5N-6

ICOTone - /dev/midi0

青柳龍也・小池帆平
(東大工学部)

1. はじめに

ICOToneプロジェクトでは、計算機を利用した新世代の音楽環境の実現を目指している。その第一歩として、既存のSUNワークステーションに楽器を接続し、SUNワークステーション上で多様なソフトウェアを開発している。

SUNワークステーション上にはUNIXオペレーティングシステムが動いているため、UNIXのデバイスとして楽器を接続することが望ましい。また、楽器側では、MIDI規格が一般的になっているため、MIDI規格に準拠することが望ましい。以上の観点から、我々は、MIDIインタフェースのハードウェアを製作し、そのドライバをUNIXに組込み、デバイス・ファイル/dev/midi0として利用可能な状態にした。

2. 問題点と解決法

MIDIインタフェースをUNIX/SUNに組込む際に問題となるのは、次の2点である。

1) MIDIコードは時間情報を含まないためリアルタイム制御を必要とする。一方、UNIX/SUNにはその機能が不足している。
2) MIDIの送受信のボーレートは31.25Kボーと速い。それに比べて、UNIX/SUNの割込み応答が遅い。

我々の解決法は、問題点1)については、

1> MIDIコードを拡張し、時間情報を加えたMIDI-tを設定し、UNIX/SUN側ではそのコードを使う。
2> インタフェースボード上にマイクロコンピュータを載せ、時間情報の解釈(リアルタイム制御)を行なう。

の2点である。この2点により、UNIX/SUNはリアルタイム制御から解放され、また、正確なタイミングで楽器を制御することが可能となる。

問題点2)については、

3> UNIX/SUNに比べ、割込み応答の速いマイクロコンピュータを使う。
4> マイクロコンピュータ側に大量のバッファを置く。

により解決する。(特に受信の際に)SUNの割込み応答が遅れても、バッファによりその遅れを吸収することができる。ICOToneの現主力シンセサイザDX-7は、最大4K強のデータを一度に送受信することがあるので、バッファの容量はその程度必要である。

また、受信するコードはマイクロコンピュータ側で時間情報を挿入したMIDI-tなので正確なタイミングの情報も得ることができる。

3. ハードウェア構成

ハードウェアは図のような構成になっている。CPU(6502)、4KバイトのRAM、2KバイトのROMの他、MIDIインタフェス用のACIA、タイマ割り込み用のPTM、SUNとの通信用の512バイトのFIFOから成っている。これらは1枚のボードに実装され、SUNのマルチバスに接続されている。

4. 拡張MIDIコード MIDI-t

一組のMIDIコードは、最上位ビットが1のステータスバイトと、それに続く数バイトのデータバイト(最上位ビットが0)から構成される。

MIDI-tは一組のMIDIコードと一組のMIDIコードの間に、その間の経過時間の情報を挿入したものである。

時間情報は、ステータスバイト$fd_{(16)}$で始まり、それに続く数バイトのデータバイトで表わされる。データバイトは、次のステータスバイトが来るまで続くものとする。データバイトは、msec単位で表わした経過時間を、下位から上位に7ビットごとに並べたものである。

例えば、ドの鍵盤を押し、1秒後にレの鍵盤を押した場合には、ドのkeyonを表わす3バイトのMIDIコードと、レのkeyonのコードの間に1秒(1000msec、2進数で111 1101000)の経過を表わすコードがはいり、次のようなシーケンスになる。

ICOTone - /dev/midi0
Tatsuya Aoyagi and Hanpei Koike
University of Tokyo

９０　３ｃ　４０
ｆｄ　６８　０３
９０　３ｅ　４０

ここで、９０はｋｅｙｏｎを意味するステータスバイトで、次のデータバイトが鍵盤の番号（３ｃがド、３ｅがレ）、次の４０は鍵盤を押した速さを表わす。

５．デバイスドライバ

デバイスドライバのソフトウェアは、ＵＮＩＸのデバイスドライバの書式に従って書かれている。インタフェースボードからのデータ到着、書込み可の割込みを受け、読込み、書込みを行なう割込みルーチンと、ユーザプログラムとのインタフェースとなるｒｅａｄ、ｗｒｉｔｅ、ｏｐｅｎ、ｃｌｏｓｅの各ルーチンから成る。

