SpaceWire-to-TCP/IP converter
“main_sthongo” コマンドの使い方

高橋弘充、松岡正之 (広島大学)、湯浅孝行 (東京大学)
e-mail : hirotaka@hep01.hepl.hiroshima-u.ac.jp,
matsuoka@hep01.hepl.hiroshima-u.ac.jp,
yuasa@juno.phys.s.u-tokyo.ac.jp

2010/05/29

概要

“main_sthongo” は、SpaceCube を 、SpaceWire パケットと TCP/IP パケットのパケット変換機として機能させるコマンドである。これにより、SpaceCube 以外の Linux や MacOS などから、TCP/IP ネットワーク (Ethernet の LAN) を経由して SpaceCube 越しに SpaceWire パケットがやり取りでき (SpaceWire over TCP/IP)、それに基づいて、 SpaceWire ネットワークに接続されたノードとの間で、RMAP 通信が行えるようになる。以下では、この main_sthongo の利点と、実際にどのように実行すれば良いのかの準備と実行例、main_sthongo の概念など、についてまとめる。main_sthongo はひじょうに便利なコマンドであり、必要な準備も SpaceCube を立ち上げる作業とほぼ同じであるので、ぜひ main_sthongo を利用していただきたい。

1 “main_sthongo” の具体的な利点

main_sthongo コマンドを SpaceCube で実行すると、SpaceCube は TCP/IP ネットワークと SpaceWire ネットワークを仲介する一種のルーターとして動作します。これにより、SpaceCube 以外の Linux や MacOS などから、ネットワーク経由して SpaceCube 越しに RMAP 通信が行えるようになります。これには大きく以下の 2 点の利点があり、ひじょうに便利です。

プログラムを SpaceCube へ recv することなく、Linux 上で実行できる

プログラムを Linux の /usr/local/te の下でコンパイルした後に、SpaceCube へ recv や ftp することなく、コンパイルした Linux 上からそのままプログラムを実行できるようになります。つまり従来 SpaceCube 上でプログラムを実行する際には、コンパイルする度に実行ファイルを recv や ftp を使って転送しなくてはならなかったのが、main_sthongo によりその必要がなくなるので、プログラムのデバックがより簡単に行えるようになります。

プログラムを実行しているのは、ネットワーク越しの Linux である

main_sthongo を実行すると、SpaceCube は RMAP 通信を仲介する役割だけを担い、プログラムを実際に処理するのは Linux となります。このため、Linux の速い CPU を利用することができ、Linux 上で動作するプログラム (それが例え SpaceCube の t-kernel では動作しない機能があっても) を利用することができます。またファイルなどの保存先も Linux となるので、SpaceCube のコンパクトフラッシュには保存しきれないデータサイズを Linux 上で取得することも可能です。もしプログラムがエラーにより終了した際も、Linux 上でのプロセスが Abort することになり、SpaceCube の動作には影響は及びません。

よって、実験室で SpaceCube 上の main_sthongo を実行しておけば、あとは (RS-232C ではなく) ネットワーク越しにどの Linux や MacOS からも、その SpaceCube を経由した RMAP 通信を行えるようにな

ります。

ネットワーク越しにデータ転送を行っているので、SpaceCube と Linux や MacOS などを結ぶネットワークの通信速度が、データの処理レートを制限する場合があるかもしれないことに注意が必要です。

2 前準備

main_sthongo を実行するためには、「main_sthongo のコンパイル」と「SpaceCube がネットワークにつながっていること」の 2 点が必要です。

1. main_sthongo **のコンパイル**

(1) スペースワイヤーユーザー会のページ
http://www.astro.isas.ac.jp/SpaceWire/wiki/index.php?Download
から、最新の SpaceWireRMAPLibrary をダウンロード。

(2) /usr/local/te の下で SpaceWireRMAPLibrary_***.zip を解凍してコンパイルを行う。

```
$ unzip  /usr/local/te/SpaceWireRMAPLibrary_***.zip
$ cd /usr/local/te/SpaceWireRMAPLibyary/build
$ make
$ cd ../executables/t-kernal
$ ls                    <== ファイル生成確認
```

上記のように/usr/local/te/SpaceWireRMAPLibrary/build/ の下で make を実行すると、
main_sthongo が
/usr/local/te/SpaceWireRMAPLibrary/executables/t-kernel/main_sthongo
として生成されます。

(3) SpaceCube の gterm 上で、

```
[/SYS]% recv -d /usr/local/te/SpaceWireRMAPLibrary/executables/t-kernel/main_sthongo
```

