Scalable Connectivity Processor for Computer Music Performance Systems

Rimas Avizienis, Adrian Freed, Takahiko Suzuki & David Wessel

Center for New Music and Audio Technologies (CNMAT)
University of California Berkeley
1750 Arch St., Berkeley, CA 94709

{rimas, adrian, takahiko, wessel}@cnmat.berkeley.edu, www.cnmat.berkeley.edu

Abstract

Standard laptop computers are now capable of sizeable quantities of sound synthesis and sound processing, but low-latency, high quality, multichannel audio I/O has not been possible without a cumbersome external card cage. CNMAT has developed a solution using the ubiquitous 100BaseT Ethernet that supports up to 10 channels of 24-bit audio, 64 channels of sample-synchronous control-rate gesture data, and 4 precisely time-stamped MIDI I/O streams. Latency measurements show that we can get signals into and back out of Max/MSP in under 7 milliseconds. The central component in the device is a field programmable gate array (FPGA). In addition to providing a variety of computer interface capabilities, the device can function as a cross-coder for a variety of protocols including GMICS. This paper outlines the motivation, design, and implementation of the connectivity processor.

1. Context and Prior Work

Hardware development for computer music performance systems has followed the standard pattern of technology evolution passing through the first two phases of design focus, function and price, to the final phase: usability. We have identified size and connectivity as primary usability issues for computer music performance systems. Laptop computers have recently become available with fast signal processing capabilities at a moderate cost and although their size makes them very attractive for musical performance, their constrained expansion capabilities limit connectivity. Currently there are no commercially available, low-latency, high-reliability, compact, multi-channel audio solutions for laptop computers. Furthermore the architecture of current laptops and most computers makes it impossible to synchronize acquired gestural data and sound I/O to satisfy the low latency/jitter needed for satisfactory reactive performance systems: 10±1ms (Freed, et al., 1997). The 10 ms latency criterion is not difficult to meet, but a maximum latency variation of ±1ms is difficult to achieve, especially when the stimulus gesture is represented as a MIDI event or a low rate signal from a non-sample-synchronous input source like a data acquisition card rather than a sample-synchronized audio input signal. The only computers with the requisite unified clock management and operating systems support for such tight synchronization are from Silicon Graphics. Unfortunately, even the smallest configurations of their machines, the O2 and Octane are too large and expensive for most performing musicians.

One of our key design goals was to eliminate virtually all latency variation in low sample rate inputs like those from gestural input devices.

