MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

A THREE-DIMENSIONAL-CMOS DESIGN METHODOLOGY

Microelectronic and Information Sciences Center

Technical Report #01

B. Hoefflinger
Electrical Engineering
University of Minnesota

S.T. Liu
Honeywell Corporate Technology Center
Bloomington, Minnesota

B. Vajdic
Electrical Engineering
University of Minnesota

MKC
9M583t

A THREE-DIMENSIONAL-CMOS DESIGN METHODOLOGY

MEIS Technical Report #01

B. Hoefflinger
S.T. Liu
B. Vajdic

The research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

ABSTRACT

A technology-updatable design methodology for three-dimensional (3D) CMOS circuits has been developed. Four levels of abstraction have been implemented with topographical congruence: 1. Technology Level, 2. Mask Level, 3. Transistor Level, 4. Logic Level. A novel transistor level symbolic representation is introduced which emphasizes the three-dimensional nature of the circuits. A logic level representation was developed in such a way to assure masking of unnecessary details while providing all the necessary logic and placement information. Potential advantages of the 3D CMOS circuits are discussed. A number of design examples is presented, emphasis being placed on processor and memory elements.

1. INTRODUCTION

Most CMOS circuits are composed of complementary pairs of p-channel and n-channel transistors where each pair shares a joint gate. Such configuration has led researchers to arrange the transistors vertically on top of each other with the insulated gate sandwiched in between. The upper transistors are formed in recrystallized layers of silicon and corresponding CMOS inverters have been reported [1,2,3]. These three-dimensional CMOS (3D CMOS) technologies are maturing rapidly. It is the purpose of this paper to offer a technology-updatable design methodology for emerging 3D CMOS circuits. This technology has the potential of gate areas 3 to 5 times smaller than conventional CMOS circuits for the same design rules and mask count.

2. KEY FEATURES OF 3D CMOS

Figure 1a shows the basic topography of the proposed 3D CMOS invecter cross-section. The shared gate is n^+ polysilicon. The upper gate oxide is grown from it. The p-channel transistor is built in a recrystallized thin film of silicon. The n-channel transistor is built in the bulk p-substrate as usual with NMOS processing. Planar auto-contacting is made at the p^+ and the n^+ drains. With this arrangement the output is carried over in n^+ polysilicon (or polycide) to the gate of the other 3D CMOS devices with no contacts and steps required. Also only one drain contact cut is necessary in 3D CMOS rather than two drain contacts required in conventional 2D CMOS. The cross-section, although highly schematic, reflects the key features of a production-oriented, high-density 3D CMOS technology:

Full oxide isolation;

Planarized layers;

Auto-contact of P^+/n^+ drains (possibly silicided);

Direct n^+ output to n^+ polysilicon gate input;

Vertically aligned p^+ and n^+ sources with thick insulator;

Buried n^+ bias.

This technology offers some very desirable features from the circuits and systems point of view. One of the main incentives for implementing the 3D CMOS integration is significant density gain. There are four major factors responsible for this gain: 1. p-channel devices of the complementary pair do not require any chip area, 2. there is no need for isolation of PMOS and NMOS transistors (wells etc), 3. intracell routing, 4. there are significant topological advantages. Based on the technology features described above, it is obvious that wire-routing problems are alleviated due to the fact that there is a smaller number of wires and that they are generally shorter and placed more systematically. Closely related to this argument is the conclusion that this technology potentially can improve the speed of the circuits. The latch-up problem is eliminated. The contact count is decreased and reliability of 3D CMOS circuits is expected to be improved compared with its 2D counterparts. The number of masks for the 3D process is no greater than that for the standard 2D process while the contact count is decreased.

3. DESIGN METHODOLOGY

The design methodology has been developed on a work-station with interactive color graphics (e.g. HP 9836 and PIGLET or CALMA GDS II). Four levels of abstraction are implemented with topographical congruence:

1. Technology Level
2. Mask Level
3. Transistor Level
4. Logic Level

While the transistor and logic levels are invariant, the technology and mask levels are updatable in the process file related to technology progress. Figure 1 shows a 3D CMOS inverter represented at all four design levels. The technology level (Fig. 1a) does not represent all technology details in order to retain clarity. The designer needs access to this level because the emerging high device densities can no longer be comprehended in a 2D top view, only. Therefore, for level 1, technology cross sections can be called by the designer in either x or y directions.

