CAPTURE OF EVENTS MIDI IN PARALLEL WITH FPGAs'

M. Peña & A. De Luca

Center of Investigation and of Advanced Studies of the National Polytechnic Institute.
dlap@delta.cs.cinvestav.mx, max@computacion.cs.cinvestav.mx,

Received: January 13th, 2003. Accepted February 21th, 2003

ABSTRACT

The project consists on designing, in FPGA system, a special dynamic memory MCS-S (MIDI Capture System-Segmented) to capture, in real time and in parallel form, musical data that come from a group of instruments while they play in an orchestra, as well as to obtain their score. Inside the system, each single memory segment stores the notes corresponding to each instrument. The control system prepares automatically the necessary memory cells for each instrument and inserts new notes in each segment in parallel form. The electronic components of this system are programmed in VHDL, to carry out later the implementation in FPGA.

RESUMEN

El proyecto consiste en diseñar, en un sistema FPGA, una memoria dinámica especial llamada MCS-S (MIDI Capture System-Segmented) para capturar, en tiempo real y en forma paralela, datos musicales que provienen de un conjunto de instrumentos mientras tocan en una orquesta, y obtener su partitura. Dentro del sistema, cada segmento de memoria almacena las notas que corresponden a cada instrumento. El control del sistema prepara automáticamente las celdas de memoria necesarias para cada instrumento e inserta de forma paralela nuevas notas para cada segmento. Los componentes electrónicos del sistema están programados en VHDL para después realizar la implementación en FPGA.

KEYWORDS: MIDI, FPGA, VHDL

1. INTRODUCTION

A current problem, in musical ambient, is the impossibility to extract, from a complete orchestra global analogical recording, a specific instrument score with musical detail expression. The sound generated in the orchestra is blended inside the frequencies sound spectrum, with a great harmonic content, and is impossible to obtain information for each instrument in a separated form.

This article describes the design of a dynamic segmented memory MCS-S. This System receives musical digital information of MIDI type from an orchestra, separates information from each instrument and automatically stores it in the appropriate memory segment. Each segments content is processed by a computer to generate the scores (one per instrument) through software musical tools. Figure 1 shows the inputs and outputs of the system MCS-S.

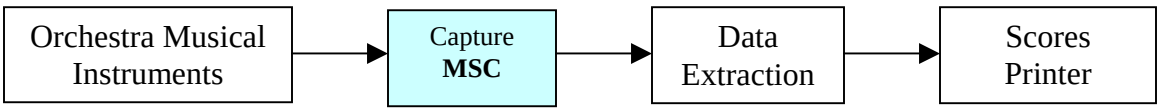

Figure 1. Capture system

The MIDI (Musical Instrument Digital Interface) system is a specific digital communications language for electronic musical instruments. The specification OSI MIDI 1.0 (1983) was defined pricipally by Roland Sequential Circuits. The MIDI interface is a standard protocol to connect hardware and software in an electronic musical systems. A 5-terminal connector of DIN type electrically connects the hardware system to an UART communications integrated circuit. The software is a programmed language that transforms musical notes into binary code and it is transmitted or received at 31.250 Bauds. The receiver (connector IN) is isolated through an optocoupler circuit. Two buffers (or double inverter) isolate the transmitter (connector OUT). The information received by the MIDI TRUE connector is copied and sent toward instruments nets systems connected in a daisy chain type [1].

2. DYNAMIC SEGMENTED MEMORY

The segmented memory MCS-S is made from the following main sections. The general control G unit is the one in charge of sending global character signals to the other subsystems. The memory control system C controls, through the cn control cells, all operations that memory cells makes. The dynamic segmented memory system M stores 8-bits data in the memory cells m_n . The time control system T detects and processes in each m_n component, the notes duration that the instruments send. The communications interface U receives, through its components u_n , a 8-bits data packages that each instrument sends; also a microcontroler MC synchronize all components contained in the system MCS-S.

The S arrangement, in expression (1), represents memory control system: the rows are data, memory cells, control cells and addresses; the columns are memory cells, where k = FFFFFh is the memory size

$$\begin{bmatrix} D \\ M \\ C \\ A \end{bmatrix} = \begin{bmatrix} d_0 & d_1 & \dots & d_{n-1} & d_n & d_{n+1} & \dots & d_k \\ m_0 & m_1 & \dots & m_{n-1} & m_n & m_{n+1} & \dots & m_k \\ c_0 & c_1 & \dots & c_{n-1} & c_n & c_{n+1} & \dots & c_k \\ a_0 & a_1 & \dots & a_{n-1} & a_n & a_{n-1} & \dots & a_n \end{bmatrix} \quad (1)$$

The general control G is a device that sends address pointers and signals to cells memory elements as registers, control cells and auxiliary components.

