンの画面で、右の図のように、EEPROM(図ではxcf02s)をクリックして選択状態にしてから、左下の「Configuration Operations」ダイアログで「Program」を選択します。再度「Device Programming Properties」というウインドウが開きますが、これもデフォルトのまま大丈夫なので「OK」をクリックすると書き込みが始まり、しばらくすると「Succeeded」という表示がでれば書き込み完了です。

FPGAに書き込んだ場合は、その時点で書き込んだ内容のロジックが動作し始めるはずです。EEPROMに書き込んだ場合は、ボードの電源を落として、再投入すると、新しいロジックがFPGAにコンフィグレートされて動作し始めます。

UserFPGA_templateの内容説明と動作確認

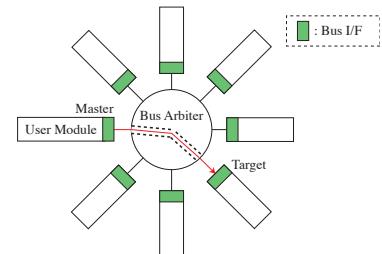
話が前後しますが、動作確認の前に、UserFPGA_templateに実装されているモジュールの説明をします。実装モジュールの一覧は、「Source」タブの中でトップファイル(userfpga_xxx_top)ツリーを展開することで確認できます。以下、それぞれのモジュールの機能を簡単に説明しておきます。



UserFPGA_templateに含まれているモジュール

- **iBus_BusController** internalBus(内部データバス)のコントローラ。各内部バスモジュール(UserModule含む)は、このモジュールに接続され、このモジュールがスイッチのように働いてモジュール間のデータ転送を実現する。調停役という意味で、アービタ(arbiter)とも呼ばれる
- **iBus_e2iConnector** internalBusとexternalBus(外部バス; SpaceWire FPGAとUserFPGAの間のデータバス)を接続するためのモジュール。SpaceWire FPGA側からやってきたRead/Writeアクセスを、internalBusへのアクセスに変換する。たとえばReadの場合は、internalBus経由で目的のUserModuleからデータを読み出し、その結果をexternalBusに返す。SpaceWire FPGA側は、返されたデータをもとに、RMAP Reply Packetを生成して、SpaceCubeなどに返す
- **UserModule_LEDModule_with_BusIF** UserModuleのサンプルの一つ。LEDを管理するレジスタを内部にもつておらず、RMAP/internalBus経由でその値を書き換えることで、LEDのOn/Offを切り替えることができる
- **Usermodule_DIOModule_with_BusIFLite** UserModuleのサンプルの一つ。Digital In/Outを管理するレジスタを内部に持っております。RMAP/internalBus経由でその値を書き換えたり、読み出したりすることで、ボードのDIOのI/Fを用いてデジタル信号を入出力できる

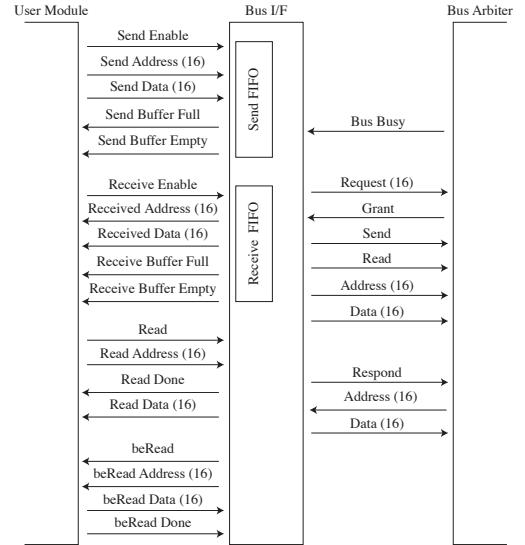
各UserModuleは、「UserFPGA_templateについて」の項の図中や、右図の緑の部分で示されるように、「BusIF」というモジュールを介してinternalBus Controllerに接続されています。これにより、internalBusの実装は隠されており(カプセル化 encapsulation されている、という)、例えば使用するFPGAが変わるとか、使用するinternalBusの実装が変わるとかの場合でも、個々のUserModuleを変更する必要はなく、BusIFモジュールの内部実装を(UserModule側のI/F仕様はそのまま)変更するだけで対応できるようになっています。また、BusIFモジュールの実装は、機能(使いやすさ)と使用するロジック数の違いによって、2つの種類が提供されています。



