A. Ast, J. Becker, R. Hartenstein, et al.: MoPL-3: A New High Level Xputer Programming Language; 3rd Int. Workshop On Field Programmable Logic And Applications, Oxford, 7.-10. September 1993

MoPL-3: A NEW HIGH LEVEL XPUTER PROGRAMMING LANGUAGE

A. Ast, J. Becker, R. Hartenstein, R. Kress, H. Reinig, K. Schmidt

Fachbereich Informatik, Universität Kaiserslautern
Postfach 3049, W-67653 Kaiserslautern, Germany
fax: (+49 631) 205-2640, e-mail: hartenst@rhrk.uni-kl.de

ABSTRACT

This paper illustrates a new high level programming language which is important for a novel class of computational devices called Xputers, which are by up to several orders of magnitude more efficient than the von Neumann paradigm of computers. Xputers are as flexible and as universal as computers. The flexibility of Xputers is achieved by using field-programmable logic (interconnect-reprogrammable media) as the essential technology platform. The paper first briefly illustrates the Xputer paradigm as a prerequisite needed to understand the fundamental issues of this new language.

1. INTRODUCTION

The next section summarizes a new machine paradigm based on field-programmable logic (see [1], [4], [5], [7]). This new paradigm supports highly flexible reconfigurable ALUs. Such a flexibility is not feasible within the von Neumann paradigm, because of its tight coupling between (instruction) sequencer and ALU. Thats´s why to obtain a similiar familiarity and universality as known from the von Neumann paradigm, another sequencing mechanism is needed. That´s why the Xputer uses a data counter (instead of an instruction counter). This leads to a new procedural paradigm, which is made accessable by a new kind of programming language: data-procedural languages. We introduce such a language (MoPL-3), which has been developped at Kaiserslautern University. This language is a C extension including primitives for data sequencing and hardware reconfiguration.

2. SUMMARIZING THE XPUTER

For convenience of the reader this section summarizes the underlying machine paradigm having been published elsewhere ([1], [4], [5], [7]). Main stream high level control-procedural programming and compilation techniques are heavily influenced by the underlying von Neumann machine paradigm. Most programmers with more or less awareness need a von-Neumann-like abstract machine model as a guideline to derive executable notations from algorithms, and, to understand compilation issues. Also programming and compilation techniques for Xputers need such an underlying model, which, however, is a data-procedural machine paradigm, also called data sequencing paradigm.

This section summarizes and illustrates the basic machine principles of the Xputer paradigm [8]. Simple algorithm examples will illustrate MoPL-3, a data-procedural programming language.

2.1 Xputer Machine Principles

The main difference to von Neumann is, that Xputers have a data counter instead of a program counter. Figure 1b illustrates this by means of the MoM-2 xputer architecture example. The key difference to computers is, that data sequencer and a reconfigurable ALU replace computers' program store, instruction sequencer

and the hardwired ALU (this view is simplified). For operator selection instead of the sequencer another unit is used, which we call residual control. Operator activation is transport-triggered (in contrast to control flow triggered activation in von Neumann computers).

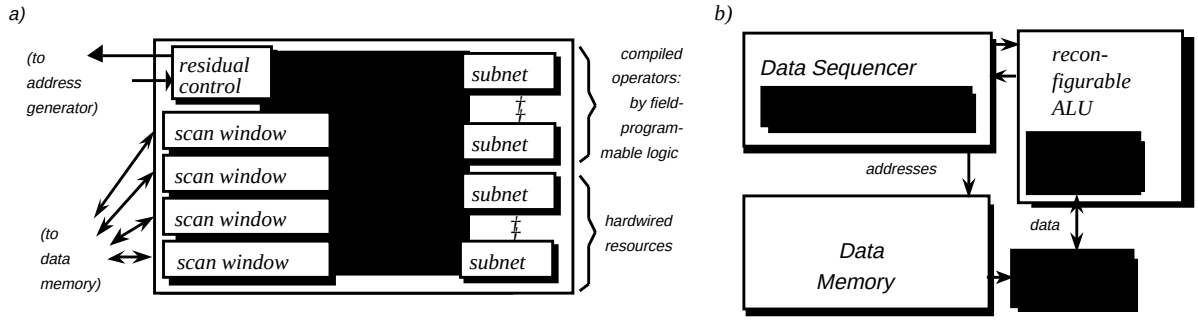


Figure 1. Basic structures of Xputers and the MoM architecture: a) reconfigurable ALU (rALU) of the MoM, b) basic structure of Xputers

Smart Register File. Due to their higher flexibility (in contrast to computers) Xputers may have completely different processor-to-memory interfaces which efficiently support the exploitation of parallelism within the rALU. Such an interface we call a scan window. It implements a hardwired window to a number of adjacent locations in the memory space. Its size is adjustable at run time. Such a scan window may be 'placed' onto a particular location in memory under control of a data sequencer.