A control cell is a device designed to control directly a memory cell. We define the cells system as combined elements C, called control cells, denoted by c_n , such that c_{n-1} is the previous immediate neighboring cell, c_{n+1} are the later immediate neighboring cell, and $n + k$ is the memory size (expression 2)

$$C = \{c_0, c_1, c_2, c_3, \dots, c_{n-1}, c_n, c_{n+1}, \dots, c_{n+k}\} \quad (2)$$

The control cells system C is constituted by individual succession cells each one of which interacts with its neighbors: the immediate previous c_{n-1} and immediate later c_{n+1} . A cell c_n generates output signals that depend on the input signals sent by neighboring cells, general control and other devices.

The cells are activated in parallel by each clock pulse. In that way, the control of the writing or reading operations are made in concurrent form, achieving with this, the maximum operation speed required to write and read data from separated memory segments that each musical instruments have. If data is written in any position *n*, all cells next to it should generate control signs to displace one site all the data starting from this position, in a clock pulse.
The function that describes the memory cell control operation is given by expression 3

$$f(i_1, i_2, i_3, \dots, i_n) = s_1, s_2, s_3, \dots, s_k \quad (3)$$

where i_n are input variables and s_k are output controls.

A memory cell is a device designed to store data. The system memory segmented is defined as **M** combination of elements called memory cells denoted by of m_n , such that m_{n-1} is a memory cell previous immediate neighbor, m_{n+1} is the memory cell later immediate neighbor of mn , and $n + k$ is the size of the memory

$$M = \{m_0, m_1, m_2, \ldots, m_{n-1}, m_n, m_{n+1}, m_{n+2}, \ldots, m_{n+k}\} \quad (4)$$

The storage data system M is a memory serial arrangement of elements mn such that each one of them can read data that its neighbors have. We define each memory operation by function **f (*W, R, P, A, M*),** where the arguments are: W is writing operation; R is reading operation; P is address pointer; A is address memory cell; and M is the memory. When a writing data operation is made in any cell mn (insert), the cells that are next should make a displacement toward the right. Also, when a reading data operation (extraction) is made, the displacement will be made toward the left. This is carried out in parallel form, in a clockwise cycle

A pointer is a data structure of 7-bits to access memory. We define the pointers system as P elements combined $W\,p_1, \ldots, W\,p_{16}$, called pointers writing type, and $R\,p_1, \ldots, R\,p_{16}$, called pointers reading type, whose numeric values have been predetermined

$$P = \{W_{p1}, \ldots, W_{p16}, R_{p1}, \ldots, R_{p16}\} \quad (5)$$

An address is a 7-bits data structure that identifies the position of a cell memory in a serial arrangement of them. The addresses system is defined as the group A of elements

$$A = \{W_{a_1}, \ldots, W_{a_{16}}, R_{a_1}, \ldots, R_{a_{16}}, D_{a_1}, \ldots, D_{a_{16}}, \phi\} \quad , \quad (6)$$

such that**,** $W\,a_1, \ldots, W\,a_{16}$ are addresses writing type**,** $R\,a_1, \ldots, R\,a_{16}$ are addresses reading type, $D\,a_1, \ldots, D\,a_{16}$ are addresses data type, and ϕ is an empty address. The address zero, denoted by ϕ allows. Reducing the corresponding hardware, that makes the control more efficient.

The system memory is divided in 16 segments of variable size. A segment is a chain of memory cells with data which is accessed by 3-fixed addresses only: writing direction, reading and data direction. With this design structure, the memory uses in total 48 addresses. To write a data, the control points to the beginning of a segment (writing address), to read, it points at the end (reading address). The intermediate data placed between the principle and the end of each segment are marked with data addresses.