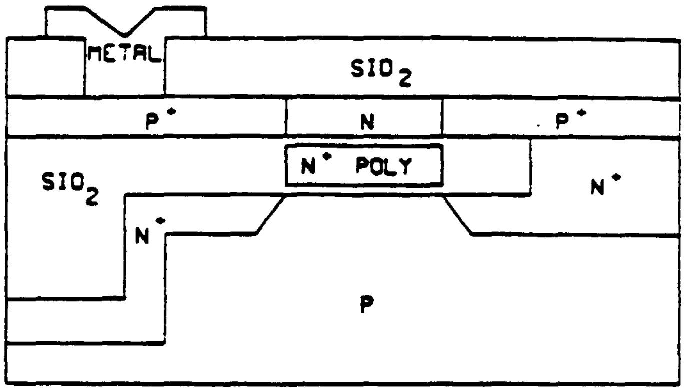
The mask level is represented in level 2 (Fig. 1b) as either individual or superimposed mask levels.

Level 3 (Fig. 1c), the transistor level, reflects the basic topography of a 3D CMOS cross section, a PMOS transistor on top of a NMOS transistor with a sandwiched joint gate. This novel symbolic representation at the transistor level is a natural consequence of the actual physical arrangement shown at the technology level.

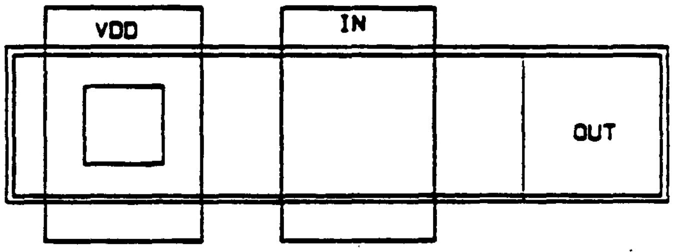
The main goal at the logic level is to achieve a topographical congruence with the three lower levels in order to simplify both design and verification. Fig. 1d indicates that for the case of a simple inverter. However, the same idea has been applied to all designed circuits.

Consider a two input NAND gate having two NMOS transistors in series and two PMOS transistors in parallel. The 3D realization of this gate is shown at the mask level (Fig. 2a), transistor level (Fig. 2b) and logic level (Fig. 2c). Note that the parallel connection for the two PMOS drains can be placed right on the structure itself resulting in a compact "one unit width" cell. A similar design applies to a NOR gate. Next is shown (Fig. 3) the complex gate performing a logic AND/NOR function involving three inputs.

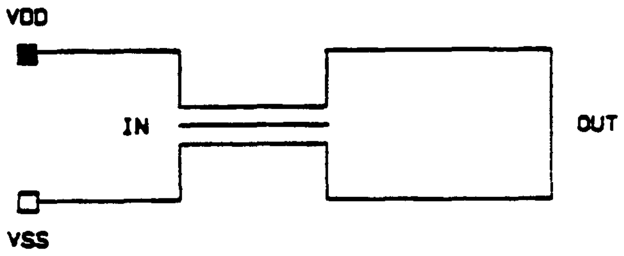
To facilitate effective 3D CMOS design, the screen is organized as an



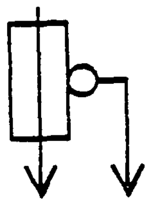
a. Technology Level



b. Mask Level



c. Transistor Level



d. Logic Level

Fig. 1. Inverter

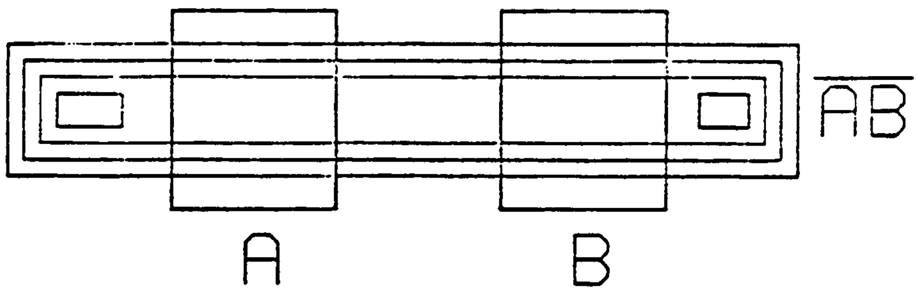
uncommitted chip image as shown in Fig. 4a for level 3, the transistor level. Potential locations of transistors or transistor pairs and grids for potential p^+, n^+ and metal interconnects are indicated. Positioning of the cursor places or removes a transistor pair or a single transistor and establishes various interconnects. Similarily, macros are placed on the grid. Fig. 4b shows an example of a master-slave flip-flop frequency divider composed of 4 AND/NOR macros. The mask level representation is shown in Fig. 4c. The logic level representation (Fig. 4d) performs the function of masking logically irrelevant details (e.g. intercell wiring) present at the transistor level. Note, however, that this level not only carries the information about the logic function of the circuit but also shows the actual physical location. This particular feature is very important for ease of logic verification as well as invert simulation. Fig. 5 shows a microphotograph of the frequency divider circuit which is part of an exploratory 3D CMOS test chip [4].