- **iBus_BusIF** 内部にFIFOを持っており、他のUserModule宛のデータ送出、他のUserModuleからのデータ受信を自動的にバッファリングする機能をもつBusIFモジュール。これにより、たとえば、このBusIFを使用するUserModule側がデータを送りたいときは、BusIFの送信FIFO(次ページの図)にデータを放り込んでおけば、internalBusの空き状況をBusIFが適当に判断して、自動的にデータを送出してくれる。受信のときも、UserModuleは特に何もする必要はなく、受信FIFOに何かデータが届いているかチェックするだけでよくなっています(詳細はソースコード内のドキュメンテーションを参考にしてください)。ただし、FIFOを2個使用するので、その分FPGAのリソースを消費します(ディテールですが、現状では、RAM専用の「BlockRAM」リソースを用いてFIFOを作っているので、このBusIFを使ったとしても、ロジックの実装に使用可能なスライス slice の数は減りません)

- **iBus_BusIFLite** よりシンプルなBusIF実装。FIFOを内蔵しないので、データ(16bits単位)の送信、受信ごとに、UserModule側での処理が必須になりますが、使用するロジックリソースは少なくなります。UserModule_DIOModuleのように、簡単なレジスタアクセスしかしないモジュールには、このBusIFLiteで十分でしょう。いっぽう、検出器データを取得して、そのデータをSDRAMに送りたい場合などは、FIFO付きBusIFのほうがすっきりと記述できると思います

internalBusでは、各BusIFモジュールのインスタンス化 instantiation の際に、管理を担当するアドレス空間を割り当てます。普通、BusIFのインスタンス化はUserModuleの中で行うので、指定するアドレス空間は、UserModule自身のinternalBus内におけるアドレス空間となります。現状、internalBusの実装はアドレス/データとも16bits幅を仮定しているので、カスケード接続などをしないシンプルなinternalBusの使用状態では、16bits(アドレス)=65kB分のアドレス空間が割り当てられていることになります。シマフジ電機製SpaceWire FPGA側では、0x0101_0000～0x0101_FFFFまでの16ビット幅の空間をUserFPGAへのアクセスだという風にマップしているので、たとえばSpaceCubeからSpaceWire DIOボードの0x0101_0aaa番地にアクセスすると、UserFPGAのinternalBusの0x0aaa番地を管理しているUserModuleにアクセスが到達することになります。



各UserModuleのアドレス空間の割り振り address mappingは、VHDLソースに直接、値を書き込んで hard codedしてしまうと、変更が面倒だけでなく、全体のアドレスマップを把握しづらくなってしまいます。それをさけるために、UserFPGA_templateでは、internalBus関連のアドレス情報(UserModuleのアドレスマップと、UserModule内の各レジスタのアドレス)をiBus_AddressMap.vhdというファイルの中で一元的に管理します。新しいUserModuleを追加する際は、そのモジュールのアドレスをこのファイルに記述してください。

