# Design of a Digital Audio Input Output chip

Michiel M. Ligthart*, Andreas Bechtolsheim†
Giovanni De Micheli and Abbas El Gamal
Center for Integrated Systems
Stanford University
Stanford, CA. 94305

## Abstract

This paper describes the design of a Digital Audio Input Output (DAIO) chip that interfaces a standard 16/32 bit microprocessor bus with audio devices based on the AES protocol. This novel design provides general-purpose microprocessor systems with the ability to communicate with Compact Disk and Digital Audio Tape players. The chip is designed using High Level and Logic Synthesis tools from a functional description in a hardware description language.

## 1 Introduction

Since the widespread introduction of Compact Disk (CD) players in the 1980s, more and more of these players are presently equipped with a digital output. This output provides a means for the serial digital transmission of the audio signal on a single twisted wire pair. A protocol has been defined by the Audio Engineering Society (AES) [1]. This protocol not only allows for digital transmission between digital audio devices (Compact Disk and Digital Audio Tape players) but can also serve as an interface between those devices and computers.

Unfortunately, digital outputs from CD players, or inputs from Digital Audio Tape players (DAT), cannot be connected directly to a PC as they transmit data in a bit-serial way whereas PCs use parallel interfaces. Existing interface chips are designed for consumer audio and use bit-serial protocols specific to the respective manufacturer, e.g. the two chip set from Sony [2]. These chips are not suitable for general-purpose microprocessor systems. The DAIO (Digital Audio Input Output) chip, as described in this paper, solves this problem by converting the AES protocol to a standard 16/32-bit microprocessor bus (see figure 1). The DAIO converts the bit-serial signal from the CD or DAT player into a bit parallel signal for the microprocessor bus and vice versa.

The DAIO has been designed using the Olympus Synthesis System, currently under development at Stanford. A netlist for a Sea of Gates implementation is synthesized from a functional description in a hardware description language. The advantages of such a design approach are many. Synthesis systems reduce the design cycle since many laborious tasks are performed by programs rather than humans. Synthesis systems also give the designer the opportunity to search the design space, and to explore changes in the specifications easily.

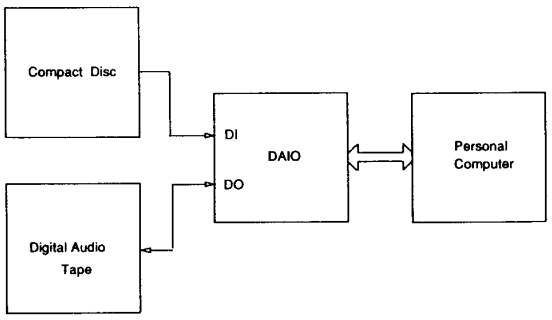


Figure 1: Digital Audio Input Output (DAIO) device.

This paper is organized as follows. In the next section the AES protocol is discussed. A global understanding of this protocol is required to understand the specifics of the chip. Section 3 describes the architecture of the DAIO and its operation in detail. Finally, in section 4.1 the Olympus Synthesis System is presented, and specific attention is given to the high level description of the DAIO.

## 2 The AES protocol

The AES protocol [1] specifies a recommended format for the bit-serial synchronous transmission of two channels of audio signals. Data is transmitted in blocks, each consisting of 192 frames. Each frame consists of two 32-bit subframes, one for each audio sample, called subframe A and subframe B respectively. Audio samples for the right and left channel alternate. The 32-bit subframe provides space for a 4-bit sync field (called preamble), a 24-bit audio quantity (only 16 bits are commonly used today) and bits for validity, user data, channel status, and parity (VUCP). The format is illustrated in fig. 2.