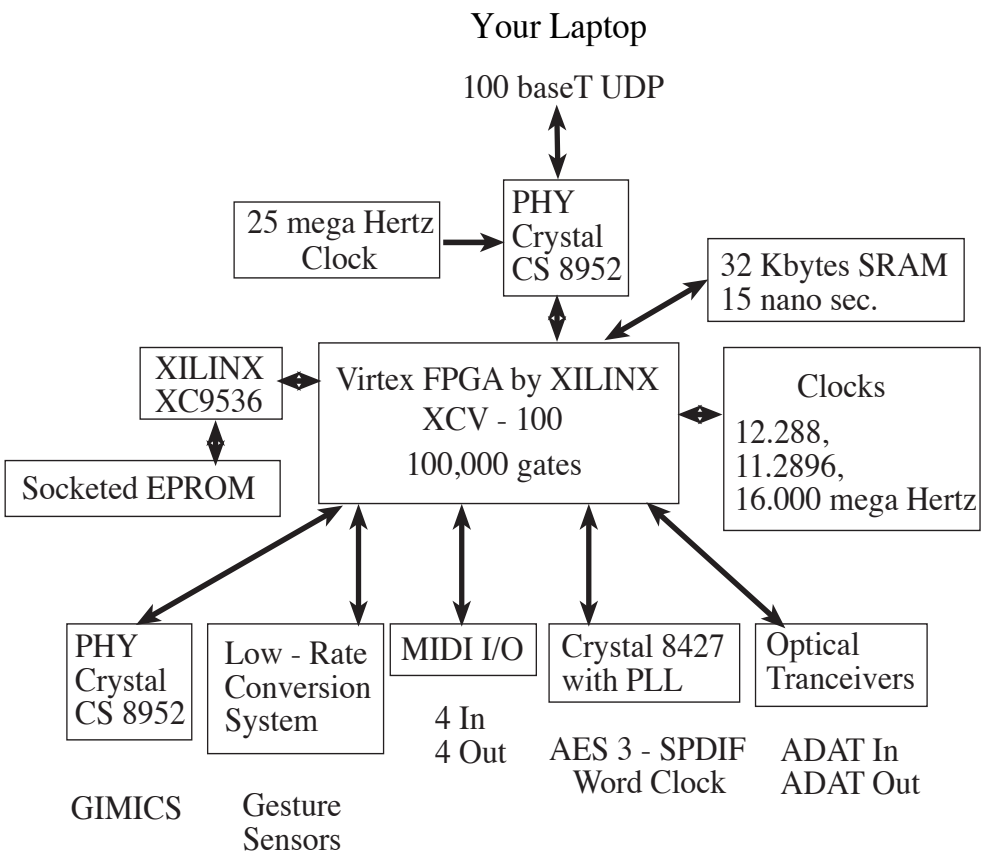
2. Introduction

We introduce here a new connectivity processor which solves the synchronization problem mentioned above and is readily connected to a laptop or any computer system with a 100BaseT Fast Ethernet port or digital audio port such as ADAT or AES/EBU.

The conventional approach for building a system to integrate gesture and sound is to combine a microcontroller, a DSP chip with A/D and D/A converters, and network interface chips for MIDI, Ethernet, AES/EBU, etc. Although this approach leverages the strengths of each chip, each processor comes with its own specialized low-level programming tools and development systems which complicate development and ultimately add cost as each chip has its own requirements for surrounding memory and associated glue chips to integrate these components.

Our alternative, more flexible approach supports scalable implementations from a few channels of audio and gestures to hundreds of channels by integrating all digital functions on a single field programmable gate array (FPGA). These functions are determined by compiling high-level hardware descriptions in VHDL into FPGA configurations (Skahill, 1996). VHDL is a acronym for VHSIC Hardware Description Language where VHSIC is an acronym for Very High Speed Integrated Circuits. This approach allows the considerable investment in developing the interface logic to each peripheral to be easily leveraged on a wide variety of FPGA's from different vendors and of different sizes up to millions of gates. FPGA's are a better match in this application than signal processors or general-purpose microprocessors because most of the communication protocols needed in multimedia and gesture systems are bit-serial. Signal processors and general-purpose processors operate on bytes and words and are not as well adapted to high-speed bit-serial protocols.

We have developed and tested VHDL descriptions for processing serial audio data for the SSI, S/PDIF, AES/EBU, AES-3, and ADAT industry standards. For gestural transductions we have VHDL modules that communicate with multichannel A/D converters, MIDI, and RS232 and RS422 serial ports. Although others have previously developed hardware language descriptions of many of these protocols for proprietary systems, our library of modules represents the first complete, independent suite available in VHDL. Our library also includes the glue to conform the asynchronous clocks required by each module to a unified synchronous sample rate clock. Our suite makes possible some unusual cross-coding strategies such as embedding gestural data in audio streams, thereby increasing temporal precision by exploiting isochronous data paths in the control processor.



We have developed a VHDL module which implements Fast Ethernet from the hardware layer up through IP to the UDP protocol of TCP/IP. This unusual module is of particular value in our application because Fast Ethernet ports are available on all modern laptops. Also, because of the commercial importance of Internet performance, Fast Ethernet implementations are extremely reliable and finely tuned on all modern operating systems.

A block diagram of our system is shown above.