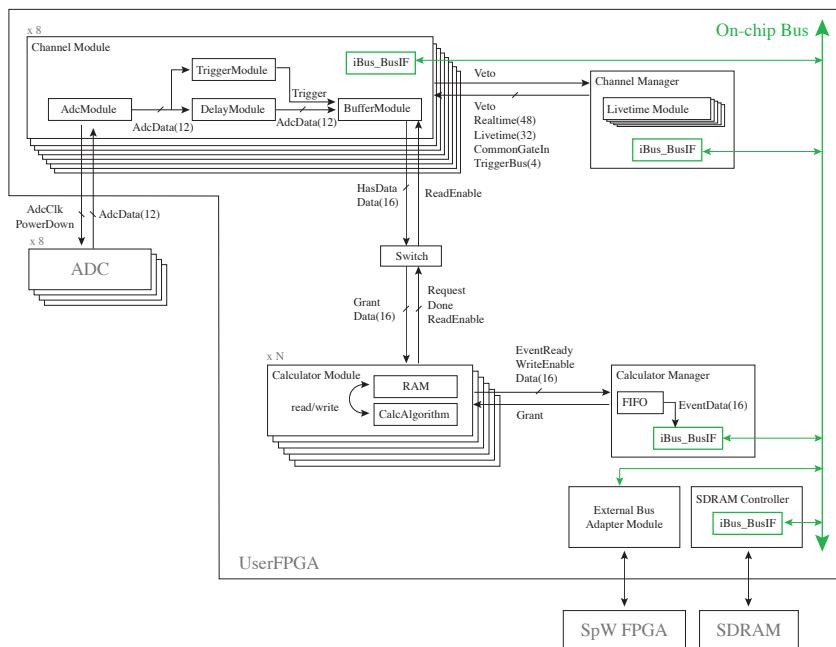
それでは、では先ほど書き込んだFPGAの動作チェックをしてみましょう。

「SpaceWire I/Fボードを接続」のセクションで行ったように、SpaceCubeとボードをSpaceWireケーブルで接続して、電源を投入してください。FPGAを書き換えたときは、消費電流の変化にも注意しましょう。ロジックをたくさん使っているときは、消費電流(電力)も増大します。典型的にはAD/DAボードやDIOボード1枚で200～300mA程度の電流が流れます。500mA以上流れている場合は、ショートが起きている可能性があります。すぐに電源を切って、間違っている箇所を特定し、PROMを書き直してください。注意：UserFPGA_templateをそのまま使っているときはショートは起きないはずです。ロジックの改変を行う際は、書き込む前に必ず、シミュレーションでの動作チェックを行い、さらに、EEPROMに書き込む前にFPGAにオンザフライで焼いてみて、ショートが起きてても電源を切ればもとの安全なロジックに戻れるようにしておきましょう。EEPROMに書き込むのは、ロジックが正しく動作し、ボードを故障させることができないとわかつてからにしましょう。

デフォルトでは、LEDModuleのLED Registerは0x0101_f020番地に、DIOModuleのDigital Output Registerは0x0101_f030番地に、Digital Input Registerは0x0101_f032番地に割り当てられているので、コマンドライン版rmaphongoなどから、これらのレジスタにアクセスして値を読み書きすると、LEDのOn/OffやDIOポートを用いた信号入出力が行えます。

UserFPGA_templateから先へ進むには

検出器データを読み出す際のUserFPGAの構成としては、例えば、検出器のパラメタ(HV、thresholdなど)をDIO経由で制御するためのUserModuleや、トリガを生成するUserModule、検出器データを受信してイベントパケットを生成するUserModule、イベントを判定してデータリダクションを行うUserModule、SDRAMにイベントパケットを書き込むモジュールなどが必要になると思われます。そういういたモジュールの開発は、UserFPGA_template内のUserModule_Template_with_BusIF.vhdやUserModule_Template_with_BusIFLite.vhdをコピーして行うと簡単かもしれません。以下に、8chの高速ADCを搭載した、SpaceWire ADC BoxのUserFPGAを開発したときのブロックダイアグラムを掲載しておきます。何かの参考にしてください(一つ一つの四角い箱が、VHDLのモジュールを表します。もし生のソースコードが必要な場合は、湯浅まで連絡してください)。



SDRAMの利用について

SpaceWire DIOボードやAD/DAボードのUserFPGAにはSDRAMが接続されておらず、SpaceWire FPGAに接続されています。この状態では、UserFPGAで取得したデータをSDRAMにバッファリングする際に、externalBusとSpaceWire FPGAを経由しなければならず、ちょっと難しい設計になっています。Clear Pulseおよびシマフジ電機と協力して製作したSpaceWire ADC Boxでは、データバッファリング用のSDRAMはUserFPGAに直接接続されています。

SDRAMは、SRAMに比べて大容量なものが安価に入手できることが特長ですが、通電状態でもある程度時間がたつと記録していたデータが消えてしまうという特性をもつので、一定時間ごとに「リフレッシュ」という作業を行う必要があります。また、データの読み書きに際し、メモリ構造の行と列を指定しないと行けない(SRAMのように、アドレスとデータを入力してWrite Enableを立てればよい、というレベルではない)など、使い方もSRAMに比べると少々難しくなっています。それらの面倒な作業をモジュール化して、ユーザロジック側からはSRAMなどと変わらないように見せるためのモジュールがSDRAMコントローラ(SDRAMC)です。SpaceWire I/Fボードに搭載されているSDRAMを使うときには、このモジュールが必要になります。