*Philips Research Labs, Signetics, P.O.Box 3409, Sunnyvale, CA 94086
†Sun Microsystems, 1550 Garcia Avenue, Mountain View, CA. 94043



IEEE 1989 CUSTOM INTEGRATED CIRCUITS CONFERENCE

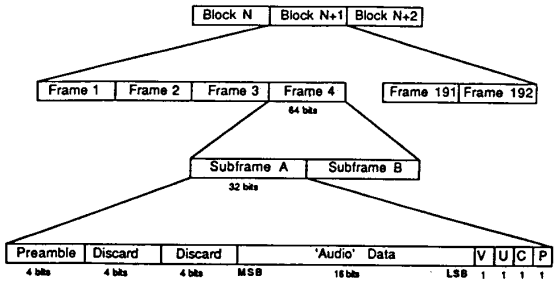


Figure 2: AES format for serial digital transmission

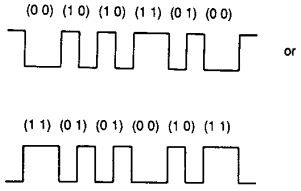


Figure 3: Example of biphase coding.

The source bits in the protocol are coded with a biphase mark, one of the family of self-clocking Manchester codes [3]. This code has the general property that all information is contained in the transitions. Clock extraction is simple and channel decoder synchronization is essentially immediate. The coding rules may be understood if each source bit is represented by a two-cell doublet. A doublet starts, and therefore also ends, with a transition. A source bit 0 corresponds to a doublet with no further transitions and a source bit 1 corresponds to a doublet with one transition between the two bits in the doublet. Hence, depending on the preceding data, a source bit 0 is represented by (00) or (11) and a source bit 1 is represented by (01) or (10). For example, a source signal 0-1-1-0-1-0 will be coded (00)(10)(10)(11)(01)(00), or (11)(01)(01)(00)(10)(11). The wave forms for this signal are shown in figure 3.

The preambles, necessary to indicate the start of a block, a subframe A or a subframe B, are coded such that they violate the biphase mark in a predefined way. A biphase violation occurs when a doublet does not end, and the next one does not start, with a transition, e.g. (01)(10). Three unambiguous types of preambles are used (see also figure 4):

preamble1 : Start of a subframe A, which is also start of a block;
preamble2 : Start of a subframe A elsewhere;
preamble3 : Start of a subframe B.

The protocol runs synchronously at the speed of the source. The destination must keep up with the source to avoid losing data.

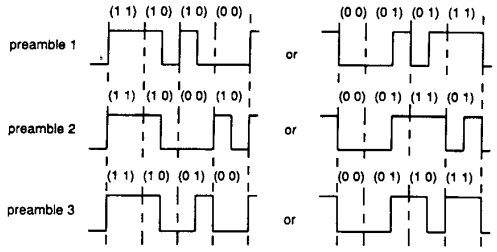


Figure 4: Preamble forms.

## 3 Architecture of the DAIO

The DAIO is a full duplex, fully symmetrical interface chip for the reception and transmission of digital audio signals. Receive in this context is defined as the data flow from the digital audio device to the microprocessor bus and transmit is defined as the data flow the other way around. Figure 5 shows a block diagram of the DAIO. Each direction, receive and transmit, has its own set of dedicated 32-bit registers. Registers for receive have a prefix RX and registers for transmit have a prefix TX. The register bank holds a total of 4 frames for each direction and is double buffered. The host processor is responsible for proper operation of the DAIO by means of controlling the MODE and STATUS registers and reading or writing the DATA and CTRL registers in time.

Before the DAIO starts actual data transfer, its MODE registers have to be written by the host processor. The MODE registers, one for receive (RXMODE) and one for transmit (TXMODE), select a direction for data transfer by means of an enable bit. The MODE registers also select the sampling clock frequency. Different frequencies are required since digital audio applications do not support a common frequency. The DAIO supports up to four clock frequencies.

### 3.1 Receive mode

After a direction and a clock frequency are selected the DAIO can start its actual data processing. In receive mode, the phase decoder has to lock into the incoming data stream and extract the actual bit values, preambles and the system clock. The phase decoder runs on the frequency specified in RXMODE. All other blocks run on the derived system clock. The sampling frequency is 640 (64 bits per frame * 10 times oversampling) times the audio frequency. A typical value is thus 640 * 44.1 kHz.

