

Re: MIDI conjunction device

Source: <http://coding.derkeiler.com/Archive/General/comp.arch.embedded/2006-03/msg00943.html>

- *From:* "Michael R. Kesti" <mrkesti@xxxxxxxxxx>
 - *Date:* Sat, 18 Mar 2006 09:28:23 -0800
-

I know that it's generally considered poor form to reply to one's own articles but I think that I'm justified in this case.

"Michael R. Kesti" wrote:

Paul Keinanen wrote:

Originally a flag variable InputSelected=NONE. When stat0 Rx interrupt is processed, the byte is written to OUT UART and setting InputSelected=IN0. When there is an interrupt IN1, it notices, that InputSelected is not NONE and stat1 is put into a queue. When dat0A is received, InputSelected is still IN0 and dat0A is written to OUT. dat1A goes to queue.

When dat0B is received, it goes to OUT and sets InputSelected to NONE (if no queued messages exists) or in this case to InputSelected=QUEUE, which will send out the queued bytes each time the TxEmpty interrupt from the OUT UART are activated. After the queue is empty InputSelected=NONE.

Thus the sequence would be

```

IN0-|stat0|dat0A|dat0B|-----
IN1----|stat1|dat1A|dat1B|-----
OUT-----|stat0|dat0A|dat0B|stat1|dat1A|dat1B|-----

```

Thus, the delay for the first (or non-overlapped) message is 1 byte time and no garbling occurs.

This should be easily handled completely in interrupt state without any polling or timer interrupts by using IN0 and IN1 RxFull interrupt and OUT TxEmpty interrupts.

To allow more channels and multiple queued messages, each input channel would need a buffer capable of holding a full input message

Re: MIDI conjunction device

and the OUT channel a queue of arbitrary length (even kilobytes). If the OUT UART is free, the received bytes are immediately sent. If the UART is in use by some other input channel, the local buffers for all other channels are transferred to the global queue when datxB Rx interrupt is processed. When the first message has been completely transmitted, the global output queue is drained using TxEmpty interrupts.

This structure has the disadvantage that if only dat0B and stat1 overlap, the transfer from IN1 local buffer to global buffer would be delayed until dat1B is received. Thus, when OUT datxB TxEmpty interrupt is received and the global OUT queue is empty, a check needs to be made for all input channel to find the input channel with most bytes in the local input buffer and select it for transmission and empty this buffer using TxEmpty interrupts.

Sure, Paul, this is certainly doable and shaves a few hundred microseconds of the time it takes data to get through the merger. It also precludes performing any kind of filtering, transforming or utilization of data as well as preventing the generation of running status on the output.

Since I wrote that there has been something bugging me about it and it came to me this morning. There is another issue concerning running status that invalidates Paul's approach to merging.

First, I should define running status. In the MIDI protocol, messages' first character is called the status character. The MIDI spec allows MIDI messages to be sent without their status characters (ie, just their data characters are sent) as long as previously transmitted messages had the same status. This is called running status. Running status thins the data stream removing redundant status characters.

Suppose that two note messages are received at one of a merger's inputs, the second of which uses running status, and a controller message is received at another of the merger's inputs with this timing:

```
IN0-|stat0|dat0A|dat0B|-----|dat0C|dat0D|-----
IN1-----|stat1|dat1A|dat1B|-----
```

Stat0, data0A, and dat0B are the first note messages, dat0C and dat0D are the second note transmitted with running status, and stat1, dat1A and dat1B are the controller message. Using Paul's merging algorithm, the output would be:

```
OUT--stat0|dat0A|dat0B|stat1|dat1A|dat1B|dat0C|dat0D|-----
```

A properly operating MIDI device that receives these data will interpret them as a note message followed by two controller messages with stat1 as

Re: MIDI conjunction device

the status of the second controller message. The second note message gets changed to a controller message which is, obviously, not the desired result.

The solution is to queue received messages, replacing any implied status characters with their actual values and (optionally) regenerate running status after the input queues are read and as the messages are transmitted.

Consider that, at 120 BPM, a 64th note is more than 31.25 milliseconds in duration and that the instruments that receive MIDI data take 10 milliseconds or more to even begin sounding the notes specified for it in received MIDI streams. Even several milliseconds of delay through a merger is just plain musically insignificant.

I think that the benefits of queueing complete received messages far outweighs the drawbacks.

This is especially true as queueing prevents unintended transformation of messages' meanings!

--

=====
Michael Kesti | "And like, one and one don't make
| two, one and one make one."
mrkesti at comcast dot net | - The Who, Bargain

.