3. Hardware Description

The central component is an FPGA: Virtex XCV-100 from Xilinx, Inc. The physical layer of the Ethernet communications protocol is handled by the Crystal Semiconductor CS 8952 PHY chip. There are two such chips on our device: one for the TCP/IP (in our case UDP) communications with the host computer and another for the emerging Global Musical Instrument Communication Standard (GMICS) (www.gmics.org). The low-rate conversion system is located in an external box and interfaces to the Connectivity Processor via a 50-pin hi-density D-Sub connector (most commonly used for SCSI II). Our initial low-rate conversion subsystem services 32 channels of analog input sampled at 1/8*(audio sampling rate) Hertz: 5512.5 Hertz for an audio rate of 44100 Hertz and 6000 Hertz for an audio rate of 48000 Hertz. Another more specialized I/O subsystem was developed for the continuous keyboard work described elsewhere in these proceedings (Freed and Avizienis, 2000).

The Connectivity Processor supplies a clock to the low-rate subsystem. For MIDI I/O we decided to provide for 4 input streams and 4 output streams. Each stream, of course, supports 16 MIDI channels. We support AES-3, S/PDIF, and word clock as well as ADAT Light-Pipe I/O. The connectors on the Connectivity Processor are as follows:

- RJ45 for GMICS
- 50 Pin hi-density D-Sub female
- 4 MIDI in and 4 MIDI out
- Locking BNC's for AES 3 and S/PDIF I/O
- Locking BNC's for Word Clock I/O
- Alesis Light-Pipe I/O (ADAT compatible)
- RJ45 for host computer 100baseT connection
- JTAG (not shown) on board for device programming

The support of the new GMICS protocol allows this box to function as a protocol bridge or digital "break out box" for GMICS-enabled devices.

IEEE-1394 shows promise as the basis for a multimedia connectivity solution and many laptops now include IEEE1394 so we are studying its implications, especially in light of Yamaha's M-LAN protocol.

4. Writing in VHDL

VHDL is a hardware description language. VHDL is widely adopted and is being used not only for the synthesis of large digital designs but also for documentation and design verification.

We provide below a code fragment to give the reader a feel for writing hardware behavior descriptions in VHDL. This VHDL code fragment defines an object which decodes a bimark phase encoded data stream like S/PDIF.

Module description

```
library IEEE;
use IEEE.std_logic_1164.all;
-- declare inputs and outputs to this object
entity bimark_dec is
  port (
       clk: in STD_LOGIC;    -- clock signal
       reset: in STD_LOGIC;  -- reset signal
       din: in STD_LOGIC;    -- data input
       dout: out STD_LOGIC   -- data output
    );
end bimark_dec;

--describe  the behavior of the object
architecture BEHAVIORAL of bimark_dec is
--state variables
signal last_dout, next_dout : STD_LOGIC;
begin process(DIN, CLK, RESET)
  begin
    if (CLK'event AND CLK = '1') then
      if RESET = '1' then -- synchronous reset
        NEXT_DOUT <= '0';
        LAST_DOUT <= '0';
      else
        NEXT_DOUT <= LAST_DOUT;
        LAST_DOUT <= DIN --remember input value
      end if;
    end if;
end process;

-- if previous input and current input are the
-- same, output '1', otherwise output '0'
  DOUT <= NEXT_DOUT XOR LAST_DOUT;
end BEHAVIORAL;
```

5. Transmission Protocol

We multiplex the audio, low-rate, and MIDI data into a single stream of UDP packets. This multiplexing strategy was developed in a previous project where we embedded low-rate data into an AES or S/PDIF stream (Freed and Wessel, 1998). The task of the software support for this protocol is to fish out upon reception of the UDP stream, the audio signals, the low-rate control signals, and MIDI events. Output transmission requires the embedding of the different forms of data into the UDP stream. The connectivity processor then unravels this stream and supplies the various forms of data at their appropriate connectors.