The Data Sequencer. The hardwired data sequencer features a rich and flexible repertory of scan patterns for moving scan windows along scan paths within memory space. Address sequences needed are generated by hardwired address generators having a powerful repertory of generic address sequences [2, 11]. After having received a Scan Pattern code a data sequencer runs in parallel to the rest of the hardware without stealing memory cycles. This accelerates xputer operation, since it avoids performance degradation by addressing overhead.

Reconfigurable ALU. Xputers have a reconfigurable ALU (rALU), partly using the technology of field-programmable logic. Figure 1a shows an example: the rALU of the MoM4 Xputer architecture. The four smart register files called scan windows are explained later. The MoM-4 rALU has a repertory of hardwired operator subnets. Within the field-programmable part of the rALU additional operators needed for a particular application may be compiled by logic synthesis techniques. A global interconnect-programmable structure is the basis of connecting these operators to form one or more problem-specific compound operators, what will be illustrated later by a simple algorithm implementation example.

rALU Configuration is no Microprogramming. Also microprogrammable von Neumann processors have a kind of reconfigurable ALU which, however, is highly bus-oriented. Buses are a major source of overhead [7], especially in microprogram execution, where buses reach extremely high switching rates at run time. The intension of rALU use in Xputers, however, is compound operator configuration at compile time (downloaded at loading time) as much as possible, so that path switching activities at are minimized and the underlying organizational overhead is pushed into compile time to save the much more precious run time.

Compound Operators. The MoM Xputer may execute expressions (which we call ´compound operators´) within a single machine step, whereas computers can execute only a single operation at a time. The rALU may be configured such a way, that one or more sets of parallel data paths form powerful compound operators which need only a single basic clock cycle to be executed. This rALU uses no fixed instruction set: compound operators are user-defined. Since their combinational machine code is loaded directly into the rALU, Xputers do not have a program store nor an instruction sequencer. Instead a data sequencer is used which steps through the data memory to access the operands via register files called data scan windows. Xputers operate data-driven but unlike data flow machines, they feature deterministic principles of operation called data sequencing.

Summary of Xputer Principles. The fundamental operational principles of Xputers are based on data auto sequencing mechanisms with only sparse control, so that Xputers are deterministically data-driven (in contrast to data flow machines, which are indeterministically data-driven by arbitration and thus are not debuggable). Xputer hardware supports some fine granularity parallelism (parallelism below instruction set level: at data path or gate level) in such a way that internal communication mechanisms are more simple than known from parallel computer systems. (For more details about Xputer see [4, 5])

3. A PROGRAMMING LANGUAGE FOR XPUTERS

This section introduces a high level Xputer programming language MoPL-3 (Map-oriented Programming Language) which is easy enough to learn, but which also is sufficiently powerful to explicitly exploit the hardware resources the Xputer offers. For an earlier version of this language we have developed a compiler [14]. MoPL-3 is a C extension, including primitives for data sequencing and hardware reconfiguration.

3.1 MoPL-3: A Data-procedural Programming Language

This section introduces the essential parts of the language MoPL-3 and illustrates its semantics by means of two program text examples (see below): the constant geometry FFT algorithm, and the data sequencing part for the JPEG Zig-Zag scan being part of a proposed picture data compression standard. MoPL-3 is an improved version of MoPL-2 having been implemented at Kaiserslautern as a syntax-directed editor [14].

The main extension issue to other programming languages is the data location or data state such, that we simultaneously have two different kinds of location or state. There is the familiar von-Neumann-type control state (current location of control), which e. g. is handled by goto statements referencing control label locations within the program text, or, by other control statements. During execution of Xputer programs such a control state is coexisting with one or more data location states, what will be illustrated subsequently (The control flow notation does not model the underlying Xputer hardware very well, since it has been adopted from C for compatibility reasons to minimize programmer training efforts). A data state corresponds to the actual position of a data counter, which contains a data address. The number of data states depends on the actual hardware. The data sequencer of the Xputer generates the corresponding sequences of data states. The purpose of this extension is the easy programming of sequences of data addresses (Scan Pattern), including code generation for the data sequencer. The familiar notation of these MoPL-3 constructs is easy to learn by the programmer.

