Consiglio Nazionale delle Ricerche

ISTITUTO DI ELABORAZIONE DELLA INFORMAZIONE

PISA

SERIAL MICROPROCESSOR ARRAY FOR
RADAR SIGNAL PROCESSING

P. Corsini, G. Frosini, G. Galati
F. Grandoni, M. La Manna

L76- 15

- Presentato al Second Symposium on "Microprocessing and microprogramming", Venice, October 1976.

Second Symposium on Micro Architecture, M. Sami, J. Wilmink, R. Zaks (eds.)

SERIAL MICROPROCESSOR ARRAY FOR RADAR SIGNAL PROCESSING

P. Corsini*, G. Frosini**, G. Galati***, F. Grandoni***, M. La Manna***

*Dipartimento Sperimentale di Elettrotecnica ed Elettronica, Università di Pisa, Pisa, Italy.
**Istituto di Elaborazione della Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy.
***Selenia, Industrie Elettroniche Associate, S.p.A., Roma, Italy.

Abstract

A digital system for signal processing in search radars is presented. The proposed organization exploits the high parallelism inherent in radar data. The main subsystem is an array of microprocessors, driven by a microprogrammed control unit. Serial arithmetic on variable length operands is implemented in the microprocessors, and fractional two's complement number representation is used. Efficient algorithms for the most frequently encountered operations are mechanized, such as a uniform-shift-of-two multiplication algorithm.

1. INTRODUCTION

In modern radar systems there is a widespread use of digital logic in the video processing section. Owing to the high throughput required, each processing algorithm is executed by a dedicated circuit, so that special purpose hardware carry out tasks like pulse compression, moving target indicator filtering, integration, and so on. Each circuit behaves like a station of an assembly line, and the digitized signal goes through this line undergoing the required processing.

Some criticism can be made of this approach. The design of a digital processing system is a cumber some task that must be carried out anew for each new radar. Moreover, any change in the processing algorithms requires a hardware redesign. These inconveniences could be avoided by using a stored-program processor, but until now the throughput required could not be achieved because of technological limits. Nowadays, the speed of digital logic circuits and memories has been increased enough to allow the design of a programmable radar signal processor. The conventional sequential organization is still too slow for this purpose. To attain higher throughput two approaches are devisable, exploiting two different types of inherent parallelism: parallelism in algorithms and parallelism in data.

This work has been sponsored by the Convention between Selenia S.p.A. and Consiglio Nazionale delle Ricerche, Pisa, Italy.

The first form of parallelism has been investigated in [1], and is exploited by configuring the processor arithmetic section as an ensemble of few highly parallel arithmetic elements designed to best perform in executing a specified set of algorithms. An example is the arithmetic element described in [1], having four multipliers and four adders. To handle the total radar workload, several units are necessary; the result is a complex operational structure, with the additional shortcoming of a low efficiency in the execution of algorithms different than the considered ones.

The radar algorithms show another kind of parallelism, namely in data. Radar data base, as will be shown in the next section, may be thought of as a time series of vectors, having a large number of components. It has long been realized that vector-structured data can be efficiently handled by array processors [2], [3], [4]. An application of the array architecture to radar signal processing can be found in [5].

A feasibility study on programmable radar signal processors, carried out as a part of a research project sponsored by the Selenia-CNR Convention, led to the conception of the Serial Microprocessor Array (SMA). The main sub-systems of the SMA are a Master Computer and a Slave Computer. The Master Computer controls the program flow, and sends the array instructions for execution to slave computer. The slave computer is constituted by two parts: a) the arithmetic array consisting in an array of serial microprocessors, each endowed with its own memory; b) a microprogrammed array control unit, that accepts array instructions and sends microorders in parallel to all microprocessors.

The serial microprocessors have a low complexity so that a large number can be used without excessive cost. Other advantages of this organization are the low logical complexity, ease of programming, and inherent availability and graceful degradation. The structure of the SMA, as shown in the following sections, is suited for the broad class of search radars.

2. RADAR DATA BASE

In a search radar the transmitted signal (Fig. 1a) usually consists of a series of pulses, M pulses for every antenna turn, emitted at discrete time instants $\ldots,t_{i-2},t_{i-1},t_i,\ldots$. The interval between two consecutive time instants (*pulse repetition interval*) is not necessarily constant but may assume cyclically some values $T_1,T_2,\ldots,T_\psi$, ψ being a submultiple of M.

The received signal is digitized (Fig. 1b) with constant sampling period T_c often equal to the duration of the transmitted pulse. In general the receiver is coherent, and supplies in-phase and quadrature components, so that the obtained samples are complex numbers. From the sequence of samples obtained in a repetition interval, only the first N samples are retained for processing, as the others contain no extractable information. The constant N is such that $NT_c=T<T_{min}$[1]. The difference between the actual repetition interval and T is referred to as "dead-time". Typical values for the mentioned quantities are $M\simeq 6000$, $T_1,T_2,\ldots,T_\psi \simeq 1.2-1.4$ msec, $\psi\simeq 3-6$, $T_c\simeq 1$ μsec, $N\simeq 1000$.