4. DESIGN EXAMPLES

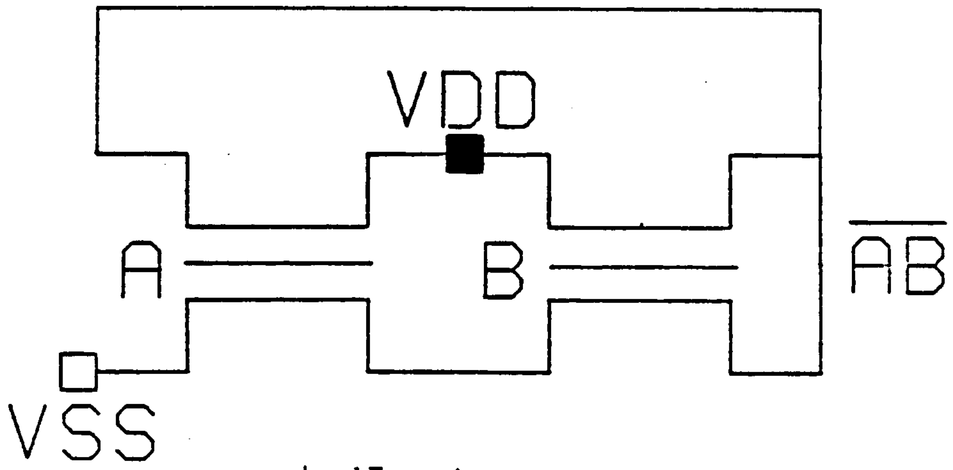
In view of what was said about potential advantages of 3D CMOS circuits, one can conclude that this technology will be found most beneficial in the areas where density, speed and reliability are of prime concern. Processor elements (adders, multipliers, etc) and memory elements should benefit most. A static type CMOS full adder is shown at the transistor level (Fig. 6a) and at the logic level (Fig. 6b). The Boolean equations are

$$Q_S = ABC + (A+B+C)\overline{Q}_C$$
$$Q_C = AB + (A+B)C$$

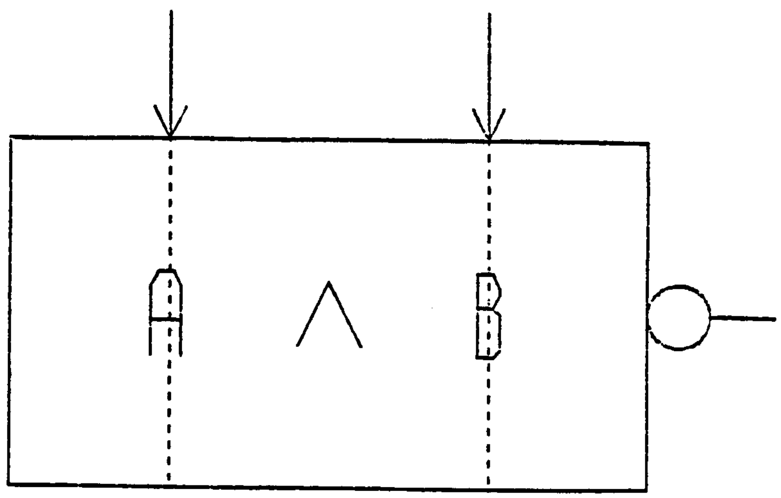
A particularly interesting circuit is the transmission - gate type full adder (Fig. 7). Due to the extensive use of very compact 3D CMOS circuits per-