When the system is initialized, the memory is empty and ready to receive data. The addresses are ordered as it is shown in the expressions 7 and 8, notice that there are only writing addresses one for each segment such as

$$W_1 W_2 W_3 W_4 W_5 W_6 W_7 W_8 W_9 W_{10} W_{11} W_{12} W_{13} W_{14} W_{15} W_{16} \quad (7)$$

or

$$01_h 0A_h 12_h 1A_h 22_h 2A_h 32_h 3A_h 42_h 4A_h 52_h 5A_h 62_h 6A_h 71_h 7A_h \ \phi \quad (8)$$

3. OPERATIONS WITH THE MEMORY

A writing operation is the process of inserting a data inside a memory segment. The first time that a data is inserted, the control generates a reading address R and places it after a W; the second insertion generates a data address D that pushes R, the following operations generate addresses D that displace the previous ones to fill the memory.

$$\begin{bmatrix} t_0 \\ t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_n \end{bmatrix} \begin{bmatrix} W_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ W_1R_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ W_1D_1R_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ W_1D_1D_1R_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ \vdots \\ W_1D_1 \dots D_1D_1R_1W_2 \dots \end{bmatrix} \tag{9}$$

The expression 9 shows the state of addresses, in different clock pulses, when data are written to the segment number one: t_0 , initial state; t_1 , first data; t_2 , second data; t_3 , third data, etc.

A reading operation is the process of extracting a data from a memory segment. The process reads the data that has an address R. This and the following addresses should make a displacement to fill the hole. R disappears in the reading of the last data and the segment is empty.

The expression 10 shows the state of addresses, in different clock pulses, when data of the segment number one are extracted: t_0 , state of the memory; tn , three data exist: t_{n+1} first data; t_{n+2} , second data; t_{n+3} , reading of the third data, R disappears and the segment is empty.

$$\begin{bmatrix} t_0 \\ \vdots \\ t_n \\ t_{n+1} \\ t_{n+2} \\ t_{n+3} \end{bmatrix} \begin{bmatrix} W_1D_1 \dots D_1D_1R_1W_2 \dots \\ \vdots \\ W_1D_1D_1R_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ W_1D_1R_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ W_1R_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \\ W_1W_2W_3W_4W_5W_6W_7W_8W_9W_{10}W_{11}W_{12}W_{13}W_{14}W_{15}W_{16}\Phi \end{bmatrix} \tag{10}$$

4. STATES FOR DATA WRITING

A writing state is the situation in which the memory dwell before the arrival of a clock pulse. The cells change state in function of the state of the input variables of each one of them. The state changes or not, when one of the writing pointers that generates the general control becomes present to all the cells through the writing channel, indicating the address to insert the new data. The states are described next.

1. To change empty status to busy status that is the condition to insert the first data in a region pointed by W_p . The following cell c_{n+1} R should become an address to receive the new data.

$$\begin{array}{ccccccc} \cdots & m_{n-1} & & m_n & & m_{n+1} & \cdots \\ \cdots & c_{n-1} & & c_n & & c_{n+1} & \cdots \\ & A & & W & & A \leftarrow R & \\ & \overbrace{\quad}^{W_p > A > 0} & & \overbrace{\quad}^{W_p = A} & & \overbrace{\quad}^{W_p < A} & \end{array}$$

2. To indicate status not empty that is the condition to insert the second data in a memory region pointed by W_p It is similar to the first case, but the reading address R has been already created (first data) and the segment memory is no longer empty. The following cell c_{n+1} D should become an address to receive the next data.

$$\begin{array}{ccccccccc} \cdots & m_{n-1} & & m_n & & m_{n+1} & & m_{n+2} & \cdots \\ \cdots & c_{n-1} & & c_n & & c_{n+1} & & c_{n+2} & \cdots \\ & A & & W & & R \leftarrow D & & A \leftarrow R & \\ & \overbrace{\quad}^{W_p > A > 0} & & \overbrace{\quad}^{W_p = A} & & \overbrace{\quad}^{W_p < A} & & \overbrace{\quad}^{W_p < A} & \end{array}$$

3. To transform this address in R that is the condition to receive the first data in a memory region which is empty. This cell is the first one after the one that has the address pointed by W_p ; it contains the last data that entered in the line, the precedent cell has writing address W.

4. To transform this address in D that is the condition to receive the second data in a memory region, it is similar to the third case, but the memory is already busy, for this reason, a data address D is generated in cn and the new data is written in the memory mn . The following cell c_{n+1} should receive the address R that had cn . This cell is the first one after the one that has the address pointed by W_p ; it contains the last data that entered in the line, the precedent cell has writing address W.

$$\begin{array}{cccccc} \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & W & A \leftarrow R & A & \cdots \\ & \overbrace{W_p = A} & \overbrace{W_p < A} & \overbrace{W_p < A} & \end{array}$$

5. Forward displacement that is the condition for the cell to receive data and its predecessor's c_{n-1} address which should not have an address active W.