SDRAMの種類によっては、ISEの付録IPコア集の中に集録されているものが利用できるかもしれませんし、ボードを製作してもらったメーカーさんからサンプルが提供していただけるかもしれません。いずれにせよ、ボードごとに異なる対応が必要なので、困ったときはSpaceWire MLに相談してみてください。

また、自分でフリーのSDRAMCを作ったという、ひじょうにすばらしい人がおられましたら、ぜひソースコードをオープンにして、SpaceWire以外の人たちにも利用可能にしてあげてください。たくさんの人から感謝されること間違いないなしです(とくに、そのような、たくさんの人向けのオープンなIPコアを開発する場として、opencores.orgというものがあります。彼らが標準データバスとして利用しているWishboneを搭載しちゃうと、世界的にもほめてもらえます)。

UserFPGA_templateをもちいた例題(VHDLの勉強用)

以下のような例題は、UserFPGAを開発するときのよい練習台になるかもしれません。UserFPGAの開発で、ちょっと自分で考えたり、調べたりしても解決できないような場合は、同じ問題にあんまりハマっていてもしかたがないので、SpaceWire MLなどに連絡してみましょう。だれかが答えを知っているかもしれませんし、そうでなくとも一緒に考えてくれるはずです。

- internalBus経由で、1秒ごとに、LEDModuleへ向けてLED Registerの値を反転させるようなデータ通信を行うUserModule
- AD/DAボードのDACから、周期や振幅がRMAP経由で可変のsin波や矩形波を出力するUserModule
- Digital Inputピンをトリガー信号入力として用いて、SpaceWire ADC BoxでPMTやAPDなどの波形をデジタル化し、FPGAの内蔵RAM(BlockRAM)に保持し、RMAP経由で読み出せるようなUserModule

SpaceWire/RMAP Libraryの応用 / Further usage of SpaceWire/RMAP Library

このセクションについて / About this section

このセクションでは、SpaceWire/RMAP Libraryを用いて、どんな応用的なことができるかを、簡単に説明します。より詳細な使い方、実装のコンセプトなどは、「SpaceWire/RMAP Library」のドキュメント[15]を参照してください。

Examples of further usage of SpaceWire/RMAP Library are shown here. More information can be found in the "SpaceWire/RMAP Library" document [15].

SpaceWire to TCP/IP Bridge On SpaceCube1

SpaceWire/RMAP Libraryは、POSIXとT-Kernelのシステムコールの差異を、ある程度吸収するレイヤーを備えているため、TCP/IP通信やSpaceWire通信、multi threadingに関しては、同じソースコードがLinuxやMac、SpaceCubeの上で再コンパイルするだけで動作します。

Since SpaceWire/RMAP Library has a layer to hide for some extent the difference between POSIX and T-Kernel system calls. Hence, The same source code on TCI/IP communication, SpaceWire communication or multi threading, can be compiled on Linux, Mac and SpaceCube