Let us schematize the radar system (Fig. 2) as a point Φ, and the environment as a plane. The "covered area", that is the circle centered in Φ and having a radius equal to vT/2, v being the velocity of light, can be thought of as decomposed 1) in N rings having the same thickness $vT_c/2$ and 2) in M sectors, the m-th of which is determined by the angle swept by the antenna in the time interval between t_j and t_{j+1}, where $m=|j|_M$[2]. Each intersection of a ring with a sector is called "resolution-cell". The resolution-cell determined by the n-th ring, $n=0,1,\ldots,N-1$ and by the m-th sector, $m=0,1,\ldots,M-1$ is called (n,m)-th resolution-cell, and n and m are called respectivaly, *range-coordinate* and *azimuth-coordinate*.

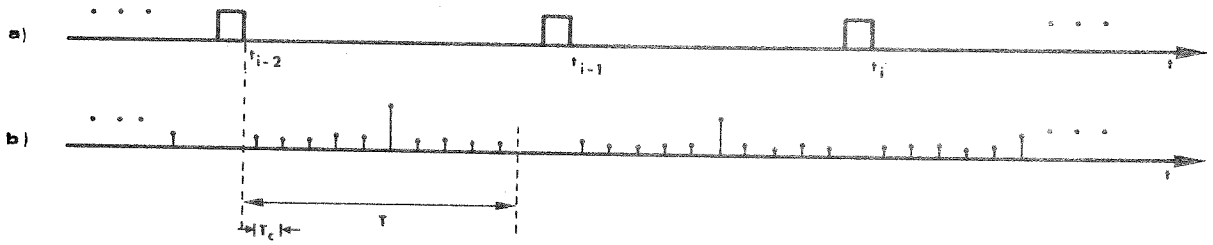

Fig. 1 a) Transmitted signal; b) Received signal (one component).

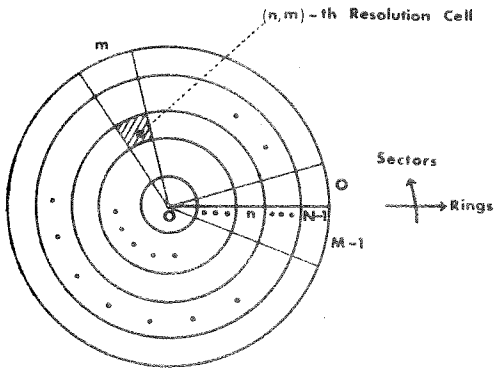

Fig. 2 Schematization of covered area.

The samples supplied by the analog-to-digital converter and considered for processing can be thought of as arranged in a time series of vectors

$$\ldots\ X^{i-2},X^{i-1},X^i,\ldots \qquad (2)$$

where

$$X^p=\{x_n^p \mid n=0,1,\ldots,N-1\},\quad p=\ldots,i-2,i-1,i,\ldots$$

represents the sequence of samples supplied in the time interval $(t_p,\ t_p+T)$. Each vector X^p is available after a time interval greater than or equal to Tmin with respect to the previous one. The element x_n^p of X^p is associated to the (n,m)-th resolution-cell, where $m=|p|_M$. This element represents the echo returning from the pertinent resolution-cell and, in less measure, from the cells belonging to the same ring and illuminated by the same antenna beam (a typical number of pulses in the beam width is 15).

Another quantity ϕ_n^m may also be associated, in ground-based radars, with the (n,m)-th resolution cell, that gives information about the pertinent "groud cluttrer". The MxN clutter information ("clutter-map") can be stored in a dedicated memory ("clutter-map memory"); due to its stationary nature the clutter is evaluated when the radar is set up. Often, several adjacent resolution-cells have associated equal clutter information, so that the clutter-map memory has generally a reduced capacity with respect to MxN. Also the

[1] T_{min} is the minimum element of $\{T_1,T_2,\ldots,T_\psi\}$.

[2] $|a|_b$ denotes "a modulo b".

clutter information can be arranged as a time series of vectors:

$$\ldots, C^{i-2}, C^{i-1}, C^{i}, \ldots \tag{3}$$

where

$$C^p = \{ c_n^p \mid c_n^p = \phi_n^{|p|_M}, n=0,1,\ldots,N-1 \},$$

$$p=\ldots,i-2,i-1,i,\ldots$$

Then, the information to be processed consists of the two time series (2) and (3). Moreover, there are some other quantities (filtering weights and thresholds) that are equal for all the resolution-cells. The way such information is processed will be seen in the next section. The result of such processing consists of one or more time series of output vectors

$$\begin{array}{l} \ldots, Z1^{i-2}, Z1^{i-1}, Z1^{i}, \ldots \\ \quad\quad\quad\quad \vdots \\ \ldots, ZL^{i-2}, ZL^{i-1}, ZL^{i}, \ldots \end{array} \tag{4}$$

Typical information carried out by the output vectors is the presence or absence of targets and the estimation of some target parameters.