a. Mask Level

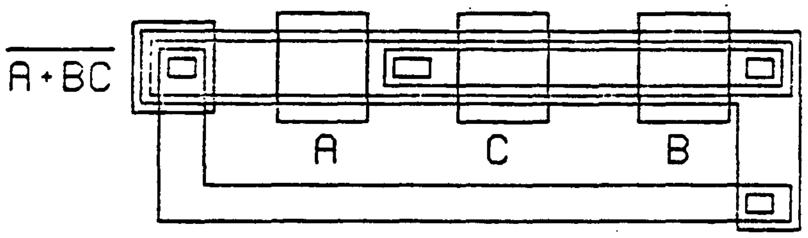


b. Transistor Level

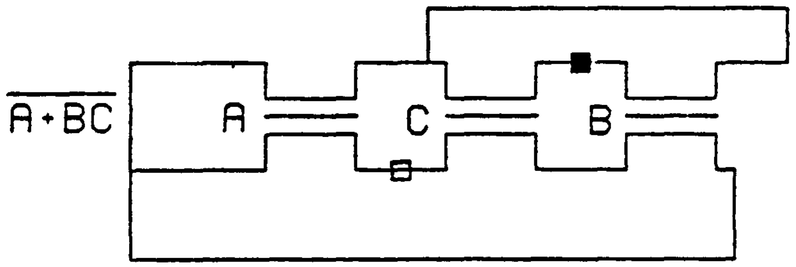


c. Logic Level

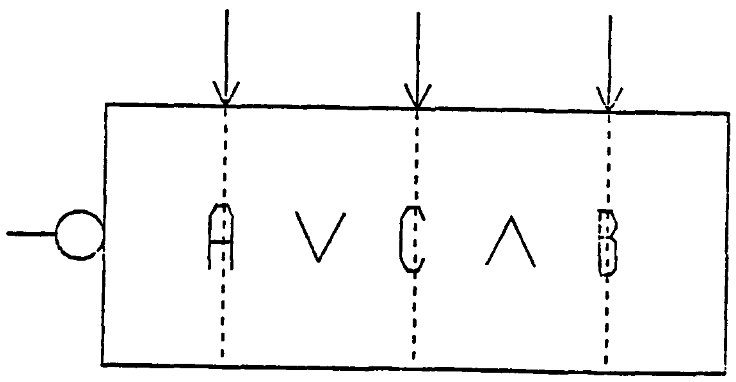
Fig. 2.   NAND Gate



a. Mask Level

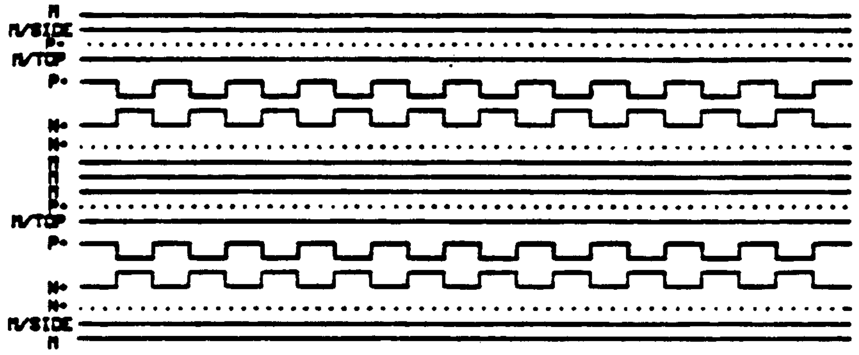


b. Transistor Level

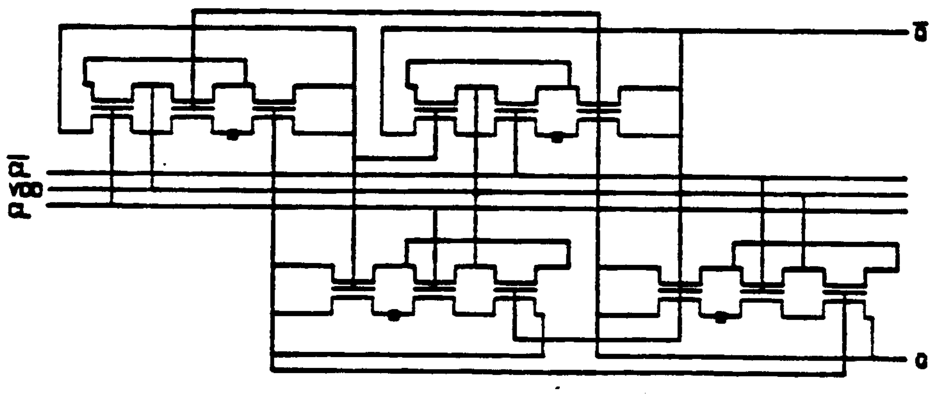


c. Logic Level

Fig. 3. AND/NOR Gate