JPEG Zig-Zag scan example. The MoPL-3 program code from Figure 4 illustrates programming the JPEG Zig-Zag scan pattern (Figure 2, for more details about the JPEG Compression Standard see also [9], [10], [13]) named JPEGzigzagScan for scanning the array PixMap declared in line (38).

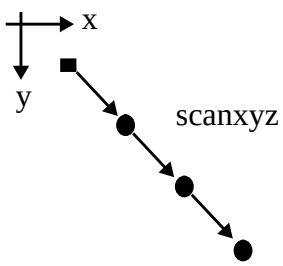
This example uses a single 1-by-1 scan window (adjusted as a single word buffer), which illustrates, that the performance benefit by the address generator can be obtained also for accessing long sequences of single memory locations. Lines (28) thru (30) declare four Scan Patterns (see also Figure 2), where the declaration statements have the form:

<name_of_scan_pattern> **is** <maximum_length_of_loop> **STEPs** <step_vector>.

Example:

ScanPattern

scanxyz is 3 steps [1,1];



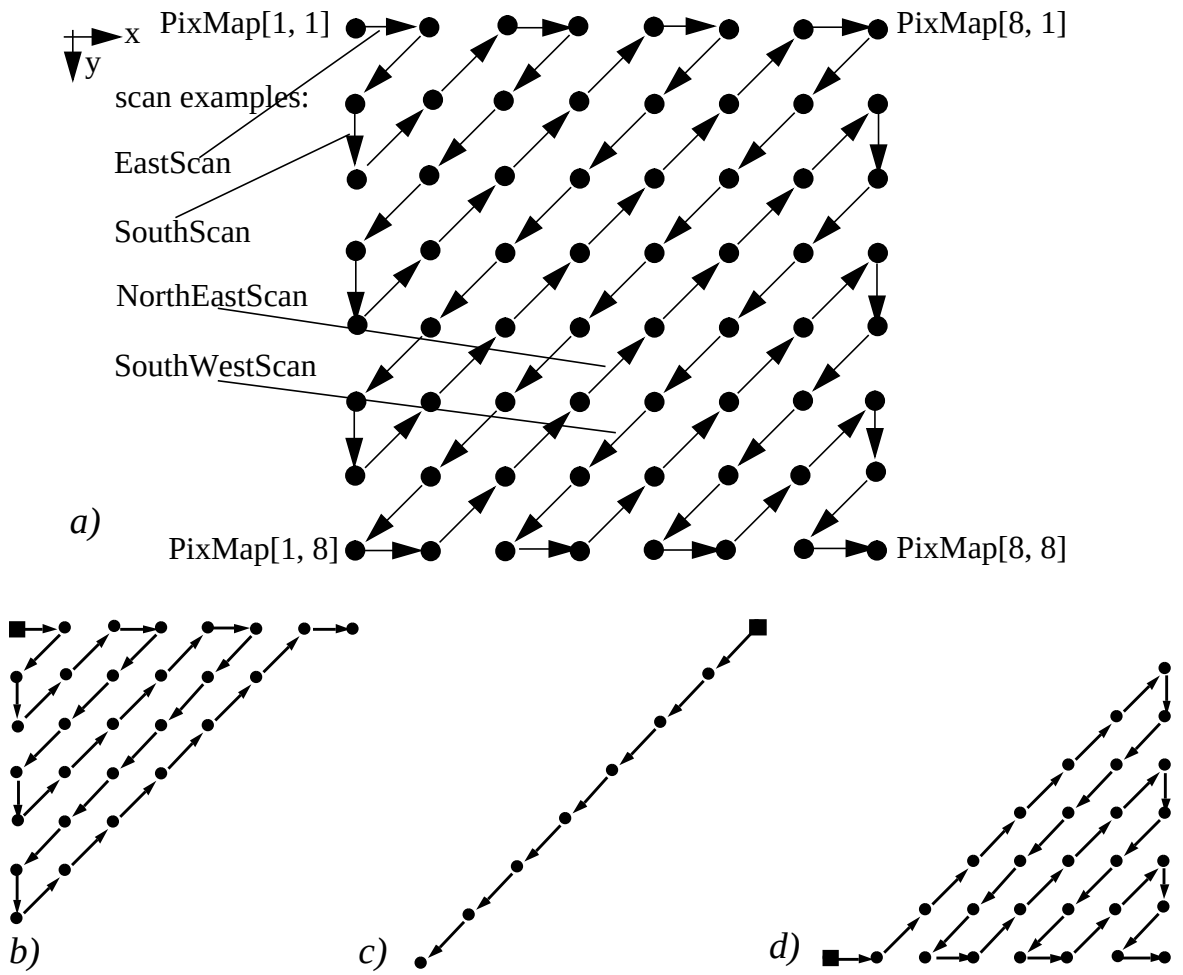


Figure 2. JPEG Zig-Zag Scan Pattern scanning an array PixMap [1:8,1:8] (a) and its subpatterns: b) upper left triangle UpLzigzagScan, d) lower right LoRzigzagScan, c) full SouthWestScan

These statements are a kind of "Transport-Instructions" (transportation of data), which realize in every step of the Scan Pattern a "read-modify-write cycle". The step vector specifies the next data location relative to the current data location (before executing a step of the scan sequence). In the below MoPL-3 program (see Figure 4) the data state manipulations (moving scan windows with data space) in Figure 3 are needed:

The 4 declared Scan Patterns (see Figure 3), which are needed for the JPEG Zig-Zag-Scan, are called in the two **`while`** loops at lines (46) thru (51) and at lines (56) thru (61) in Figure 4, e.g. see line (47), where the Scan Pattern EastScan is called (its analogous to a procedure call in C). By an escape a scan may also be terminated before <maximum_length_of_loop> is reached. In this case there will be an escape from Scan Pattern, when the boundary of the data map is reached or exceeded. E. g. see the **`until`** clause (escape clause) in line (48) indicating an escape on reaching a leftmost word within the PixMap array (see Figure 2: the first execution of SouthWestScan at top left corner of the array reaches only a loop length of 1). The condition @[1,] says: escape if within current array a data location with an x subscript 1 is reached. The empty position behind the comma says: ignore the y subscript.

Before the execution of the first Scan Pattern, you have to specify the starting point in the data map. For this purposes we use another data state manipulation, the **`moveto`** statement. With this statement you are able to realize absolute jumps of the scan window inside the data map. E.g. see line (65) in Figure 4, where the scan window is moved to the upper left corner of the PixMap**.**