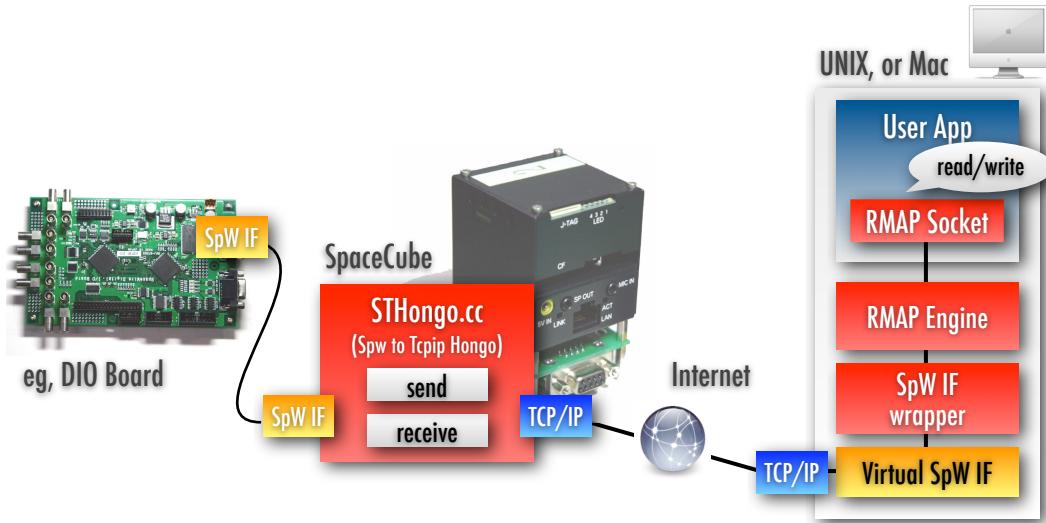
SpaceCube1は、SpaceWire I/Fだけでなく、Ethernet I/Fも備えているので、とくにSpaceWire/RMAP Libraryのうち、SpaceWire LibraryとTCP/IP SocketLibraryを用いて、SpaceCube1をSpaceWireとTCP/IPのプロトコルコンバータにして、普通のPCから透過的にSpaceWire I/Fを利用することができます。そのためのSpaceCube1用プログラムが、source_Executables内のsthongo (SpaceWire to TCP/IP Hongoの略)です。PC上のプログラムで、SpaceWireIFViaSpW2TCPIPBridge.ccという仮想的なSpaceWire I/Fをインスタンス化することで、TCP/IPレイヤーを透過して、SpaceCube1のSpaceWire I/FがPCから利用できます。

Since SpaceCube1 has an Ethernet I/F in addition to SpaceWire I/F, it can be used as a SpaceWire to TCP/IP protocol converter by using SpaceWire Library and TCI/IP SocketLibrary in SpaceWire/RMAP Library. Then it enables a normal PC to use SpaceWire I/F transparently through it. The program for SpaceCube1 is sthongo (SpaceWire to TCP/IP Hongo) in source_Executables folder. Once an instance of SpaceWireIFViaSpW2TCPIPBridge.cc, a virtual SpaceWire I/F is created in a program on a PC, SpaceWire I/F of SpaceCube1 can be used by the PC through TCP/IP layer.

実際に、POSIX環境でSpaceWire/RMAP Libraryをコンパイルして生成されるコマンドライン版rmaphongoを、PC上で起動すると、sthongoが動いているSpaceCube1サーバのアドレスを入力するよう促され、SpaceCube1に正しく接続できると、その後はSpaceCube1上でrmaphongoを動かしたときと全く同じように、コマンドラインからSpaceWire I/Fポートなどに対して、RMAP Read/Writeが行えるようになります。

For example, if command-line rmaphongo created by compiling SpaceWire/RMAP Library in POSIX environment is started on a PC, you are asked to input the IP address of SpaceCube1 server where sthongo is running. Then once the connection

is established, RMAP Read/Write becomes available to SpaceWire I/F ports, from command line, just like the case you use rmaphongo on SpaceCube1.



RMAP Target Simulator on SpaceCube1

SpaceWire/RMAP LibraryのSpaceWireIF classはパケットの受信だけを行うこともできます。受信したパケットデータをRMAPPacket classに設定して解釈させると、RMAPコマンドの情報(Write/Read、アドレス、データ長さ、戻りアドレス)を抽出することができます。

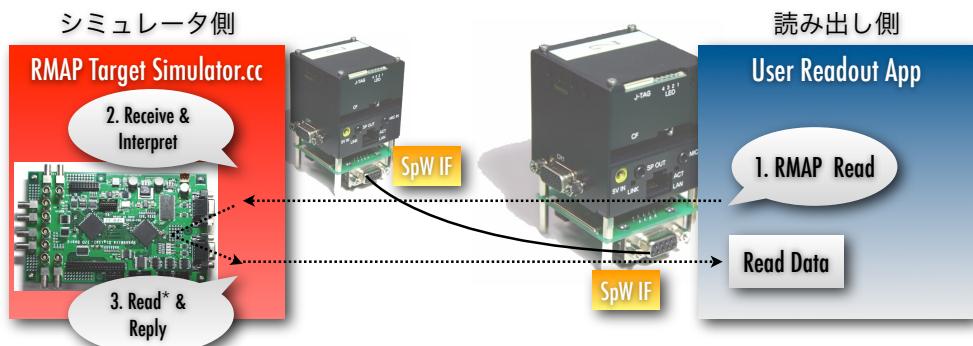
SpaceWireIF class in SpaceWire/RMAP Library can be used for only receiving packets. If the received packet data are passed to RMAPPacket class to interpret, information of the RMAP command (Write-or-Read, address, data length, return address) can be picked up.