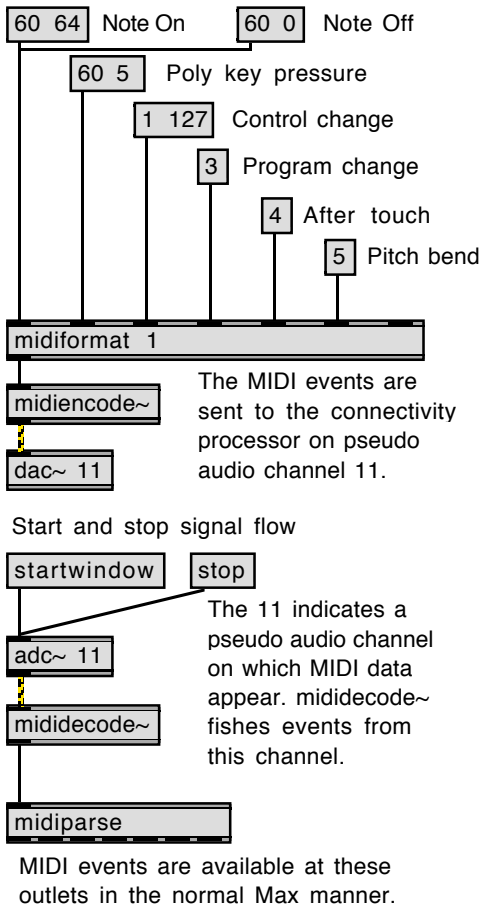
6. Software Support

We have developed custom drivers and objects for the Max/MSP (Puckette and Zicarelli, 1998) (Zicarelli, 1998) programming environment for the Power PC Macintosh Platform and the Open Sound World (OSW) (Chaudhary, et al., 2000) programming environment for Intel platforms.

We based the Max/MSP drivers on the ASIO specification developed by Steinberg. We chose in the initial implementation to use "pseudo audio streams" as the mechanism to input low-rate and MIDI data, representing these data as if they were audio for the purpose of getting into Max/MSP via ASIO. We accomplished this using the standard `dac~` and `adc~` objects with distinct channel numbers associated with these pseudo audio streams.

For the low-rate gestural input data we have developed an MSP object called `low-rate-in~` that upsamples the low-rate data to the audio-sampling rate. Languages such as CSOUND and SuperCollider have both an audio sample rate and a lower control sample rate called the K-rate. We would expect to bring the low-rate signals in to these languages directly as K-rate signals. We have learned that future versions of MSP will provide for multirate processing. OSW is fully multirate.

For MIDI we developed some rather unusual signal processing objects for MSP. MIDI input event data is fished out of the audio stream by `mididecode~` and MIDI output is embedded into an audio output stream by `midiencode~`. In MSP the diacritical mark ~ used as a suffix indicates that object is a sample-synchronous data flow object whereas Max objects without the ~ process discrete events. In the Max/MSP patch fragment below we use `midiencode~` to send MIDI data from Max to the interface. We also show how `mididecode~` is used to pass the MIDI data from the interface into the event world of Max.



7. The GMICS protocol

The impetus for the GMICS protocol was research carried out at CNMAT to digitize the electric guitar. Our original motivation was to provide a simple and robust encoding of multichannel audio and related control data in a single cable at the guitar itself. The multichannel encoding was important because not only did we want to provide for hexaphonic guitar effects where each string is sensed and treated separately but we also wanted to provide for more elaborate multi-dimensional sensing of each string. After we finished a number of initial studies concerning various protocols we handed our results off to our research sponsor Gibson Guitar Inc. Gibson's development team elaborated considerably on CNMAT's early efforts and developed the GMICS specification which is available at the following web site (www.gmics.org). The physical layer of GMICS is Ethernet. The connectivity processor described here gives a GMICS equipped electric guitarist access to a large array of hexaphonic effects that we have developed in Max/MSP all running on a very portable and easily set up system. It also provides for cross coding of the GMICS protocol to ADAT, AES-3, and S/PDIF.