The phase decoder waits for a transition in the biphase signal and starts looking for a 'start of block' preamble. The preambles and actual bit values are extracted from the biphase signal by means of counting samples in a ten bits wide window. At a transition, a 4-bit counter is reset and ten samples of the biphase signal are taken. If the biphase signal is high the counter is incremented, if the biphase signal is low the counter is decremented. At the end of the tenth sample, the value of the counter reflects the actual value of the received bit. If a source bit 0 (doublet (00) or (11)) is sent the counter

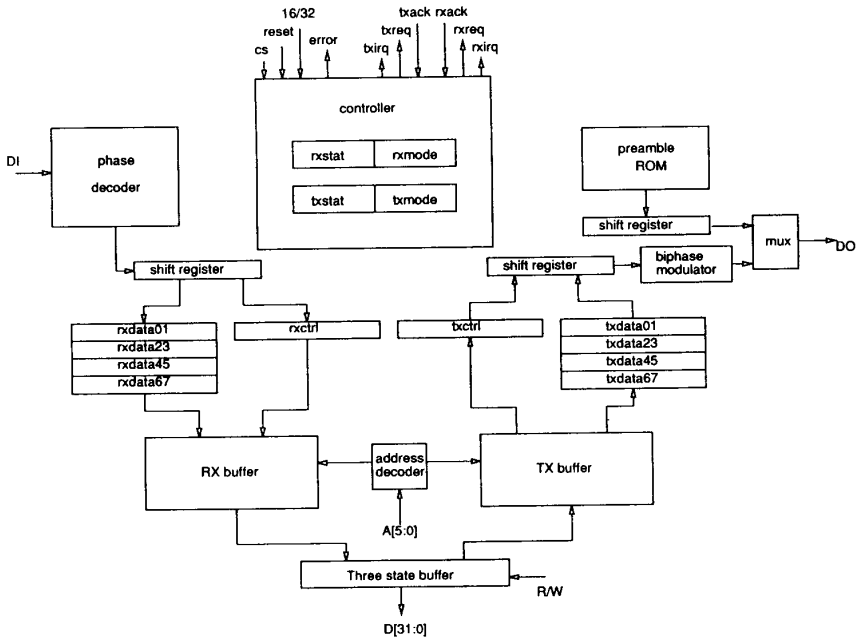


Figure 5: Block diagram of the DAIO.

will be either ten (0+10) or six (0-10). If a source bit 1 (doublet (10) or (01)) is sent the value of the counter will be zero. If at the end of the tenth sample no transition is found a flag indicating a biphase violation is raised. The last four extracted bits and the possible biphase violations are saved and monitored for the occurrence of a preamble. If a preamble is found this is indicated to the control section. This scheme is able to recover from jitter or spikes in the biphase signal up to two samples per window.

The extracted bits are shifted in a 20-bit shift register. As long as no preamble is detected, the superfluous bits are discarded at the last stage of the shift register. When a 'start of block' preamble is detected by the phase decoder the control section starts counting bits. At the end of the subframe, which occurs 32 bits later, the last 20 bits of the subframe will be in the shift register. Those are the significant bits of subframe A. The preamble and the 8 unused bits are discarded. Bits 4 to 19 of the shift register, the audio bits, are now parallel shifted into the leftmost 16 bits of RXDATA01. Bits 0 to 3, the VUCP bits, are distributed in RXCTRL such that at the end of 8 subframes all V, U, C and P bits are adjacent (VV..., UU..., CC..., PP...). In the meantime, bits for the second subframe, this time a subframe B, are shifted in the shift register. The control section checks for the occurrence of a 'start of subframe B' preamble and starts counting bits again. At the end of subframe B, bits 4 to 19 in the shift register are parallel shifted into the rightmost part of RXDATA01, and bits 0 to 3 are shifted into the appropriate positions of RXCTRL. At the end of subframe B the variable frame_count is incremented. Frame_count keeps track of the number of processed frames in the current block.

The shifting of subframes is repeated for the next 6 subframes, with the understanding that frames are loaded into RXDATA23, RXDATA45 and RXDATA67 respectively. At the end of the fourth full frame the contents of RXDATA and RXCTRL are loaded into a buffer accessible by the host processor. Four frames are stored instead of a single frame to give the host processor more flexibility in reading the data. At the end of a block, indicated by the value of frame_count, the control section waits for a new 'start of block' preamble.