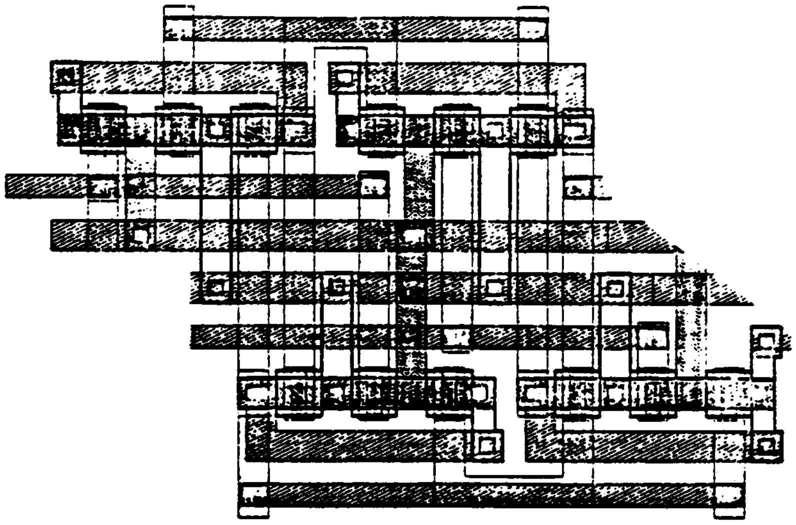


a. Uncommitted Chip Image

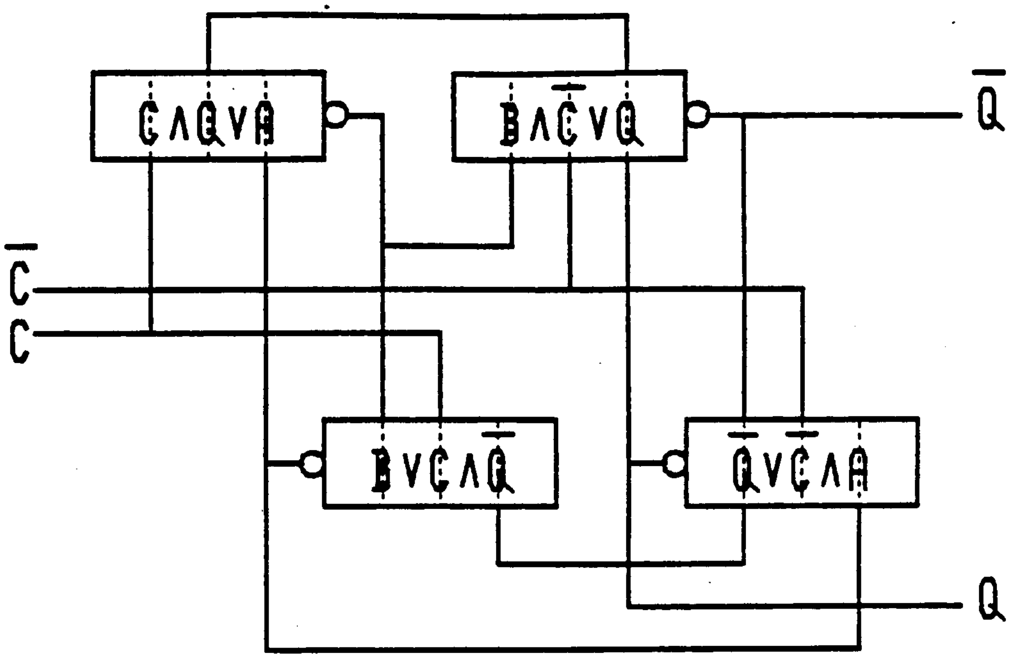


b. Transistor Level

Fig. 4. Master-Slave Flip-Flop



c. Mask Level


$C \wedge Q \vee A$
$B \wedge \overline{C} \vee Q$
$\overline{Q}$
$\overline{C}$
C
$B \vee C \wedge \overline{Q}$
$\overline{Q} \vee \overline{C} \wedge A$
Q