をし、SpaceCube へ main_sthongo をコピー。

2. SpaceCube **のネットワーク設定**

(1) SpaceCube のネットワーク環境は、SpaceCube の gterm 上で、

```
[/SYS]% netconf -c
```

と実行すれば変更することができる。とくに、SpaceCube の IP アドレス (もしくはドメインネーム) は、Linux/Mac 上のプログラムから、SpaceCube に接続するときに必要になるので重要です。たとえば、SpaceCube と Linux/Mac でローカルなネットワークを形成する場合は、192.168.0.?? のようなアドレスが利用できます (Wikipedia の `http://ja.wikipedia.org/wiki/IP` **アドレス**の中の、「プライベート IP アドレス 」の説明などを参照してください)。

```
[/SYS]% netconf              （オプション無し）
```

と打てばネットワーク設定の確認を行うことができます。

(2) SpaceCube から、自分自身はもちろんプログラムを実行する本体としてして利用したい Linux や MacOS へ ping が通ることを確認。筆者の環境では、Linux 側から SpaceCube への ping は通っていませんが、main_sthongo は正常に動作しています。

3 実行例

main_sthongo を利用する環境は、SpaceCube 上で main_sthongo を 1 度だけ実行しておけば起動します。あとは Linux 側でプログラムの「実行、終了」を何度でも繰り返すことができます。なお Linux 側で実行するプログラムは、SpaceCube で動作する t-kernel 形式でなく、posix (Portable Operating System Interface) 形式でコンパイルされたものです。SpaceWireRMAPLibrary のディレクトリ構造を利用していれば、make のたびに t-kernel と posix の両者の実行ファイルが生成されているはずです。

main_sthongo を SpaceCube の起動時に自動的に起動する方法は、4.4 で説明します。自動起動させておくと、シリアルコンソールの接続が必要なくなるので、楽になると思います。

以下では、main_rmaphongo を例にして、具体的な画面表示を列挙します。

始めに gterm を使って SpaceCube で main_sthongo を起動

main_sthongo は、Linux や MacOS 側からプログラムを実行する前に、SpaceCube で 1 度だけ実行して起動するだけです。

main_sthongo の起動 @ gterm

```
[/SYS]% main_sthongo
===================================
SpaceWire to TCP/IP bridge : sthongo
  Takayuki Yuasa et al. June 2008

  This software is supported by
  SpaceWire/RMAP 'Hongo' Library.
===================================

opening SpaceWire IF...
Which SpaceWireIF implementation:
  Shimafuji IP Core          [2]
  Shimafuji Router IP Core   [3]
select > 2                        <== シマフジ電機製 SpW IP コア搭載 SpC なので「2」
Which port? [1,2,3] > 1           <== SpW 分配ボードの CN2 が port 1
SpaceWireIFSpaceCube1ShimafujiIPcore : initialize
spw0 opened
RX TimeOut : 1000 ms
TX TimeOut : 1000 ms
fpga ver. = 0x1000
SpaceWireIFSpaceCube1ShimafujiIPcore : open (20)
connecting ...
status = 0x0000003f
connected.
SpaceWireIFSpaceCube1ShimafujiIPcore : connect (0)
RX TimeOut : 100 ms
TX TimeOut : 100 ms
waiting for a new connection from client...

===== 今後 Linux 側からプログラムを受け付けられる =====
```

Linux 側からプログラムを実行されると @ gterm

```
starting daemons...
running...
running...
```

Linux 側のプログラムが終了すると @ gterm

```
data or control flag not received
SendThread : IP connection lost
SendThread joined
Quitting ReceiveThread
closing socket
deleting threads



waiting for a new connection from client...

===== SpaceCube 側では何もすることなく、
      Linux 側から次のプログラムを受け付けられる =====
```

main_sthongo の終了 @ gterm

```
waiting for a new connection from client...
の待機状態の際に「ctrl + c」で終了。

[/SYS]%
通常のコマンドラインへ復帰。
```

main_sthongo の起動後、Linux 側で posix ファイルの実行

SpaceCube 側で main_sthongo が起動していれば、Linux や MacOS 側からは posix 形式の実行ファイルを何度でも動作できる。

```
===== SpaceCube で main_sthongo が起動した後、
      SpaceCube とネットワークでつながっている Linux 上で =====

> cd /usr/local/te/SpaceWireRMAPLibrary/executables/posix/
> ./main_rmaphongo

input hostname on which sthongo (spw-tcpip bridge) is running :
***.***.***.*** <= SpaceCube の IP アドレスやホスト名を入力

SpaceWireIFOverIPClient::SpaceWireIFOverIPClient()
##############################################
### SpaceWire sample program : rmaphongo ###
###           ver 2.1 (20080624)         ###
###                                      ###
###   This program is totaly 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
...

===== あとは SpaceCube 上で t-kernel の
      コマンドを実行した場合とまったく同様 =====
```