3. RADAR DATA PROCESSING

The overall computation performed on radar data is the same for all the resolution-cells.

Let us consider the time instant t_p+T at which the two p-th input vectors X^p and C^p are available. In the time interval $(t_p+T,\ t_p+T+Tmin)$ the computation produces an intermediate sequence of vectors

$$\{ Y[1]^p,\ Y[2]^p,\ \ldots.,\ Y[H]^p \} \tag{5}$$

some of which constitute the p-th vectors of the output time series (4).

The processing algorithms mainly consist of linear non-recursive filters which generally involve for every resolution-cell, quantities relative to a set of (Q+1)(R+S+1) neighbouring resolution-cells. That is, if a vectors $Y[k]^p$ is produced by a filtering operation, its n-th element is given by

$$y[k]_n^p = \sum_{q=0}^{Q} \sum_{s=-R}^{S} w_{q,s}\ y[h]_{n-s}^{p-q},\quad h < k$$

where $\omega[\delta]_\beta^\alpha$ is the β-th element of vector $\Omega[\delta]^\alpha$, and $w_{q,s}$ is the (q,s)-th filter weight. The filter weights do not depend on p and n, i.e. on the considered resolution-cell.

If R=S=0, the resulting filter is one-dimensional, and is called *azimuth filter*. An example is the Moving Target Indicator (MTI) filter (for this filter Q≃1-4).

If Q=0, the resulting filter is one-dimensional and is called *range filter*. An example is the Matched filter for Pulse Compression (for this filter R=S≃3-6).

If $w_{q,s}$ does not depend on q and s, the two-dimensional filter is the equivalent of an integration or of an average. An example is the Constant False Alarm Rate (CFAR) algorithm.

Note that the quantities $y[h]_{n-s}^{p-q}$ involved in a linear digital filter may be real numbers or complex numbers. On the contrary, the filtering weights are often real numbers and frequently their values are 1 or -1 (single canceller MTI, matched filter for Barker code, etc.).

In addition to the filtering operations, other computations perform comparisons with fixed or calculated thresholds, binary or multilevel quantizations, evaluation of the module of a complex number, and so on. Such computations involve, for every resolution-cell, quantities relative to the same resolution-cell.

Referring to the number of bits employed to represent data, we observe that the required precision is not constant during the overall computation. The dynamic range of the samples supplied by analog-to-digital converter is usually covered by 6-12 bits. The elements of the intermediate vectors of the sequence (5) normally require a decreasing number of bits to be represented, as the processed information tends to become binary (presence or absence of target). In every case 16 bits are deemed an upper bound to represent all quantities involved in the computation.

The computation required for a typical search radar consists of pulse compression, MTI filter, coherent limiter, CFAR algorithm, binary quantization and pulse integration. The involved operations are mainly additions / subtractions and multiplications, and their numbers are, respectively, in the order of 100 and of 10 for each resolution-cell.

In order to perform the radar data processing, we propose to use an array of N Processing Elements (PE's) working in parallel, one PE for each range-coordinate. The n-th processing element PE_n, n=0,1,...,N-1, starting from the input quantities x_n^p and c_n^p produces in the time interval $(t_p+T,\ t_p+T+Tmin)$, p=...,i-2,i-1,i,... the sequence of elements

$$\{ y[1]_n^p,\ y[2]_n^p,\ \ldots,\ y[H]_n^p \}. \tag{6}$$

In such a way, the input time series (2) and (3) are transformed into the pertinent output time series (4).

In the processing algorithms it is infrequent to find data dependent decisions, in the sense that it rarely happens that different operations must be performed for different range-coordinate, depeding on tests made on previous local results. Whenever this situation occurs (Fig. 3a), programs implementing the processing algorithms can be structured as illustrated in Fig. 3b. That is, a test on a data dependent conditional expression $\mathcal{E}$, followed by two blocks of operations A and

Π on the two branches, can be restructured as a test on $\mathcal{E}$ with operations Λ on the TRUE branch and a null operation on the FALSE branch, followed by a test on the complement $\bar{\mathcal{E}}$ of $\mathcal{E}$, with operations Π on the TRUE branch and a null operation on the FALSE branch. In such a way the case in which different PE's must perform simultaneously different operations is avoided. Instead, the PE's for which a test results FALSE must be blocked, while all the others execute the same operation. This is easy to be mechanized in an array system, as explained in the following sections.