The DAIO supports both programmed IO and DMA. In programmed IO mode an interrupt request is generated when the buffer is full. In DMA mode a DMA request is set. The DMA controller will acknowledge the pending request and establish access to the respective data register. A data register is selected through the address lines A[5:1]. The DAIO can operate in either 16 or 32 bit mode. In 16 bit mode the host processor must alternate left and right addressing of the data registers. The host processor must finish reading the data registers before the next 4 frames are available. If the buffer is not emptied the contents will be lost and an overflow flag will be set.

### 3.2 Transmit mode

Before the DAIO can actually start transmitting a serial biphase signal the host processor must fill the buffer registers. After writing those registers, the TXMODE register must be written with the clock frequency is selected and the transmit enable bit. Since transmit and receive are fully independent processes both can be selected at the same time, provided they operate at the same clock frequency. The host processor must be able to read/write all registers within the given time frame. Once transmit is selected the contents of TXbuffer are loaded into the registers TXDATA01, TXDATA23, TXDATA45, TXDATA67 and TXCTRL. An interrupt signal is sent to the host pro-

cessor indicating TXbuffer is empty. The buffer can be filled either through programmed IO or DMA. The four TXDATA registers contain the audio data for two subframes per register, and the TXCTRL register contains the corresponding VUCP bits for all 8 subframes. Notice that the audio data in a subframe is currently only 16 bits wide. The remaining 8 bits in a subframe will be transmitted as zero.

Two shift registers are maintained for the parallel to serial conversion. The first one generates the header of the outgoing biphase signal and is 24 bits wide, i.e. 12 source bits coded as doublets. This register contains the preamble followed by 8 biphase zero's. Dependent on the value of the last bit of the previous subframe, the preamble starts either with a (11) or a (00) doublet (see fig. 4). This also determines the coding of the trailing 8 zero's. The six different preamble values are hard-coded and are parallel shifted into the header shift register upon selection by a control signal. The second shift register is 20 bits wide and contains the audio data (16 bits) and VUCP bits (4 bits) for one subframe. The contents of this data shift register are not coded with a biphase mark.

At the beginning of a block a preamble 1 with corresponding trailing zero's is loaded into the header register. The contents of the header shift register are shifted out of the Digital Output (DO) port at the specified clock frequency. A typical value is 2*64*44.1 kHz (2 cells per doublet * 64 bits per frame * audio frequency). Depending on the alternating sequence of left/right channel data, the most significant or least significant 16 bits from TXDATA01 and the corresponding 4 VUCP bits from TXCTRL are loaded into the data shift register. Once the last bit is shifted out of the header shift register, the data shift generator takes over and starts shifting out bits at half the original frequency. The shifted bit is subject to biphase encoding which operates at full frequency. At the very last stage the signal is buffered by a D-latch.

To assure a continuous data stream the header shift register should be available right after the last bit is shifted out from the data shift register. This means a choice of whether to start the next preamble with a (11) or a (00) doublet must be made before the value of the last biphase coded bit is available. This is accomplished by taking the exclusive-or of the second bit of the previous doublet with the last one exclusive orred with the value of the very last bit. For example, if the contents of the data shift register end with ...01 and the 0 is coded in the biphase encoder (11) then the preamble of the next subframe should start with (00). (This a priori information is availables for each source bit and can be extended over more bits.)

After two subframes are processed the variable frame_count is incremented. At the end of the eighth subframe the new data from the buffer is loaded into the TX registers. At the end of a block a new block is started. If no new data is loaded into the TXbuffer all subframes are transmitted as zero's. Transmit is terminated by clearing the enable bit in TXMODE.

# 4 Design of the DAIO

The DAIO has been automatically generated from a high level behavorial description by the Olympus Synthesis Sytem. In this section both the synthesis system in general and its specific application for the DAIO are presented.