あるアドレスへのRMAPアクセスに対して、あるデータを返すようなプログラムを記述すると、SpaceWire I/FボードのようなRMAPにおけるターゲット(正しくはRMAP Destination)の動作をソフトウェア的に模擬することができます。検出器の読み出し系開発において、検出器自体がまだ完成していないときにも、このような仕組みをもちいて検出器のSpaceWire/RMAPレイヤの動作をシミュレートして、読み出しプログラムの開発を開始することができるわけです。

A program to return some data by an RMAP address to some address can be used as a software-based simulator of an RMAP destination. For example, this can be used as a detector simulator that simulates the behaviors of SpaceWire/RMAP layer of the detector. This enables development of read-out program even when fabrication of an actual detector is not finished.

このようなプログラムのサンプルは、SpaceWire/RMAP Libraryのsource_Executables/main_rmaptargetsimulator.ccとして集録されています。

The sample program is source_Executables/main_rmaptargetsimulator.cc and is included in SpaceWire/RMAP Library.



参考文献 / References

- [1] S.M. Parkes et al., "SpaceWire: Links, Nodes, Routers and Networks", European Cooperation for Space Standardization, Standard No. ECSS-E50-12A, Issue 1, January 2003
URL : <http://spacewire.esa.int/content/TechPapers/documents/SpaceWire%20Standard%20ISWS%202003.pdf>
- [2] Parkes, S. & Rosello, J. 2003, "SpaceWire ECSS-E50-12A", International SpaceWire Seminar, Noordwijk, The Netherlands
URL : <http://spacewire.esa.int/content/TechPapers/documents/SpaceWire%20Standard%20ISWS%202003.pdf>
- [3] S.M. Parkes et al., "SpaceWire Protocol ID", European Cooperation for Space Standardization, Standard No. ECSS-E-50-11 Draft B, February 2005
URL : <http://spacewire.esa.int/content/Standard/documents/SpaceWire%20Protocol%20ID%20ECSS-E-50-11%20Draft%20B%202005.pdf>
- [4] S.M. Parkes et al., "RMAP Protocol Specification", European Cooperation for Space Standardization, Standard No. ECSS-E-50-11 Draft F, December 2006
URL : <http://spacewire.esa.int/content/Standard/documents/SpaceWire%20RMAP%20Protocol%20Draft%20F%204th%20Dec%202006.pdf>
- [5] T. Takahashi, "科学衛星データ処理系の将来展望", 宇宙科学シンポジウム, JAXA相模原キャンパス, 2005
URL : http://www.astro.isas.jaxa.jp/~takahashi/DownLoad/ISAS_Sympo_DataProcess2005_3.pdf
- [6] M. Nomachi, "次世代データ収集プラットフォーム開発", 「ストレンジネスで探るクオーク多体系」研究会, 仙台, November, 2007
URL : <http://nexus.kek.jp/Tokutei/workshop/2007/pdf/P2605Nomachi.pdf>
- [7] T. Takahashi et al., "SpaceCube 2 -- An Onboard Computer Based on SpaceCube Architecture", International SpaceWire Conference, Dundee, September, 2007
URL : presentation slide
<http://spacewire.computing.dundee.ac.uk/proceedings/Presentations/Onboard%20Equipment%20and%20Software/takahashi.pdf>
URL : proceeding
<http://spacewire.computing.dundee.ac.uk/proceedings/Papers/Onboard%20Equipment%20and%20Software/takahashi.pdf>
- [8] T. Hagiwara, "撮像型X線TESマイクロカロリメーターのデジタル信号処理", 東京大学理学系研究科修士論文, January, 2007
URL : http://www.astro.isas.jaxa.jp/~mitsuda/lab/index.php?plugin=attach&refer=Thesis&openfile=MThesis_THagiwara.pdf
- [9] H. Odaka et al., "Development of a SpaceWire-based Data Acquisition System for a Semiconductor Compton Camera", International SpaceWire Conference, Dundee, September, 2007
URL : presentation slide
<http://spacewire.computing.dundee.ac.uk/proceedings/Presentations/Missions%20and%20Applications%202/odaka.pdf>
URL : proceeding
<http://spacewire.computing.dundee.ac.uk/proceedings/Papers/Missions%20and%20Applications%202/odaka.pdf>
- [10] T. Yuasa et al., "Development of a SpW/RMAP-based Data Acquisition Framework for Scientific Detector Applications", International SpaceWire Conference, Dundee, September, 2007
URL : presentation slide
<http://spacewire.computing.dundee.ac.uk/proceedings/Presentations/Missions%20and%20Applications%202/yuasa.pdf>
URL : proceeding
<http://spacewire.computing.dundee.ac.uk/proceedings/Papers/Missions%20and%20Applications%202/yuasa.pdf>
- [11] W. Kokuyama, "衛星搭載用超小型重力波検出器の開発", 「高エネルギー天体現象と重力波」研究会, 東京大学本郷キャンパス, November, 2007