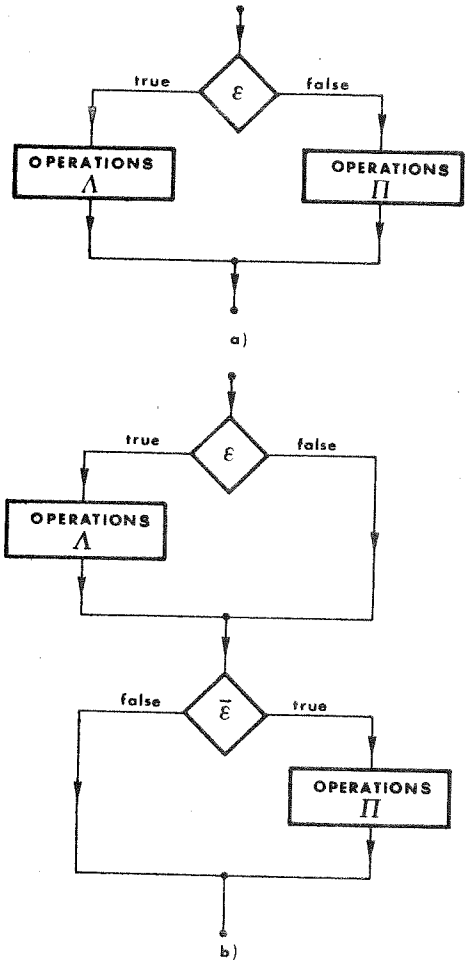

Fig. 3 Flow-chart rearrangement.

4. SYSTEM ORGANIZATION

The overall structure of the proposed digital system for radar signal processing is given in Fig. 4a.

The master computer (MC) stores the program implementing the processing algorithms and those quantities that are equal for all the resolution-cells (as the filtering weights). Moreover, it fetches the instructions from memory and executes directly those instructions that control the program flow. The execution of the arithmetic and logic instructions that operate on radar data is delegated to the slave computer (SC).

Since the program flow does not depend on the results of the computations performed by the SC, the MC and the SC behave as producer-consumer, and information is sent from the MC to the SC through a asynchronous FIFO buffer (AFB). Each word of the AFB stores all the quantities needed by the SC for executing instructions, such as the operation code, the addresses of the operands stored in the SC and the operands stored in the MC.

The input buffer is constituted by N registers. In the time interval (t_p, t_p+T) it is loaded with the elements of X^P and C^P (Fig. 4b), a pair of elements (x_n^P, c_n^P) being inserted by means of a shift down operation for every time interval T_c. At the time instant t_p+T the input buffer stops loading new input data, and in a subsequent fraction δI of the dead time Δ_p it is unloaded by the slave computer. All the N registers are unloaded in parallel, each register serially bit by bit.

The slave computer is constituted by an array of N processing elements (PE's) driven by the same control unit. The n-th PE, in the interval of time δI, where $\delta I < \Delta_p min$, takes from the n-th register of the input buffer the quantities x_n^P and c_n^P serially bit by bit. In the time interval $(t_p+T+\delta I, t_p+T+\delta I+Te)$, where Te is the effective computation time, the n-th PE performs the required computation, according to the instructions supplied by the AFB, and evaluates the sequence of elements (6). In the following time interval δO, the n-th PE loads serially bit by bit the n-th register of the output buffer with the quantities $z1_n^P, z2_n^P, \ldots, zL_n^P$ representing the n-th elements of the output vectors. The constant time intervals δI, Te and δO must satisfy the condition $\delta I+Te+\delta O=Tmin$.

The output buffer, similarly to the input buffer, is constituted by N registers. In the time interval $(t_p+T+\delta I+Te, t_p+T+\delta I+Te+\delta O)$ it is loaded by the slave computer, the N registers being loaded in parallel, each register serially bit by bit. In the following time interval T the output buffer is unloaded, a set of elements $\{z1_n^P, z2_n^P, \ldots, zL_n^P\}$ being extracted by means of a shift down operation for every time interval T_c.

Since δI and δO are of the order of a few μsec, it follows that each PE must perform the overall computation relatively to a resolution-cell in a time interval nearly equal to Tmin (about 1000 μsec). This time is sufficient to execute the required operations even if a serial arithmetic