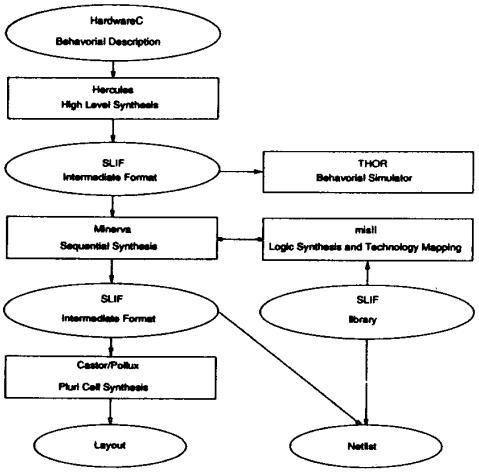


Figure 6: The Olympus Synthesis System.

## 4.1 The Olympus Synthesis System

The Olympus Synthesis System is a vertical system for synthesis of digital circuits from a behavioral description to layout and consists of several tools (see figure 4). A behavorial description is written in the hardware description language HardwareC and is synthesized by the high level synthesizer Hercules [4]. Hercules' output is a register transfer level description of the behavior in the intermediate language SLIF (Stanford Logic Intermediate Format). The register transfer level description can be simulated using the functional simulator Thor [5]. The SLIF file is used as input to the synchronous logic synthesizer Minerva [6], which uses MisII [7] for optimization of the combinational logic. At this point in the synthesis process, technology mapping is optional. One can either decide to map the optimized logic onto a specified library, or postpone technology mapping to a later stage. In either case, Minerva will write a new SLIF file containing a netlist description of the mapped network or the set of equations containing the optimized combinational and sequential logic. If the network has been mapped onto a specific library the SLIF file can be translated into the appropriate netlist format. Otherwise, the programs Castor and Pollux [8] can be used to create a custom layout using a structured cell based implementation.

## 4.2 Computer Aided Synthesis of the DAIO

The DAIO has been described as a set of independent processes in HardwareC. A process consists of a hierarchy of statements or procedures and executes concurrently and independently with respect to other processes in the system. For example, the receive part of the DAIO has been described in 6 processes. The top level process *daio()*, selects receive or transmit activity and a clock frequency from the MODE registers. Process *phase_decode()* extracts the value of the source bits, preambles and system clock. Processes *shift_register()* and *load_subframe()* shift bits into the appropriate positions in the RXDATA and RXCTRL registers. *Receive_control()* controls all this activity and keeps track of data correctness. Finally, process *buffer()* controls register read/write activity over the bus and is shared with transmit. Transfer of data between processes is accomplished through the use of parameters.

| process name | inverters | simple gates | complex gates | cells | flipflops | total eq. gates |
|---|---|---|---|---|---|---|
| daio() | 20 | 7 | 8 | 2 | 5 | 96 |
| receive() | 79 | 220 | 76 | - | 90 | 1319 |
| phase_decode() | 59 | 142 | 66 | 4 | 57 | 929 |
| shift_register() | 23 | 41 | - | - | 22 | 284 |
| load_subframe() | 89 | 107 | 162 | - | 194 | 2573 |

Table 1: Gate use for some processes in the DAIO.