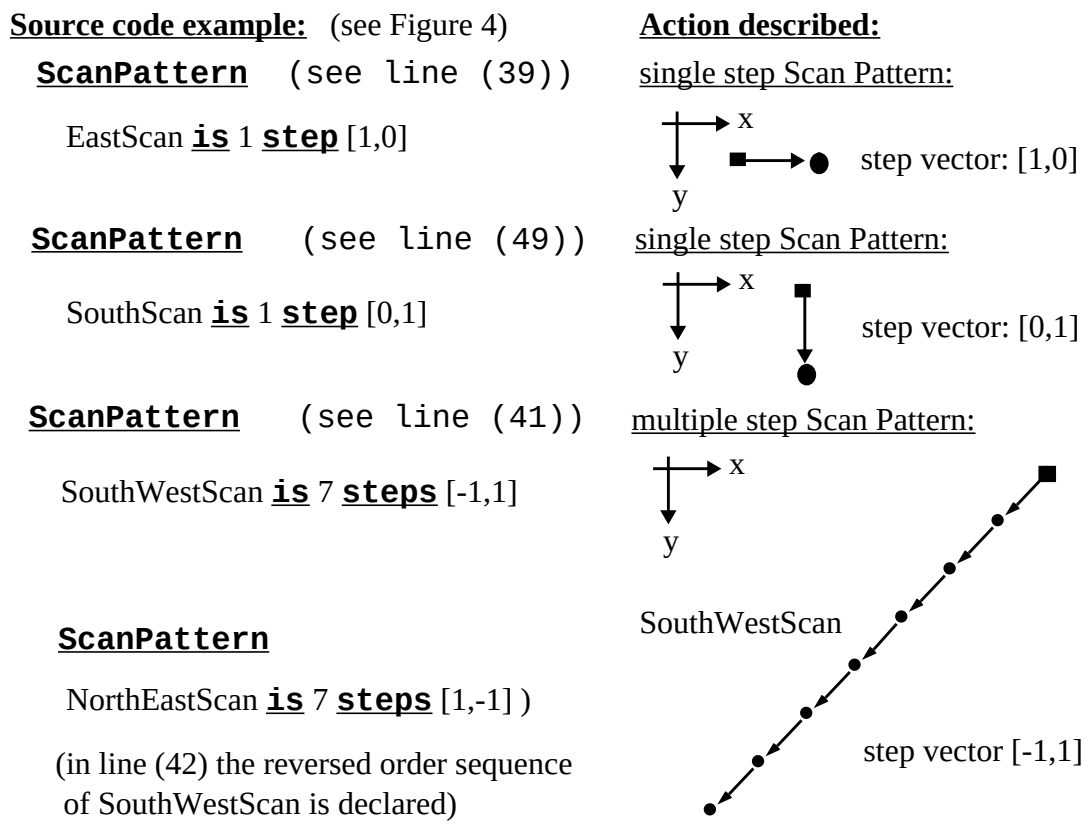


Figure 3. Scan Pattern declared for the JPEG example (see also

Hardware-supported Escapes. To avoid overhead for efficiency the **until** clauses are directly supported by MoM hardware features of escape execution [4]. To support the **until** @ clauses by off-limits escape the address generator provides for each dimension (x, y) two comparators, an upper limit register and a lower limit register.

Table 1: Scan Patterns Transformation Functions

Transformation Function	Corresponding Operation
rotl	turn left
rotr	turn right
rotu	turn 180^{o}
mirx	flip x
miry	flip y
reverse	reversed order sequence

The above MoPL-3 program (see Figure 4) covers the following strategy. The first **while** loop at lines (46) thru (51) iterates the sequence of the 4 scan calls EastScan thru NorthEastScan for the upper left triangle of the JPEG scan, from PixMap[1,1] to PixMap[8,1] (see Figure2). The second **while** loop at lines (56) - (61) covers the lower right triangle from PixMap[8,1] to PixMap[8,8]. The SouthWestScan between both **while**

```
   /* assuming, that rALU configuration has been declared and set-up */
Array          PixMap [1:8,1:8,15:0];                          (38)
ScanPattern    EastScan      is  1 step  [ 1, 0],              (39)
               SouthScan     is  1 step  [ 0, 1],              (40)
               SouthWestScan is  7 steps [-1, 1],              (41)
               NorthEastScan is  7 steps [ 1,-1],              (42)
                                                               (43)
               UpLzigzagScan    is                             (44)
               begin                                           (45)
                    while (@[<8,])                             (46)
                    begin  Eastscan;                           (47)
                           SouthWestScan until @[ 1,];         (48)
                           SouthScan;                          (49)
                           NorthEastScan until @[, 1];         (50)
                    end                                        (51)
               end UpLzigzagScan ,                             (52)
                                                               (53)