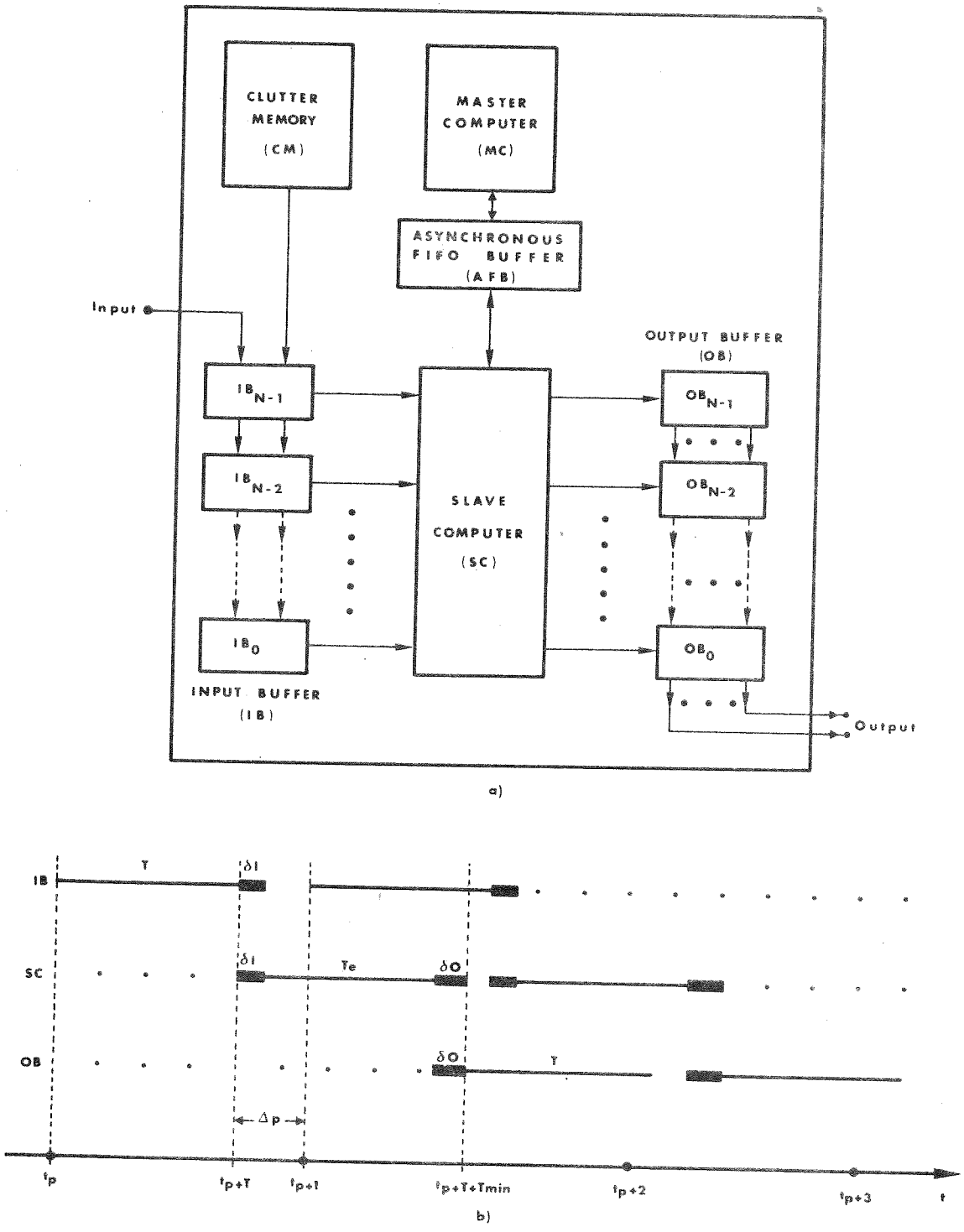

Fig. 4 a) Organization of the SMA; b) Timing diagram of the SMA.

is implemented.

5. STRUCTURE OF THE MASTER COMPUTER AND OF THE SLAVE COMPUTER

As seen in the previous section, the master computer has the task of fetching instructions and executing those instructions that control the program flow, and its structure resembles that of a very simple minicomputer. The main memory, storing the program and those data that do not depend on the specific resolution-cell, has a capacity of 1024 words, each having a 16 bit lenght. The Processing Unit has eight 16-bit general registers R0,R1,...,R7, and very limited arithmetic and logic capabilities (addition, subtraction, shift).

The program is written as a pure procedure and implements the radar algorithms relatively to one resolution-cell of the first sector. The slave computer performs the specified operation in parallel for all the resolution cells of the sector. The MC registers are normally assigned the following tasks: register R1 is used as index register; register R2 holds a constant operand (i.e. an operand which has the same value for all resolution-cells); registers R3 and R4 hold the lengths of operands.

In order to utilize the same program successively for the other sectors, a dynamic address modification that uses R0 as a base register is implemented, and the contents of R0 is properly incremented at the beginning of every pulse repetition interval.

The slave computer (Fig. 5) executes the arithmetic and logic instructions sent by the master computer via the asynchronous buffer. The specified operations are executed serially, bit by bit, and the length of operands is variable from 1 to 16 bits. A microprogrammed control unit interprets the instructions and sends proper microorders in parallel to an array of N processing elements (PE's). The microprograms, due to the array organization, are not conditioned by variables coming from the PE's.

The n-th processing element PE_n is constituted by a high speed memory M_n and by a serial processing unit PU_n.

The memory M_n has a maximum capacity of 2048x1 bits (the memory size is related to the overall memory of azimuth filtering). It is seen by the program instructions as organized in words of 16 bit length, up to a maximum of 128 words. The leftmost L bits of a word (L=1-16) store an L-bit operand. When PE_n is working on the (n,m)-th resolution-cell, M_n contains the last input information relative to that resolution-cell and some intermediate results relative to the computations previously performed on the resolution-cells belonging to the same ring (such quantities are needed for azimuth filtering). The intermediate results relative to the computations performed on the resolution-cells belonging to the same sector (such quantities are needed for range filtering) are taken from the memories of the neighbouring PE's. More precisely, PE_n can operate directly on an operand stored in M_{n-1} or in M_{n+1}. Moreover, a quantity can be moved from M_{n-3} (M_{n+3}) to M_n, so that M_{n-1} and M_{n+1} contain new operands coming from M_{n-2} and M_{n-4} (M_{n+2} and M_{n+4}) on which PE_n can directly operate, and so on.