$$\begin{array}{cccccc} \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & W & A \leftarrow D & A \leftarrow R & \cdots \\ & \overbrace{W_p = A} & \overbrace{W_p < A} & \overbrace{W_p < A} & \end{array}$$

6. To make displacement toward the right that is the condition to copy address and data that has the precedent cell. This address should be zero, the previous cell should have a not active address W.

$$\begin{array}{cccccc} & \underbrace{d(m_{n-1}) \leftarrow d(m_{n-2})} & \underbrace{d(m_n) \leftarrow d(m_{n-1})} & \underbrace{d(m_{n+1}) \leftarrow d(m_n)} & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A(c_{n-1}) \leftarrow A(c_{n-2}) & A(c_n) \leftarrow A(c_{n-1}) & A(c_{n+1}) \leftarrow A(c_n) & \cdots \\ & \overbrace{W_p < A} & \overbrace{W_p < A} & \overbrace{W_p < A} & \end{array}$$

7. To make displacement to the right that is the condition to copy address and data that has the precedent cell. This address should be zero, the previous cell should have an address R.

$$\begin{array}{cccccc} & \underbrace{d(m_{n-1}) \leftarrow d(m_{n-2})} & \underbrace{d(m_n) \leftarrow d(m_{n-1})} & \underbrace{d(m_{n+1}) \leftarrow d(m_n)} & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & W & A \leftarrow W & A \leftarrow 0 & \cdots \\ & \overbrace{W_p < A} & \overbrace{A = 0} & \overbrace{A = 0} & \end{array}$$

8. To transform an address zero in R; it is presented for a cell whose address is zero and that has precedence of an address active W.

$$\begin{array}{cccccc} & \underbrace{d(m_{n-1}) \leftarrow d(m_{n-2})} & \underbrace{d(m_n) \leftarrow d(m_{n-1})} & \underbrace{d(m_{n+1}) \leftarrow d(m_n)} & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & R & A \leftarrow R & A \leftarrow 0 & \cdots \\ & \overbrace{W_p < A} & \overbrace{A = 0} & \overbrace{A = 0} & \end{array}$$

9. Generate no action that is the condition for any cell whose address is zero and that doesn't have precedence of addresses W (W_1 = 0) and R (R_1 = 0).

$$\begin{array}{ccccc} \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & W & 0 \leftarrow R & 0 & \cdots \\ & \overbrace{W_p = A} & \overbrace{W_p < A} & \overbrace{W_p < A} & \end{array}$$

10. Generate no action that is the condition for any cell whose address is bigger than zero and smaller than the writing pointer W_p .

$$\begin{array}{ccccc} & \overbrace{static} & \overbrace{static} & \overbrace{static} & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A & A & A & \cdots \\ & \overbrace{W_p > A} & \overbrace{W_p > A} & \overbrace{W_p > A} & \end{array}$$

5. STATES FOR THE READING OF DATA

For these cases, the general control sends the sign C R = 1 and the reading pointer R p to read data and to carry out displacements back in such a way that the left holes are empty.

11. To enable data output and to move back that is the condition to copy address and data that has the following cell c_{n+1} .

$$\begin{array}{ccccc} & \overbrace{static} & \overbrace{static} & \overbrace{static} & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A & A & A & \cdots \\ & \overbrace{W_p > A > 0} & \overbrace{W_p > A > 0} & \overbrace{W_p > A > 0} & \end{array}$$

12. Generate no action that is the condition for any cell whose address is bigger than zero, smaller than the reading pointer R p and that the following cell $cn+1$should not have active writing address R (AURI = 0).

$$\begin{array}{ccccc} & & \overbrace{d(c_n) \leftarrow d(c_{n+1})} & & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A & R \leftarrow A(c_{n+1}) & A & \cdots \\ & \overbrace{R_p > A > 0} & \overbrace{R_p = R} & \overbrace{R_p < A} & \end{array}$$

13. To change not empty state to empty state that is the condition to assure that the segment memory is unoccupied resetting the memory cell.

$$\begin{array}{ccccc} \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A & W & R & \cdots \\ & \overbrace{R_p > A > 0} & \overbrace{R_p > A > 0} & \overbrace{R_p > A > 0} & \end{array}$$

14. To transform this address in R and to absorb the next data that is the condition to make left displacement.

$$\begin{array}{cccccc} & & \underbrace{d(m_n) \leftarrow d(m_{n+1})} & & & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A & A \leftarrow R & R & \cdots \\ & \overbrace{R_p > A > 0} & \overbrace{R_p > A > 0} & \overbrace{R_p > A > 0} & \end{array}$$