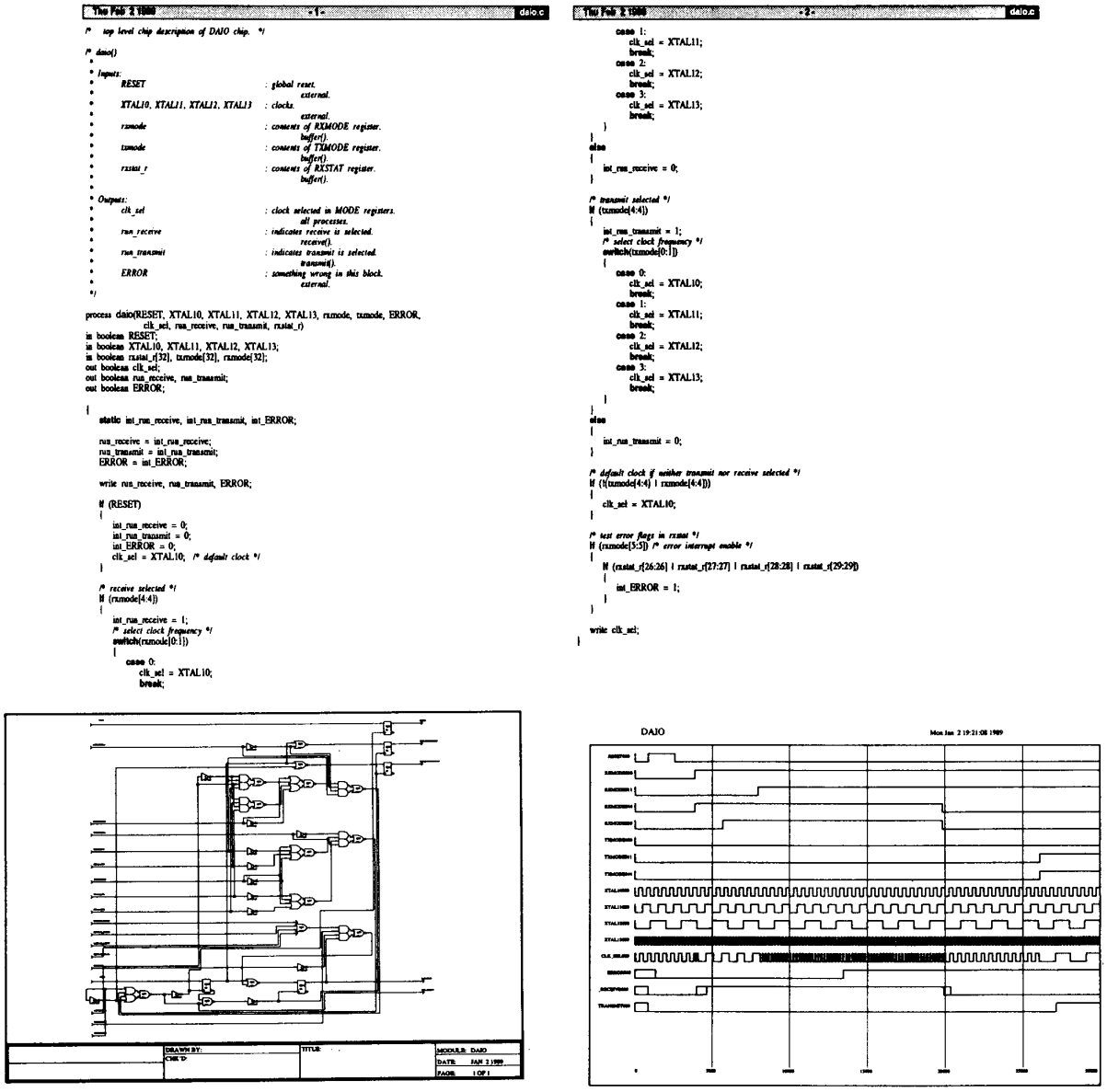


Figure 7: HardwareC description and synthesized schematic plus simulation result for the toplevel process of DAIO.

Each process is independently synthesized and mapped onto LSI Logic's macrocell library for the LCA 10000 Compacted Array series [9]. An LSI Logic netlist is generated for each process and different processes are interconnected using LSI Logic's Modular Design Environment [10]. Table 1 gives statistics on the number and type of cells used from the LSI library by several processes. The gate types are broken down into inverters, simple gates like nands and nors, complex gates like and-or-inverts, cells like muxes and full adders, and flipflops. The equivalent gate count is the number of 2-input nands/nors required to implement the circuit.

Hercules currently synthesizes circuits with master-slave flipflops as the default sequential element, without utilizing the wide range of available flipflop implementations. Therefore, in process *shift_register()*, the reset is implemented by means of adding logic to the m-s flipflop rather then utilizing an existing flipflop with built-in reset. Technology mapping also does not take full advantage of both polarities at the outputs of a flipflop. Only a rudimentary scheme for deletion of obsolete inverters has been implemented thus far. Therefore redundant inverters might be found near the $Q$ and $\overline{Q}$ outputs of flipflops. Table 1 does not reflect the use of some of the LSI macro functions. Macro functions are predefined instances of combinations of macro cells to describe complex elements, e.g. adders, comparators, counters, etc.. Hercules supports the use of macro functions because it is recognized that well known functions, like adders, should be instantiated from an optimally designed template rather than being synthesized again and again. Currently, macro functions are structurally specified in the HardwareC descriptions by the user.