The processing unit PU_n is constituted by a one-bit full adder / subtractor, and by a small amount of local control logic that allows all the PE's to execute multiplication and division operations (whose single steps depend on the local operand values) under the control of the same microorder sequence. A fractional two's-complement serial arithmetic is implemented.

Moreover, PE_n is provided by one 1-bit tag register TR_n, that enables (TR_n=1) or disables (TR_n=0) PE_n to operate. TR_n is set or reset according to the result of proper test instruction on quantities stored in the pertinent PE memory. TR_n controls the PE_n clock, so that an instruction of the program can be executed either unconditionally by all the PE's, or conditionally only by the enabled PE's, depending on the contents of the T field of the instruction.

The overall hardware of PE_n consists, besides the memory, of 25 gates and 7 one-bit registers. It is well suited to be integrated as a serial microprocessor.

6. INSTRUCTIONS AND FIFO BUFFER WORDS

The set of program instructions can be partitioned in two classes:
- instructions that control the program flow, which are executed by the master computer;
- instructions that perform arithmetic and logic operations on radar signals, that are fetched by the master computer and executed simultaneously by the PE's of the slave computer.

The instructions of the first class are similar to those of a typical minicomputer, with indexing facility for the control of program loops.

The instructions of the second class are less conventional, have a 32 bit format, and can be partitioned in arithmetic instructions and logical instructions.

The arithmetic instructions have the format shown in Fig. 6a, where OP represents the operation code and (Xi, Yi), i=1,2,3 specifies the logical address Li of one operand stored in the PE memories, each memory thought of as arranged in 16 bit words. The logical address Li is given by Yi, if Xi=0, and is obtained by adding Yi to the contents of register R1 (index register) of the master computer, if Xi=1. The physical address F_i of an operand is obtained by adding Li to the contents of Register R0 (base register) of the master computer. Register R0 is incremented at every time instant $\ldots,t_{i-2},t_{i-1},t_i,\ldots$ so that the same program can elaborate successively radar data relative to consecutive sectors. The T bit