URL : presentation slide

http://www.gw.hep.osaka-cu.ac.jp/HE_ASTRO_GW/viewgraph/he_astro_gw07_1117_kokuyama.pdf

[12] K. Ishidohiro, “FPGAを用いた小型宇宙重力波検出器(SWIMμv)のデジタル信号処理系”, 重力波研究交流会, April, 2008

URL : http://tamago.mtk.nao.ac.jp/gw_talks/080404/talk1/gw_talks20080404final.ppt

[13] NEC Corporation, “SpaceWire Router IP 取り扱い説明書 Rev1.0”, NECST-SPF5PL-06004, March 2007

[14] T. Yuasa, “Development of SpaceWire-based waveform-sampling pulse height analyzer and its application to a hard X-ray detector”, The University of Tokyo, Master Thesis, 2008

URL : http://ceres.phys.s.u-tokyo.ac.jp/~yuasa/paper/mron/yuasa_mron.pdf

[15] T. Yuasa et al., “SpaceWire/RMAP Library”, 2008 (最新版はSpaceWire Wikiで入手可能)

オンラインで入手可能なその他の参考文献

[A] H. Odaka, T. Yuasa et al., “SpaceWireのつなげかた”

URL : http://www.astro.isas.jaxa.jp/~odaka/spacewire/documents/HowToConnectSpaceWire_1.0.pdf

(要パスワード。SpW-MLに問い合わせてください)

[B] Doxygen URL : <http://www.doxygen.jp>

[C] SpaceWire Wiki URL : <http://www.astro.isas.jaxa.jp/SpaceWire/wiki/>

[D] Xilinx ISEのダウンロード、マニュアル URL : http://japan.xilinx.com/ise/logic_design_prod/webpack.htm

[E] Yuasa, “SpaceWire-based Data Acquisition System”, 理化学研究所 牧島宇宙放射線研究室 地の共有ゼミ, 2008年4月

URL : http://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/conference/riken_200804/yuasa_spw_20080430_1230.pdf

[F] JTAGの解説 URL : http://www.ednjapan.com/content/issue/2007/03/content05_02.html

[G] Eclipse IDE URL : <http://www.eclipse.org/>

[H] SpaceWire/RMAP Library Online API Reference

URL : <http://www-utheal.phys.s.u-tokyo.ac.jp/~yuasa/spc/SpaceWireRMAPLibrary/apireference/html/>

参考書籍

[a] パーソナルメディア株式会社, “T-Kernel 組み込みプログラミング 強化書”, パーソナルメディア, ISBN978-4-89362-246-4, 2008

[b] Herbert Schildt, “独習C++”, 翔泳社, ISBN978-4798103181, 2002

[c] 塚越一雄, “はじめてのC++”, 技術評論社, ISBN978-4774108438

[d] NRIラーニングネットワーク株式会社, “EclipseではじめるC++”, 翔泳社, ISBN978-4798112220, 2007

[e] 坂巻佳寿美, “見てわかるVHDL”, 工業調査会, ISBN978-4769312130, 2002

[f] 吉田 たけお, “VHDLで学ぶディジタル回路設計”, CQ出版, ISBN978-4789833592, 2002