4 補足

4.1 この実装の概念図

main_sthongo で実装している SpaceWire-to-TCP/IP converter の概念図は以下のようになっています (SpaceWire Wiki でダウンロード可能なドキュメント「SpaceWire/SpaceCube Tutorial」から抜粋)。

基本的に、PC 上で動作しているプログラムは、仮想的な SpaceWire I/F がその PC に直接つながっているかのように見えています。この SpaceWire I/F の実体は、SpaceCube 上で動作している converter プログラムと通信して、ユーザアプリケーションが送信した SpaceWire パケットを SpaceCube 経由で、その先の SpaceWire

ネットワークに流します。逆に、たとえば RMAP リプライパケットのように、SpaceCube が SpaceWire ネットワークからパケットを受信すると、SpaceCube は PC 上の仮想的な SpaceWire I/F にそのパケットを送信し、PC 上では、そのパケットが上位のレイヤーに渡されることで、あたかも PC が直接 SpaceWire パケットを受信したかのように見せかけることができます。

この方式では、PC 側から SpaceCube にパケットを送信する際に、パケットのサイズ情報など、付加的な情報もやり取りするため、原理的には転送すべき情報量が増大してしまい、転送速度が低下するように思われます。しかし実際の利用においては、RMAP パケットの解釈や、読み出したデータの保存など、CPU やディスクアクセスの性能に依存した処理が多いため、SpaceCube よりも PC 上でユーザアプリケーションを実行する方が、全体的なスループットが向上しています。

下図で、PC 上で動作している RMAPSocket や RMAPEngine は、SpaceCube で使用しているものと全く同じもの (同じソースコード) なので、ユーザアプリケーションは RMAP Read や RMAP Write などを、SpaceCube 上で実行するのと同様に PC から使用できます。

図中、Internet となっている部分は、LAN でも Internet でもかまいません。現実的かどうかはわかりませんが、たとえば本郷の東京大学から、相模原宇宙研の実験室にある SpaceWire ネットワークに接続して、SpaceWire 経由でアクセス可能な検出器のモニタリングを行う事もできるでしょう (ラウンドトリップ時間の制限で、スループットは構内 LAN に比べて大幅に低下すると思われます)。

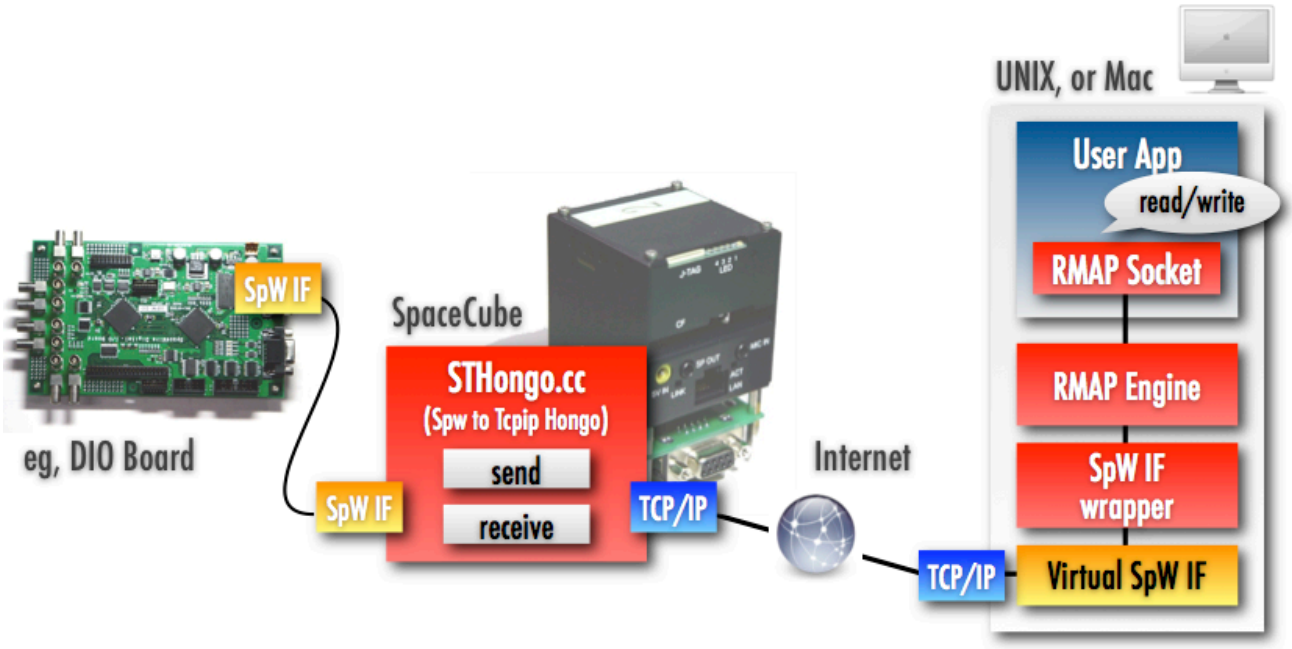

図 1 SpaceWire-to-TCP/IP converter の概念図。赤い四角で示されたコンポーネントは、SpaceWire/RMAP Library に含まれているので、ユーザは User Application の部分だけを記述すればよいことになります。