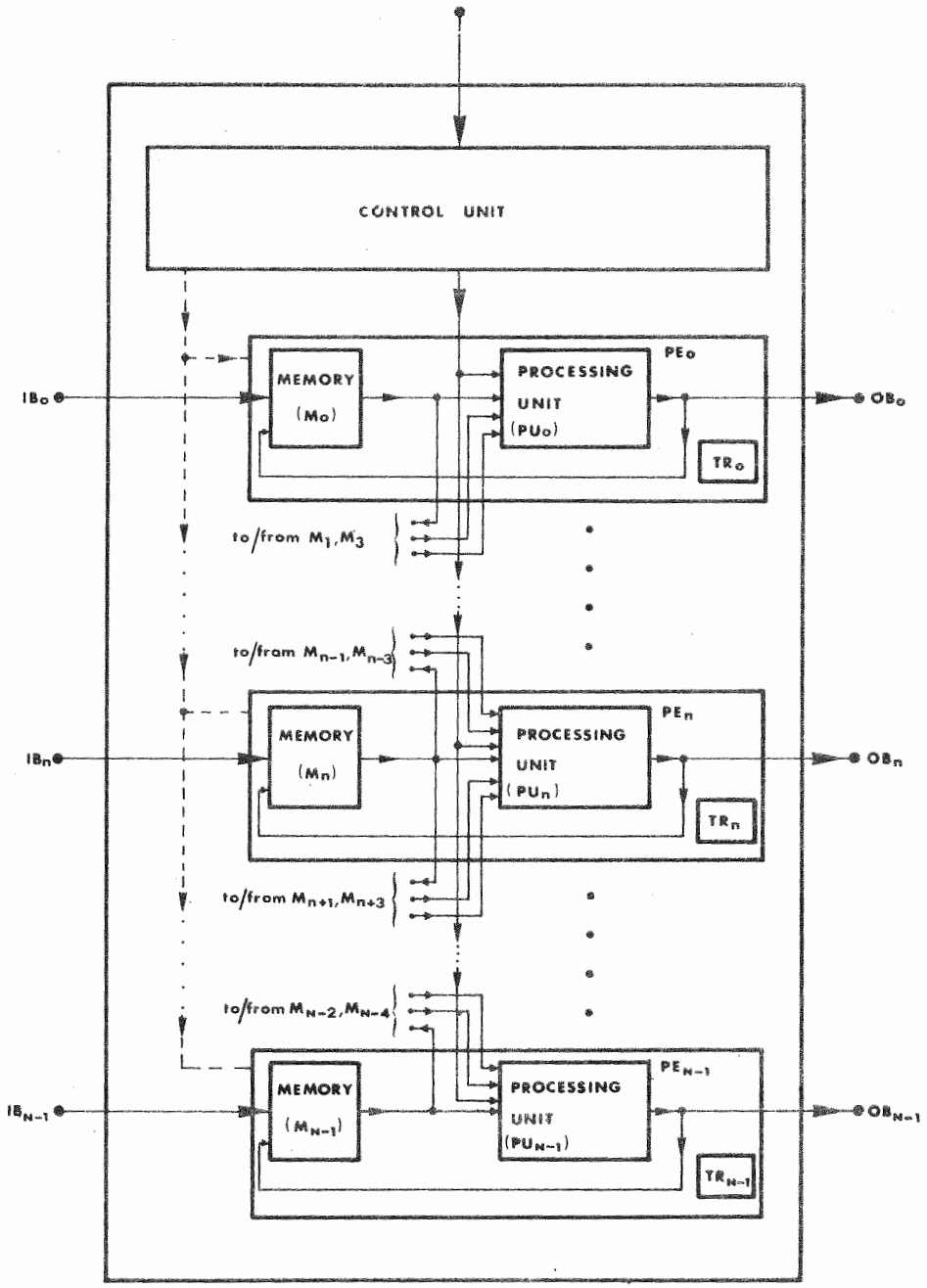

Fig. 5 Structure of the Slave Computer.

specifies if the instruction is to be executed in parallel either by all the PE's (T=1) or only by those PE's for which the tag is set (T=0). For some instructions, the (X2, Y2) field is not

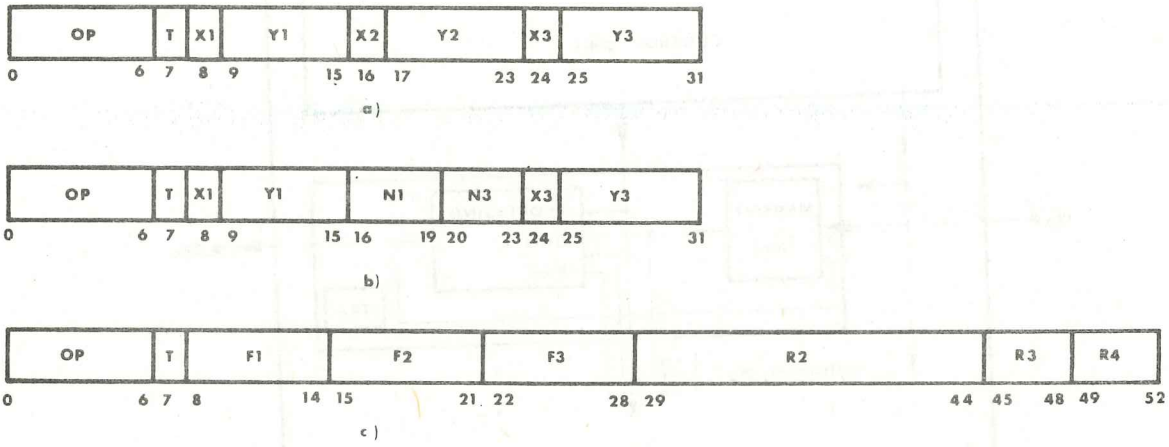

Fig. 6 a) Arithmetic instruction format; b) Logic instruction format; c) AFB word format.

specified, and the second operand is constant for all the PE's and is constituted by the contents of register R2 of the master computer. The operands have a variable length, the actual lengths being specified by the contents of registers R3 and R4 of the master computer. Due to the serial arithmetic, the execution time of an instruction depends on the operand lengths so that the variable length operand facility saves time when the operands have a length less than 16 bits.

The following are typical arithmetic instructions. The notation $M_\alpha(Fi)_{R\beta}$ indicates the leftmost γ bits of the contents of the Fi-th word of M_α, where γ is the contents of register R_β of the master computer. Similarly, $R2_{R\beta}$ indicates the leftmost γ bits of the contents of register R2 of the master computer.

AD (add)
$M_n(F1)_{R3}$ added to $M_n(F2)_{R3}$ replaces $M_n(F3)_{R3}$;

ADU1 (add 1st upper)
$M_{n+1}(F1)_{R3}$ added to $M_n(F2)_{R3}$ replaces $M_n(F3)_{R3}$;

MADU3 (move and add 3rd upper)
$M_{n+3}(F1)_{R3}$ replaces $M_n(F3)_{R3}$, and $M_{n+3}(F1)_{R3}$ added to $M_n(F2)_{R3}$ replaces $M_n(F2)_{R3}$;

ADD1 (add 1st down)
$M_{n-1}(F1)_{R3}$ added to $M_n(F2)_{R3}$ replaces $M_n(F3)_{R3}$;

MADD3 (move and add 3rd down)
$M_{n-3}(F1)_{R3}$ replaces $M_n(F3)_{R3}$, and $M_{n-3}(F1)_{R3}$ added to $M_n(F2)_{R3}$ replaces $M_n(F2)_{R3}$;

ADC (add constant)
$M_n(F1)_{R3}$ added to $R2_{R3}$ replaces $M_n(F3)_{R3}$;

MUL (multiply)
$M_n(F1)_{R3}$ multiplied by $M_n(F2)_{R4}$ replaces $M_n(F3)_{R3}$;

MULC (multiply by constant)
$M_n(F1)_{R3}$ multiplied by $R2_{R4}$ replaces $M_n(F3)_{R3}$.