                                                               (54)
               JPEGzigzagScan    is                           (55)
               begin                                           (56)
                       UpLzigzagScan;                          (57)
                       SouthWestScan;                          (58)
                       rotu (reverse(UpLzigzagScan));          (59)
               end JPEGzigzagScan ;                            (60)
          (* end of declaration part*)                        (61)
                     :                                         (62)
                     :                                         (63)
                     .
begin                    (* statement part*)                  (64)
     moveto PixMap [1,1];                                      (65)
     JPEGzigzagScan ;                                          (66)
end                                                            (67)
```

Figure 4. MoPL program of the JPEG scan pattern shown in Figure 2

loops at line (67) from PixMap[8,1] to PixMap[1,8] connects both triangular scans to obtain the total JPEG pattern. In line (59) two Scan Pattern transformation functions (**rotu**, **reverse**) are used. With these functions (see Table 1) one can easy realize new Scan Patterns by using predeclared Scan Pattern, which structure is similar to the newer ones.

Constant geometry FFT example. The second example illustrates parallelism by running several windows synchronously. It is the constant geometry Fast Fourier Transform (FFT) illustrated by Figure 7, with a data map (CGFFT) size of 9 by 16 words (Figure 7a). Figure 8 shows the MoPL-3 section, declaring the scan patterns and the rALU configuration. The declaration of the Scan Pattern starts with the keyword **ScanPattern**. ´HLScan´ is the outer loop, whereas ´SP1´ and ´SP23´ are used for inner loops running in parallel (see Figure 7c). ´SP1´ is used for scan window ´SW1´ and ´SP23´ is used for two scan windows ´SW2´ and

´SW3´. The configuration of the rALU will be specified in two parts: the adjustment declaration (window size declaration, see Figure 5) and the declaration of the rALU subnet (see Figure 6).