8. Extensibility

The connectivity processor is programmable. New functionality can be easily added. In fact, since our current suite of VHDL modules only occupies about 50% of the FPGA we have room for more modules. For in system programmability we have provided a JTAG connector on the board of the connectivity processor so that the device can be interfaced to a computer running a VHDL development environment. Using an EPROM burner, new programs can be stored permanently in an EPROM memory and placed on the board. In fact, we have already begun experiments implementing bit serial signal processing algorithms on the FPGA. One such experiment was carried out by Norbert Lindlbauer, a visiting scholar to CNMAT. Lindlbauer implemented a bank of sinusoidal oscillators for a MIDI controlled additive synthesizer on the FPGA. (www.cnmat.berkeley.edu/~norbert)

There are a number of extensions to our environment that we plan to explore. When working with sensors one can envision a number of data conditioning algorithms that could be implemented in the FPGA itself. And, indeed, additional protocols to the ones we have implemented may be easily included.

9. Analog Subsystems

Our first analog subsystem whose digital output is connected to the Connectivity Processor via the D-Sub connector allows for 32 channels of input. The sample rates we support in this initial device are 44100/8=5512.5 Hertz and 48000/8=6000 Hertz. We roll off our anti-aliasing filter at 2000 Hertz.

Adrian Freed developed another subsystem for his continuous position keyboard controller that is reported upon elsewhere in these proceedings.

As the FPGA is programmable we can adapt the connectivity processor to send multichannel low-rate data to external devices.

10. Applications

We are especially excited about the very high degree of control intimacy we can obtain by having the low-rate gesture data synchronized so tightly with the audio.

Indeed, applications abound. And with the increased control intimacy many of the previous controller strategies like gloves, wands, force-sensing resistor systems, accelerometer systems, etc. should attain a new level of musicality. One area of particular interest here at UC Berkeley involves musical applications of micro electronic mechanical systems MEMS. We are working with members of Kris Pister's group on a new unobtrusive sensor system for the hands based on MEM accelerometers (Perng, et al., 1999).

11. Conclusions

The connectivity processor is a flexible and programmable piece of hardware that addresses many problems facing the performing musician. It uses the ubiquitous 100 BaseT Ethernet, is compact, provides for multi-channel audio I/O, low-rate gestural data input, MIDI I/O, and GMICS, and allows for cross coding among these formats. Furthermore, these data sources are tightly synchronized by the audio-rate sample clock, providing for a high degree of control intimacy. Latency is under 7 ms for the Max/MSP environment with jitter in the sub-nanosecond range.

12. Acknowledgments

This work was supported by a grant from the Digital Media Innovation Program and Gibson Guitar, Inc. We also received support in the form of equipment and software from Xilinx, Inc., We would also like to thank David Zicarelli of Cycling 74 for his advice and assistance. Additional thanks to Ahm Lee, Nate Yeakel, Matthew Wright, and Amar Chaudhary.

13. References

A. Chaudhary, A. Freed, and M. Wright (2000), "An Open Architecture for Real-time Music Software," proceedings of the International Computer Music Conference, Berlin, Germany.

A. Freed and R. Avizienis (2000), "A New Music Keyboard featuring Continuous Key-position Sensing and High-speed Communication Options," proceedings of the International Computer Music Conference, Berlin, Germany.

A. Freed, A. Chaudhary, and B. Davila (1997), "Operating Systems Latency Measurement and Analysis for Sound Synthesis and Processing Applications," proceedings of the Proceedings of the International Computer Music Conference, Thessaloniki, Hellas.

A. Freed and D. Wessel (1998), "Communication of Musical Gesture using the AES/EBU Digital Audio Standard," proceedings of the International Computer Music Conference, Ann Arbor, Michigan.

J. K. Perng, B. Fisher, S. Hollar, and K. Pister (1999), "Accelerator Sensing Glove (ASG)," proceedings of the ISWC International Symposium on Wearable Computers, San Francisco.

M. Puckette and D. Zicarelli (1998), "MAX - An Interactive Graphic Programming Environment (V3.5.9)," , 3.5.9 ed: Op-Code Systems.

K. Skahill (1996), *VHDL for Programmable Logic*. Reading, Mass.: Addison-Wesley.

D. Zicarelli (1998), "An Extensible Real-Time Signal Processing Environment for Max," proceedings of the International Computer Music Conference, Ann Arbor, Michigan.