Other arithmetic instructions involve subtraction, division, absolute value and logarithm.

The logical instructions have the format shown in Fig. 6b, where OP, T, (X1, Y1) and (X3, Y3) have the same meaning as in Fig. 6a, and N1 and N3 have a meaning depending on the specific instruction.

The following are typical logical instructions (the notation $M_\alpha(F_i,Ni)$ means the Ni-th bit, from the left, of the contents of the F_i-th word of M_α).

SHR (shift right)
$M_n(F1)_{R3}$ shifted arithmetically N1 times replaces $M_n(F3)_{R3}$;

TQ (test and quantize)
if $M_n(F1)_{R3} \geq M_n(F3)_{R3}$ then 1 replaces $M_n(F3,N3)$ otherwise 0 replaces $M_n(F3,N3)$;

TCQ (test against constant and quantize)
if $M_n(F1)_{R3} \geq R2_{R3}$ then 1 replaces $M_n(F3,N3)$, otherwise 0 replaces $M_n(F3,N3)$;

TST (test and set tag)
if $M_n(F1)_{R3} \geq M_n(F3)_{R3}$ then 1 is stored in TR_n, otherwise 0 is stored in TR_n;

TCST (test against constant and set tag)
if $M_n(F1)_{R3} \geq R2_{R3}$ then 1 is stored in TR_n, otherwise 0 is stored in TR_n;

ANDB (AND bit)
$M_n(F1,N1) \wedge M_n(F3,N3)$ replaces $M_n(F3,N3)$;

ORB (OR bit)
$M_n(F1,N1) \vee M_n(F3,N3)$ replaces $M_n(F3,N3)$.

Other logical instructions involve left arithmetic shift, and different types of tests.

Two input/output instructions for unloading the input buffer and for loading the output buffer are provided. Moreover, a WAIT instruction is used, for synchronizing the start of the program with the time instants $\ldots, t_{i-2}+T, t_{i-1}+T, t_i+T, \ldots$.

The master computer, recognizing that a fetched instruction must be executed by the slave computer, evaluates the physical addresses of the operands in the PE memories. Then it sends to the slave computer, via the asynchronous FIFO buffer, all the information needed for instruction execution, i.e., the operation code, the T bit, the physical addresses of the operands (the F2 address can be replaced by the integers N1 and N2 for the logical instructions), and, possibly, the contents of register R2 (constant operand) and the rightmost four bits of registers R3 and R4 (operand lengths).

Then a word of AFB has the format shown in Fig. 6c.

The slave computer takes from AFB the contents of one word at a time, and executes the pertinent instruction. When it encounters the WAIT instruction, it waits for a signal informing that the input buffer is full.

The most used instructions for radar signal processing are additions and multiplications by constant. For ten-bit operands such operations are performed by each PE respectively in 35 and 150 cycles. The small number of cycles involved in the multiplication by constant is due to the use of a uniform-shift-of-two multiplication algorithm [6].

CONCLUSIONS

The Serial Microprocessors Array has been described, designed for processing high-bandwidth raw radar data. The intended application is search radar, as in air traffic control installations. The main video signal processing algorithms of a typical search radar require some 100 additions and 10 multiplications for every resolution-cell. Assuming a data word length of 10 bits, an AD instruction is executed in 35 cycles and a MULC instruction in 150 cycles, so that 5000 cycles are needed. 20% of cycles for miscellaneous instructions is to be added. Therefore, in order to process data relative to one resolution-cell, and then (due to the array structure of the SC) data relative to one sector, 6000 cycles are required. Since 1000 μsec are typically available for this computation, it follows that a cycle time of 100-150 nsec for the SC is well suited for the required throughput.

REFERENCES

[1] B.P. Shay, "Design Considerations of a Programmable Predetection Digital Signal Processor of Radar Application", Information Systems Group, Naval Research Laboratory, NRL Report 7455, December, 1972.

[2] D.L. Slotnick, W.C. Borck, R.C. McReynolds, "The SOLOMON Computer", Proc. FJCC, 1962, pp. 97-107.

[3] G.H. Barnes, R.M. Brown, M. Kato, D.J. Kuck, D.L. Slotnick, R.A. Stokes: "The Illiac IV Computer", IEEE Trans. on Comp., Vol. C-17, n° 8, Aug. 1968, pp. 746-757.

[4] J.A. Rudolph, "A Production Implementation of an Associative Array Processor-STARAN", Proc. FJCC, 1972, pp. 229-241.

[5] G.R. Couranz, M.S. Gerhardt, and C.J. Young, "Programmable Radar Signal Processing Using the RAP", Sagamore Computer Conference on Parallel Processing, Springer-Verlag, Berlin, 20-24 August, 1974, pp. 37-52.

[6] L.P. Rubinfield, "A Proof of the Modified Booth's Algorithm for Multiplication", IEEE Transactions on Computers, Vol. C-24, October 1975, pp. 1014-1015.