d. Logic Level

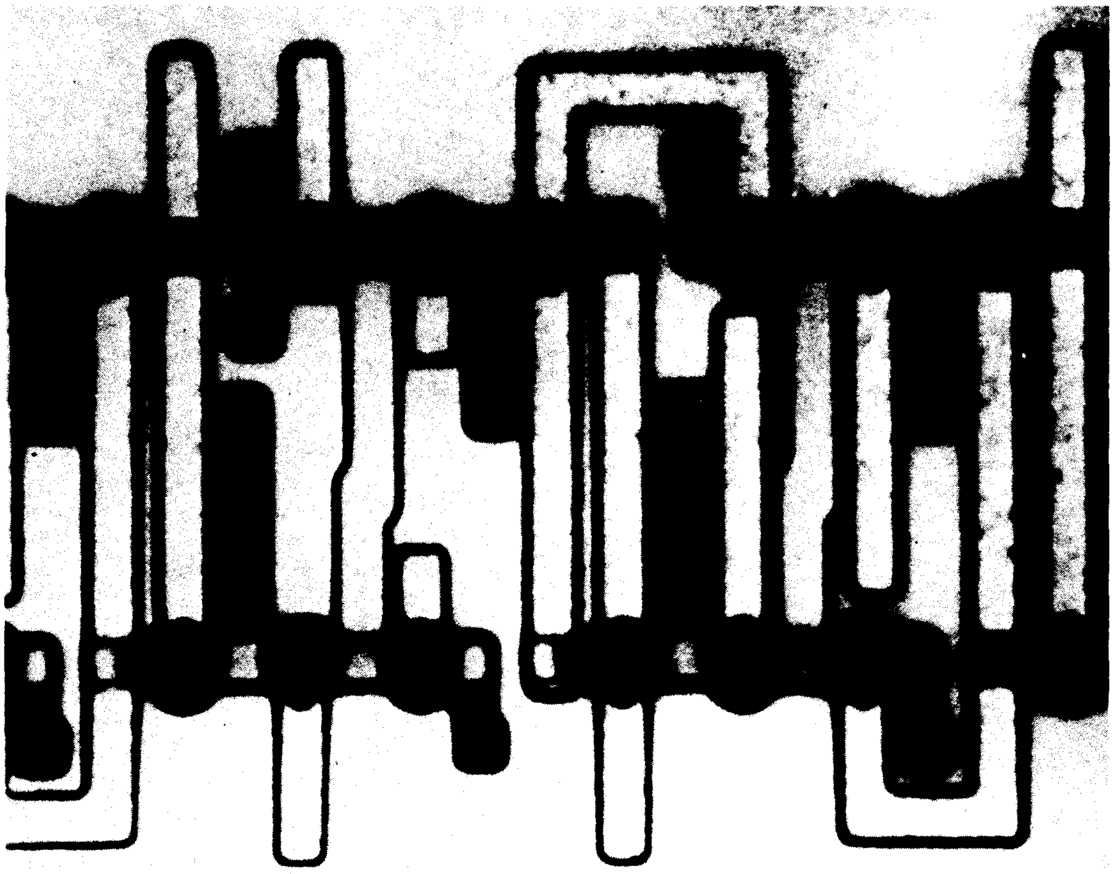
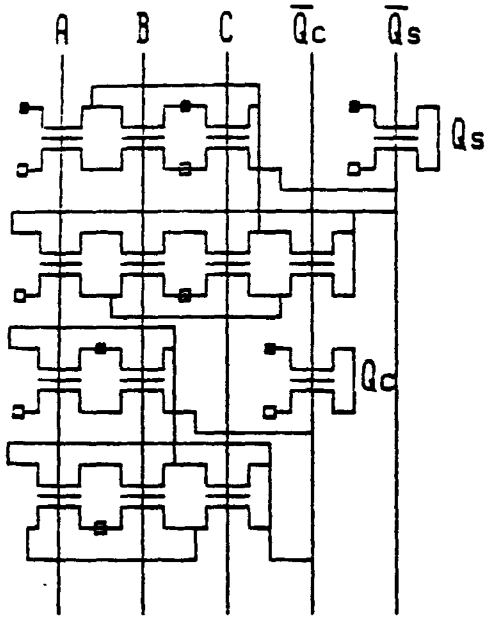
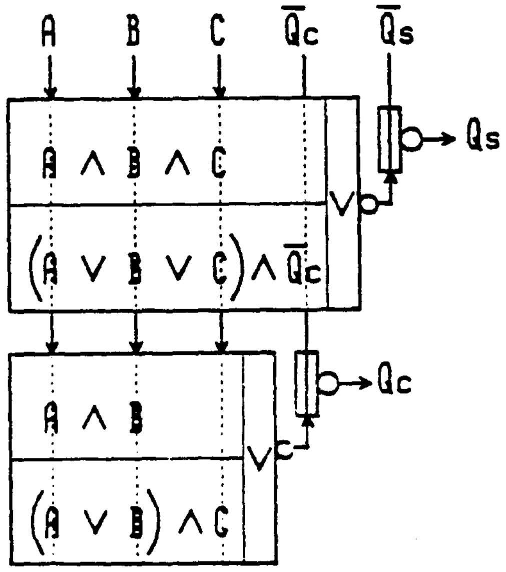