Source code example: see Figure 8

```
Adjustment  ThreeW  is
    SW1 [1:2, 1:2, 63:0] handle [1,1]
    SW2, SW3 [1:1, 1:1, 63:0] handle [1,1]
```

Size of the scan windows:

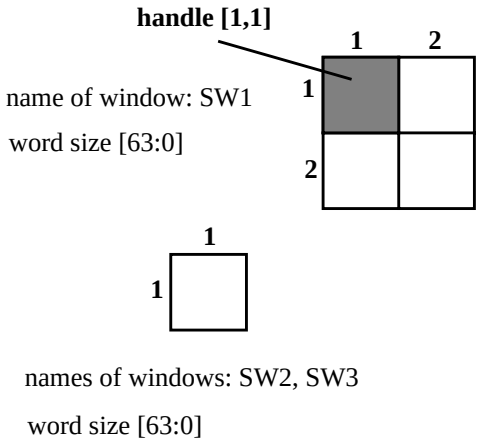


Figure 5 Adjustment declaration of the FFT example (see also Figure 8)

The declaration of the scan windows and their sizes starts with the keyword **Adjustment**. With **handle** you can specify the reference point of a window (see Figure 5). This point will be needed, when you move a scan window to a specific place in the data memory. The execution of the adjustment named ThreeW (see above: including 3 scan windows with the names SW1, SW2 and SW3) will be done by the keyword **adjust** followed by the name of the adjustment (see line (80) in Figure 9)

The declaration of the problem-specific compound operators starts with **rALUsubnet** followed by the

Source code example: see Figure 8

```
rALUsubnet   FFT   is
SW2 = SW1 [1,1] + SW1 [2,1] * SW1 [2,2],
SW3 = SW1 [1,1] - SW1 [2,1] * SW1 [2,2];
```

rALU configuration:

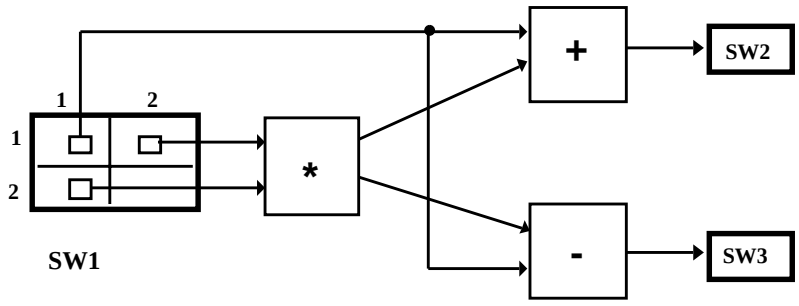


Figure 6. Declaration of the 2 compound operators of the FFT example (see

name of the rALU subnet and the keyword **is**. Thereafter the compound operators are specified (see Figure 6 and also lines (76) thru (78) in Figure 8). These operators are executed in every single step of a Scan Pattern. The calling of the rALU subnet "FFT" will be done by the keyword **apply** followed by the name of the rALU subnet (see line (81) in Figure 9).

Figure 9 shows the MoPL-3 program section, which move scan windows to proper starting points and calls