4.2 TimeCode

あまり用途はないかもしれませんが、PC 側からの TimeCode の送出もできるようになっています。プログラム内で、void SpaceWireIFOverIPServer::sendTimeCode(unsigned char flag_and_timecode) を利用してください。TCP/IP ネットワークを経由してから、本当の SpaceWire ネットワークに TimeCode が送られるので、PC 上での送出タイミングと、実際の送出タイミングにはある程度のずれが発生すると思われます。

4.3 TCP/IP のポート番号

PC と SpaceCube 上の main_sthongo の通信経路は、標準では TCP/IP の 10030 番ポートを利用しています。このポート番号は、ハードコードされている訳ではなく、TCP/IP 経由の仮想的な SpaceWire I/F を表現するクラス「SpaceWireIFOverIPServer」のヘッダファイル (source_SpaceWireRMAPLibrary/SpaceWireIFOverIPServer.hh) 中で、以下のように定義されています。ネットワークの制限などから、これを変更しなければいけない場合は、数値を書き換えてから、build/の中で、ライブラリ全体を make clean してから make しなおしてください。

```
class SpaceWireIFOverIPServer : public SpaceWireIF {
...
(中略)
...
public:
static const unsigned short spw2tcpiphongo_dataport=10030;
};
```

4.4 SpaceWire-to-TCP/IP converter を自動的に起動する方法

SpaceCube の電源投入と同時に、SpaceWire-to-TCP/IP converter が自動的に起動するようにしておけば、シリアルコンソールを SpaceCube に接続する必要がなくなり、PC から読み出しソフトを実行すれば SpaceWire 通信ができるようになります。以下にそのための設定方法を記述します。

SpaceCube(で動いている T-Kernel) では、CLI というコマンドラインプログラムが動作しています (Linux の bash のようなもの)。電源投入と同時に CLI が起動することで、シリアルコンソールからコマンド入力ができるわけです。

CLI は、初期設定スクリプトとして、STARTUP.CLI というファイルに記述されたコマンドを起動時に実行します。なので、main_sthongo の起動コマンドをこの中に書いておけばよいということになります。次の内容が、そのサンプルになります。

```
(main_sthongo を自動起動するための STARTUP.CLI)
/SYS/.xcli
alias do /SYS/$$PROGRAM.BOX/DLED /SYS/USR 0
if &DBG == 0
        do
        exit
else
        do &
endif

lodspg spw
main_sthongo shimafuji3_port1 &
```

前半部分は、SpaceCube にもともと入っている STARTUP.CLI の中身で、後半 2 行が新しく追加した部分です。一行目で、シマフジ製 SpaceWire ドライバ ("spw"というファイル名) を読み込んでいます。二行目で、sthongo を起動するのですが、引数として「シマフジ製 SpaceWire I/F(3 ポート版) のポート 1 を TCP/IP から利用する」という意味のオプションを与えています。最後の「&」は、バックグラウンドで実行せよ、ということです。これを付け忘れると、sthongo がフロントジョブとしてキーボードの入出力をとってしまうので、Ctrl-C で sthongo を終了するまで、シリアルコンソールからコマンドの実行ができなくなります。

このような内容の STARTUP.CLI を、シリアルケーブル経由 (recv コマンド) か、FTP で SpaceCube に転送し、SpaceCube を再起動すると自動的に sthongo が自動的に起動するはずです。

4.5 そのほかの SpaceWire over TCP/IP の実装

参考までに記述しておくと、STAR-Dundee 社の SpaceWire-USB Brick でも、SpaceWire over TCP/IP の機能が実現されているようです。こちらは、キャラクターレベルでの変換を行っているため、パケットの 1 バイト 1 バイトが、送られてきた順番で他端で再生されるようになっているそうです。繰り返しになりますが、main_sthongo の実装では、1 パケットを受信しおわってから、他端へと送信し、その内容が上位レイヤーに渡されるので、若干動作が異なります (ユーザアプリケーションから見たときの差異は小さいと思われます)。