Fig. 5. Microphotograph of the Flip-Flop circuit prior to metallization

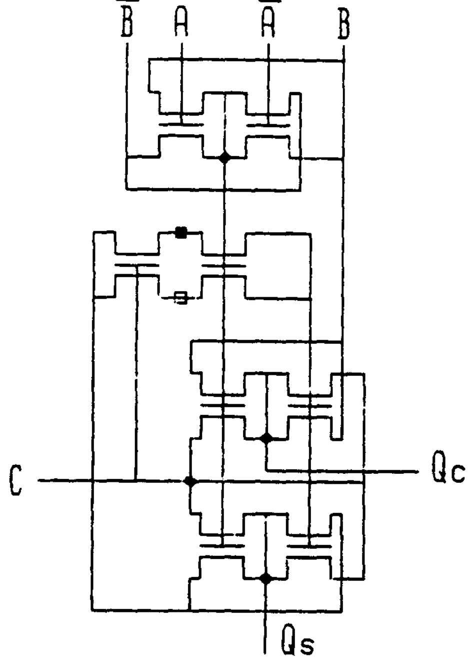


a. Transistor Level

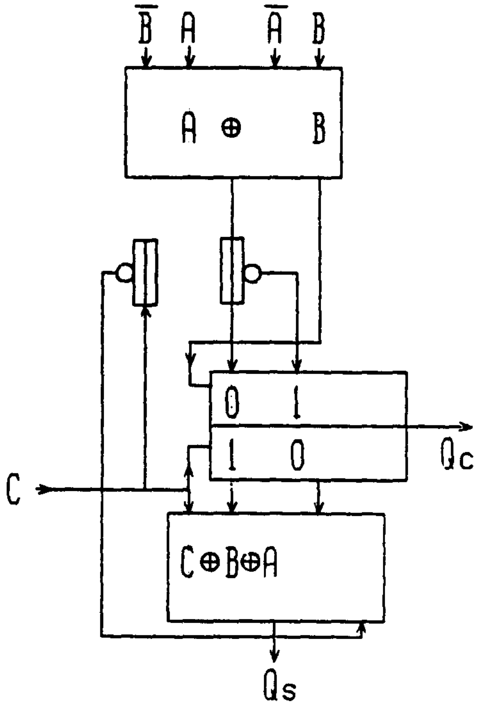


b. Logic Levl

Fig. 6. Static Type Full Adder



a. Transistor Level



b. Logic Level

Fig. 7. Transmission-Gate Type Full-Adder

forming XOR and SELECT functions, a very significant gain in density is obtained. This is documented in Table 1 which contains a comparison of required areas. Note that both adder circuits have been laid out in a gate-matrix style.

AREA COMPARISON - FULL ADDER	
BULK CMOS	$13P \times 15P = 195P^2$
SOI	$11P \times 12P = 132P^2$
3-D CMOS-1	$8P \times 7P = 56P^2$
3-D CMOS-2	$6P \times 6P = 36P^2$

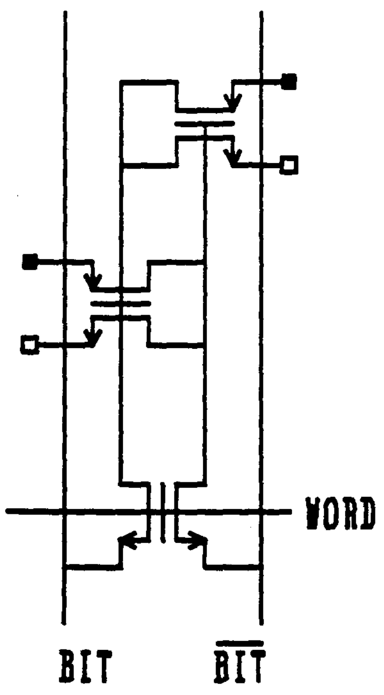
Table 1: Area comparison in pitch units

A 6-transistor static RAM cell is shown in Fig. 8. The density gain over its bulk CMOS counterpart in this case, is attributed to the fact that there is no need for an area consuming well and also to the clear topological advantage. Cross-coupling in 2D CMOS presents a special layout problem which tends to increase the area and contact count. In 3D CMOS, cross-coupling is accomplished by simply aligning the input of an inverter with the output of the other. Not a single extra contact is needed. Table 2 contains some of the average figures supporting the density gain expectations.

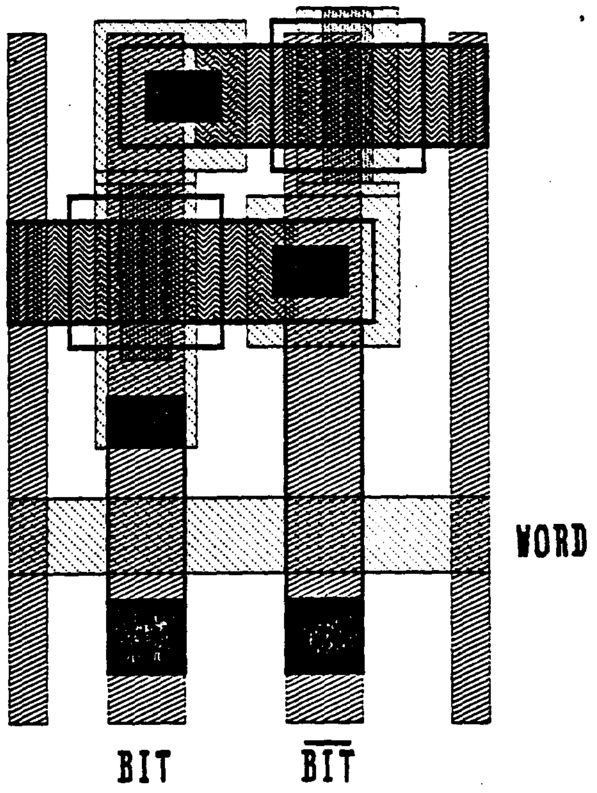
AREA COMPARISON - (RELATIVE)

	2-D CMOS		
	BULK	SOI	3-D CMOS
STATIC CMOS RAM	3	2	1
COMPLEX CMOS GATES	5	3	1
CMOS PROCESSOR ARRAY	3	2	1

Table 2: Density gain figures



a. Transistor Level



b. Mask Level

Fig. 8. 6-Transistor Static RAM cell

5. CONCLUSION

The design methodology described here was successfully applied to the design of a test chip and numerous test structures. The 3D CMOS circuits have many desirable features. They are listed in Table 3.