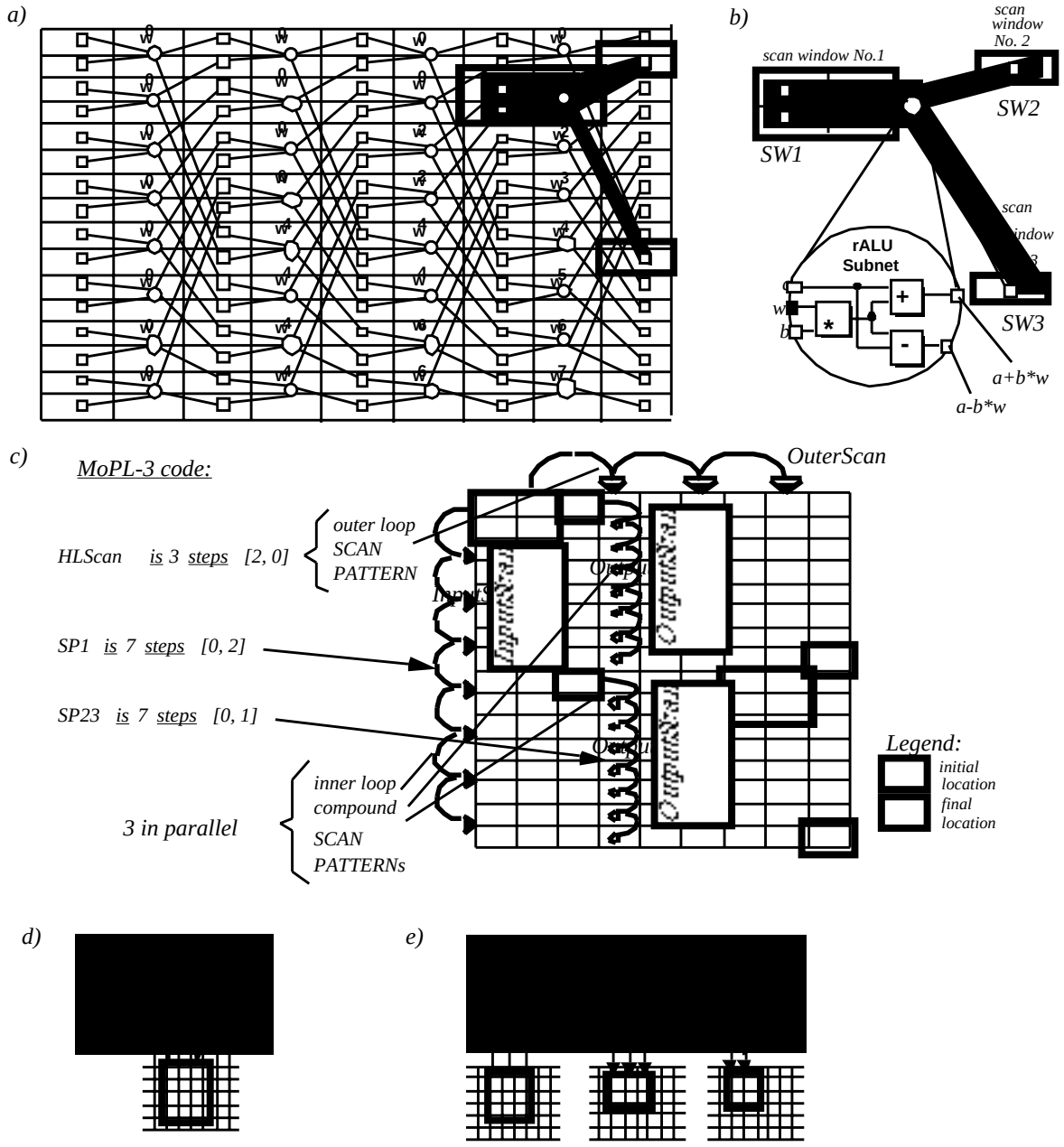


Figure 7. Constant geometry FFT algorithm 16 point example using 3 scan windows synchronously in parallel: a) signal flow graph with data map grid and a scan window location snapshot example, b) deriving rALU subnet, scan window sizes and interconnect from compound operator, c) nested Scan Pattern illustration (including their MoPL-3 declaration, see Figure 8), d) illustration of fine grain parallelism: single

the nested compound scan pattern (such as illustrated in Figure 7c). Figure 7 shows an algorithm implementation example, a 16 point constant geometry FFT, where three scan windows run in parallel. Figure 7a shows the signal flow graph and the storage scheme (the grid in the background). The 16 input data points are stored in the leftmost column.

Weights w are stored in every second column, where each second memory location is empty (for regularity reasons). Figure 7b shows the window adjustments: the 2-by-2 window no. 1 is the input window reading the operands a and b, and the weight w. Windows no. 2 and 3 are single-word result windows. Figure 7b also shows the compound operator and its interconnect to the three windows. This is an example of fine granularity parallelism, as modeled by Figure 7e, where several windows communicate with each other through a common rALU. Figure 7c illustrates the nested compound Scan Patterns for this example. Note, that with respect to performance this parallelism of scan windows makes sense only, if interleaving memory access is used, which is supported by the regularity of the storage scheme and the Scan Patterns.

```
Array        CGFFT  [ 1:9, 1:16, 63:0];                    (68)
ScanPattern                                                 (69)
             SP1    is  7  steps  [0,2] ,                   (70)
             SP23   is  7  steps  [0,1] ,                   (71)
             HLScan is  3  steps  [2,0] ;                   (72)

Adjustment     ThreeW  is                                   (73)
         SW1       [1:2, 1:2, 63:0] handle [1,1],           (74)
         SW2, SW3 [1:1, 1:1, 63:0] handle [1,1];            (75)

rALUsubnet FFT is                                           (76)
         SW2 = SW1[1,2] + SW1[1,1] * SW1[2,2],              (77)
         SW3 = SW1[1,2] - SW1[1,1] * SW1[2,2],              (78)

                      .
                      .
```