15. To displace to the left. The cells that complete this condition only make left displacement.

$$\begin{array}{cccccc} & \underbrace{d(m_{n-1}) \leftarrow d(m_n)} & \underbrace{d(m_n) \leftarrow d(m_{n+1})} & \underbrace{d(m_{n+1}) \leftarrow d(m_{n+2})} & & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & A(c_{n-1}) \leftarrow A(c_n) & A(c_n) \leftarrow A(c_{n+1}) & A(c_{n+1}) \leftarrow A(c_{n+2}) & \cdots \\ & \overbrace{R_p < A} & \overbrace{R_p < A} & \overbrace{R_p < A} & \end{array}$$

16. Do nothing. The cells that complete this condition don't move, they remain static.

$$\begin{array}{cccccc} & \underbrace{static} & \underbrace{static} & \underbrace{static} & & \\ \cdots & m_{n-1} & m_n & m_{n+1} & \cdots \\ \cdots & c_{n-1} & c_n & c_{n+1} & \cdots \\ \cdots & 0 & 0 & 0 & \cdots \\ & \overbrace{A=0} & \overbrace{A=0} & \overbrace{A=0} & \end{array}$$

6. CONCLUSIONS

So far, we have designed VHDL programs to implement memory up to eight inputs of cells feigned with Xilinx whose initials tests were satisfactory. We describe the equations of each one of the output signals in function of the input signals to implement in software the detailed behavior of the components that the memory control system has. Software tools are available for the extraction and printing of the scores, besides tests and results obtained with commercial software.

7. ACKNOWLEDGEMENT

We thank the support received from the authorities of the institutions: Center of Investigation and of Advanced Studies (CINVESTAV) of the IPN, Section of Calculation of the Department of Electric Engineering, likewise, the Superior School of Mechanical and Electric Engineering (ESIME) of the National Polytechnic Institute, Section of Studies of Graduate Degree and Investigation.

8. REFERENCES

[1] Anderton, C. MIDI for Musicians, Amasco Publications, New York/London/Sydney, 1986.

[2] Gourlay, J. S. A Language For Music Printing, Communications of the ACM, may 1986, v. 29, No. 5, pp. 388-401.

[3] Takebumi I., Manning P. D., & Purvis A. Distributed Parallel Processing: Lessons Learned from a 160 Transputer Network, Computer Music Journal, Volume 21, Number 4, pp. 42-54, Winter 1997.

[4] Peña, G. M., Algoritmos Para Simulación de una Orquesta Sinfónica en una Máquina Distribuida o Paralela, Mex., Instituto Politécnico Nacional, CINVESTAV IPN, 1998.

[5] Peña, G. M., Mi Nena, en Delfines del Trópico, Dir: A. Pacheco, stereo L.P.BO-06, Discos Boga 1/3 rmp (Masterworks), siete composiciones, 1981.
[6] Rader, G. M., Creating Printed Music Automatically, Mesa State College, Grand Junction, Colorado, Computer IEEE, june 1996, pp. 61-68.
[7] Selfridge-Field, E. DARMS, Its Dialects, and Its Uses, in Eleanor Selfridge-Field, ed., Beyond MIDI: The Handbook of Musical Codes, The MIT Press, USA, 1997. pgs. 630. AAA. pp: 163-174.

Authors Biography

Maximino Peña-Guerrero

Was born in Pachuca Hidalgo, México, in 1949. Graduate as Comunications & Electronic Engineer (ESIME-IPN, 1983). He has a M.Sc. degree in Electric Engineering (CINVESTAV-IPN, 1987). He prepares the PhD thesis in Electric Engineering (CINVESTAV-IPN). SNI and National Academy of Sciences member since 1987. He has been professor and researcher in the National Polytechnic Institute, México, from 1981, and CINVESTAV-IPN, from 1987.

Adriano de Luca-Pennachia

Was born in Foggia (Italy), graduate in Mathematics and Physics at Sto. Severo (Foggia), Doctorate in "Nucleonica e Automazione" at Milano (Italy). Member of SNI since 1986. He has been working during the last 10 years in ININ, Mexico, conforming various teams of researchers and participating actively in the group that gain the National Prize of Applied Instrumentation in the 1974. From 1982 to 1985 worked in the SGS Thompson in Milano. He obteined U.S.A. Patent N. 07/483915 in 1991: "Digital anode to determine the location of electrons in a given surface" and published a text Book "Digital Systems" by Metropolitan Autonomous University in 1992, as well as 92 articles.