Figure 7 shows an example of a synthesized part. The top level process that selects the direction and system clock is described in 80 lines of HardwareC code. After synthesizing and translation into LSI Logic's netlist format, a schematic was generated using LSI Logic's Modular Design Environment. The netlist was simulated for worst case conditions with time steps of .1 ns. The simulation results show outputs clk_sel, error, run_receive and run_transmit as a function of the inputs. The glitches in clk_sel are due to a change in clock frequency.

## 5 Summary and conclusions

This paper described the architecture of a Digital Audio Input Output (DAIO) chip that interfaces a standard 16/32 bits microprocessor bus with any audio device based on the AES protocol. This novel design accommodates general-purpose microprocessor systems with the ability to communicate with Compact Disk and Digital Audio Tape players.

The DAIO is designed with the aid of the Olympus Synthesis System, currently under development at Stanford University. The Olympus Synthesis System is a vertical system for synthesis of digital circuits from a behavioral description to layout. In the case of the DAIO, the layout is a mapping onto predefined cells in a Sea of Gates implementation.

Currently, the receive portion of the DAIO chip has been synthesized and simulated. Simulation has been done using worst case conditions and process parameters. We are currently working on the synthesis of the transmit part. The finalized DAIO chip will be fabricated in a Sea of Gates implementation. At present, the intermediate results show the feasibility of our synthesis approach. The paradigm used in Hercules, together with the supporting tools, produces working designs according to their specification in HardwareC.

## 6 Acknowledgements

The design of the DAIO originally started as a class project at Stanford. In the beginning many people were involved in both hardware design and software development for the synthesis system. Goodwin Chin, Woosang Park, Shin Chen and Michael Ang designed the architecture of the DAIO and created the first HardwareC descriptions. Their work and ideas are gratefully acknowledged. Larry Soule wrote the interface between SLIF and THOR. Frederic Mailhot wrote the SLIF parser that is used as front end for the Thor interface and functions as the interface to LSI Logic's netlist description. David Ku's help in writing and debugging HardwareC code was indispensable.

This research was sponsored by the Stanford Center for Integrated Systems seed fund no. 172c062 and by an NSF Presidential Young Investigator Award. Signetics is acknowledged for giving the first author the opportunity to pursue this research.

## References

[1] Audio Engineering Society, Inc., AES Recommended Practice for Digital Audio Engineering — Serial Transmission Format for Linearly Represented Digital Audio Data, Ansi S4.40-1985.

[2] CX23053, Digital Audio Data Receiving and Demodulating IC. CX23033, Digital Audio Data Modulating and Transmitting IC.

[3] P.Z. Peebles Jr., *Digital Communication Systems*, Prentice-Hall, 1987.

[4] G. De Micheli and D.C. Ku, "Hercules, a System for High Level Synthesis", *Proceedings of the $25^{th}$ Design Automation Conference*, 1988, pp.483-488.

[5] R. Alverson et.al., "Thor user's manual", *Stanford Technical Report CSL-TR-88-348 and CSL-TR-88-349*, January 1988.

[6] G. De Micheli and T. Klein, "Algorithms for Synchronous Logic Synthesis", *Proc. of the ISCAS*, 1989.

[7] R.K. Brayton et.al., "MIS: A Multiple-Level Logic Optimization System", *IEEE transactions on CAD, vol.CAD-6*, Nov. 1987, pp.1062-1081.

[8] F. Mailhot and G. De Micheli, "Automatic Layout and Optimization of Static CMOS Cells", *Proc. of the ICCD*, 1988, pp.180-185.

[9] Databook 1.5-Micron Compacted Array Technology, LSI Logic Corporation, July 1987.

[10] LDS Design System, user manual, LSI Logic Corporation, cu10-000128-50A, January 1987.