Figure 8. FFT example, section of the declaration part (operator definition omitted)

This section has introduced the essentials of the language MoPL-3 by means of two algorithm implementation examples. The main objective of this section has been the illustration of the language elements for data sequencing programs and the illustration of its comprehensibility and the ease of its use.

```
begin                                                       (79)
adjust ThreeW;                                              (80)
apply FFT ;                                                 (81)
moveto CGFTT [0,0], [2,0], [2,8] ;                          (82)
HLScan fork SP1, SP23, SP23 join ) ;                        (83)
end                                                         (84)
```

Figure 9 Statement part of the FFT example

4. CONCLUSIONS

The paper has briefly summarized the new Xputer machine paradigm, has demonstrated its basic execution mechanisms, and, has shown its very high efficiency having been published earlier. The paper has introduced a new high level Xputer programming language MoPL-3 and has illustrated its conciseness, comprehensibility and the ease of its use in data-procedural programming for Xputers. An earlier version of the language (MoPL-

2) has been implemented at Kaiserslautern on VAX station under ULTRIX. It is an essential new aspect of this new computational methodology, that it is the consequence of the impact of field-programmable logic and features from DSP and image processing on basic computational paradigms. Xputers, their languages and compilers open up several promising new directions in research and development - academic and industrial.

REFERENCES

[1] A. Ast, R. W. Hartenstein, H. Reinig, K. Schmidt, M. Weber: "A new machine paradigm as a consequence of DSP evolution"; in: (ed.: M. A. Bayoumi) VLSI Design Methodologies for DSP Architectures and Applications; Kluwer Academic Publishers, 1993

[2] M. Christ: Texas Instruments TMS 320C25; *Signalprozessoren 3*; Oldenbourg-Verlag 1988

[3] R. Freeman: "User-Programmable Gate Arrays"; IEEE Spectrum, Dec.1988.

[4] R. W. Hartenstein, A. G. Hirschbiel, K. Lemmert, K. Schmidt, M. Weber: "A Novel Paradigm of Parallel Computation and its Use to Implement Simple High Performance Hardware"; Int´l Conf. on Information Technology, Tokyo, Japan, Oct. 1990.

[5] R. W. Hartenstein, A. G. Hirschbiel, K. Lemmert, K. Schmidt, M. Weber, “The Machine Paradigm of Xputers and its Application to Digital Signal Processing”, Proc. of 1990 International Conference on Parallel Processing, St. Charles, Oct. 1990.

[6] R. W. Hartenstein, A. G. Hirschbiel, K. Schmidt, M. Weber: "A Novel ASIC Design Approach based on a New Machine Paradigm"; 16th European Solid-State Circuits Conference, Grenoble, France, Sept. 19-21, 1990.

[7] R. Hartenstein, G. Koch: "The universal bus considered harmful"; in: (eds.) R. Hartenstein, R. Zaks: Microarchitecture of Computer Systems, North Holland, 1975.

[8] A. G. Hirschbiel: "A Novel Processor Architecture based on Auto Data Sequencing and Low Level Parallelism"; Ph. D. dissertation, Universität Kaiserslautern, 1991.

[9] Josef Hoffmann: "Redundanz raus"; Computer Time, Heft 6, 1991.

[10] L. Matterne et al.: "A Flexible High-performance 2-D Discrete Cosine Transform IC"; Proc.. Int’l Symp on Circuits and Systems, Vol. 2, IEEE New York, 1989

[11] N. N. (Motorola): DSP 56000 / 56001 Digital Signal Processor User's Manual; Motorola Corp., 1989.

[12] N. N. (Plessey): Quickgate (Product Overview); Plessey Semiconductors, Swindon, U.K., May, 1990.

[13] Gregory K. Wallace: "The JPEG Still Picture Compression Standard"; Communications of the ACM, Vol. 34, Nr. 4, April 1991.

[14] M. Weber: "An Application Development Method for Xputers"; Ph. D. dissertation, Kaiserslautern University, 1990.

[15]

A. Ast, J. Becker, R. Hartenstein, et al.: MoPL-3: A New High Level Xputer Programming Language; 3rd Int. Workshop On Field Programmable Logic And Applications, Oxford, 7.-10. September 1993

A. Ast, J. Becker, R. Hartenstein, et al.: MoPL-3: A New High Level Xputer Programming Language; 3rd Int. Workshop On Field Programmable Logic And Applications, Oxford, 7.-10. September 1993