６．インタフェースボード上のソフトウェア

インタフェースボード上のソフトウェアは、リアルタイム制御を行なう。ＳＵＮ側から来たＭＩＤＩ－ｔコード中の時間情報を解釈し、適当なタイミングでＭＩＤＩコードを楽器に送出する。楽器側から来たＭＩＤＩコードには、その到着時刻を示す時間情報を挿入し、ＭＩＤＩ－ｔコードとしてＳＵＮに送出する。

７．今後の課題

今後の課題としては、次の２点があげられる。

１）　多数のタスクからのアクセス
２）　多数の楽器からの受信

これらの課題は、次のような場面を想定するとわかりやすい。

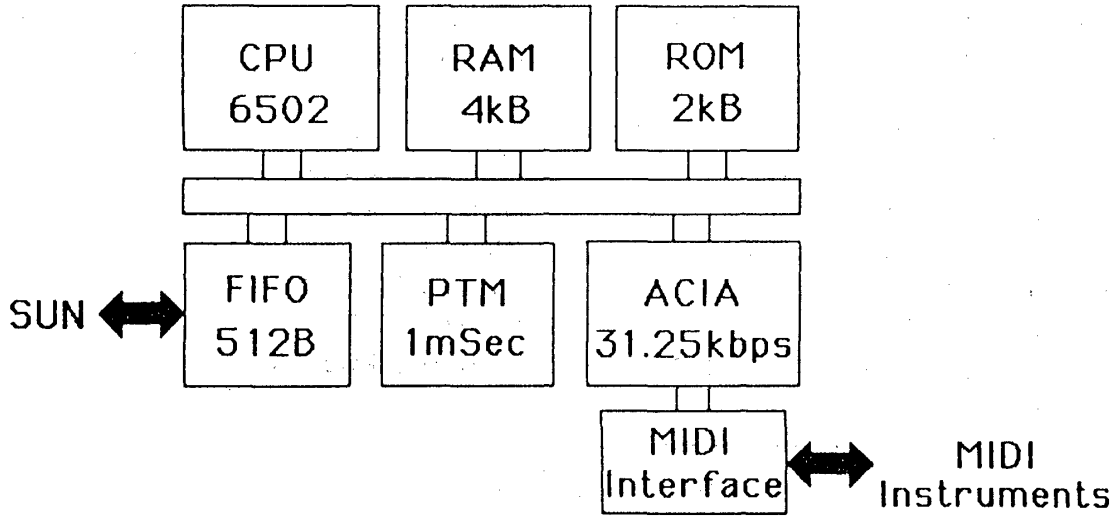
自動演奏用のウィンドウ、各楽器（シンセサイザ、リズムボックス等）制御用のウィンドウを開き、自動演奏をさせながら、演奏中に楽器の音色を変えたり、音量を調節したりする。

このような状況では、多数のタスクが／ｄｅｖ／ｍｉｄｉ０にアクセスする。ところで、一組のＭＩＤＩコード（ステータスバイト＋データバイト）は、必ず連続して送受信されなければならない。多数のタスクが同時にアクセスする場合、（送信、受信ともに）なんらかの方法でコードの連続性を保存しなければならない。これが１つめの課題である。

各楽器ごとにウィンドウを開き、制御する場合には、各楽器からＭＩＤＩコードを受信し、音色の名前等の情報を得たい。しかし、現在のハードウェア構成では、同時にいくつかの楽器にＭＩＤＩコードを送信することはできても（ＴＨＲＵ端子を用いる）、同時にいくつかの楽器からＭＩＤＩコードを受信することはできない。いくつかの楽器からＭＩＤＩコードを受信するためには、なんらかの方法でＭＩＤＩコードを多重化する必要がある。これが２つめの課題である。

以上の２点を解決するためには、ＴＨＲＵ端子を用いる現在の接続法ではなく、各楽器を独立に接続する必要がある。すなわち、インタフェースボード上に、各楽器に対応した（各ＭＩＤＩチャネルごとの）ＡＣＩＡを置く。６５０２の時間管理も、各ＡＣＩＡごとに行なう。ＳＵＮとインタフェースボード上の通信は、ヘッダを付けるなどの方法で多重化する。デバイスドライバ側は、各チャネルごとに／ｄｅｖ／ｍｉｄｉ０からｍｉｄｉ１５までの独立したデバイスとして使えるようにする。

このようにしても、多数のタスクが一つの楽器を制御する場合のコードの連続性の問題は依然解決されない。しかし、ほとんどの応用に関しては、これで充分であろう。



図．ハードウェア構成