Features	Consequences
high density	higher level of integration
reduced interconnects	less capacitance and higher speed
fewer contacts	yield and reliability
high regularity	better design productivity and verification

Table 3: Features of 3D CMOS circuits

References

1. J. F. Gibbons and K. F. Lee, IEEE EDL-1, 117 (1980).
2. G. T. Goeloe, E. W. Maby, D. J. Silversmith, R. W. Mountain, and D. A. Antoniadis, IEDM Tech. Digest 554-556 (1981).
3. J. P. Colinge, E. Demoulin, and M. Lobt, IEEE ED-29, 585 (1982).
4. S. T. Liu, B. Hoefflinger, and B. Vajdic, UGIM-83 Symposium Proceedings, 76-78 (1983).

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

AN INTERFACE CATALYTIC EFFECT: Cr AT THE Si(111)-Au INTERFACE

Microelectronic and Information Sciences Center

Technical Report #02

A. Franciosi
Department of Chemical Engineering
and Materials Science
University of Minnesota

D. G. O'Neill
Synchrotron Radiation Center
University of Wisconsin-Madison

J. H. Weaver
Department of Chemical Engineering
and Materials Science
University of Minnesota

AN INTERFACE CATALYTIC EFFECT:
Cr AT THE Si(111)-Au INTERFACE

MEIS Technical Report #02

A. Franciosi
D. G. O'Neill
J. H. Weaver

The research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

ABSTRACT

Synchrotron radiation photoemission studies of the effect of Cr interlayers on Si(111)-Au interface reaction show that Cr concentrations below 1 x 10^{15} atoms/cm^2 retard Si-Au intermixing, concentrations between 1 and 7.5 x 10^{15} atoms/cm^2 promote Si-Au intermixing, and concentrations in excess of 8 x 10^{15} atoms/cm^2 sharply reduce intermixing. There variations are shown to depend on the three formation stages of the Si-Cr junction. Cr itself is shown only to be indirectly involved in the Si-Au reaction and Si is to be the only moving species.

The driving forces responsible for atomic interdiffusion at Si-metal interfaces have been the subject of intense discussion.[1-6] To understand this atomic interdiffusion, we must identify the parameters that control interdiffusion kinetics and the junction profile, and we must determine the relationships that exist between these parameters and the chemical activity of the species involved in the interface formation process.[7-9]

In order to better understand interdiffusion, we performed an investigation of the effect of Cr interlayers on the atomic interdiffusion of Si and Au at the Si(111)2x1-Au interface, with special emphasis on the chemical aspects of interdiffusion.

In this paper, we show that for Cr coverages from 2 to 9 Å the interlayer promotes the Si-Au intermixing, while the interdiffusion is reduced for Cr coverages above 9 Å (10 monolayers) or for coverages less than one monolayer. This strikingly non-monotonic behavior rules out simple "diffusion barrier" effects and is related, instead, to the three stages of Si-Cr reaction. Each stage corresponds to a different morphology or microscopic arrangement of the Si atoms at the surface and, hence, a different energy content of the Si-Si bonds.

Our results show that the interlayer Cr atoms do not appear directly involved in the Si-Au reaction. Instead, since the Cr interlayer affects the formation of the Si-Au junction and its concentration modulates the Si-Au reaction, it seems natural to define the role of the Cr atoms as that of an "interface catalyst" for the Si-Au reaction.

The Si-Cr-Au system was chosen because the Si-Cr and the Si-Au interfaces are well characterized[10-13] and neither gives rise to island formation at room temperature. The relative surface concentration of the different species and the evolution of the chemical bonding was followed systematically through synchrotron radiation photoemission from valence and core electronic states, both as a function of Au coverage and Cr interlayer thickness. The experiments were performed at the University of Wisconsin Synchrotron Radiation Center with a "Grasshopper" grazing-incidence monochromator for $40 \leq h\nu \leq 140$ eV. The photoelectrons were analyzed by a double-pass cylindrical mirror energy analyzer, and the overall resolution (monochromator + analyzer) was 0.3-0.5 eV. Cr and Au were evaporated <u>in situ</u> from W coils onto cleaved n-type Si crystals (P-doped , 10^{15} cm^{-3}) and the overlayer thickness was measured by a quartz-crystal monitor ($\Theta_{Cr} = 1$ Å = 1.1 monolayers; $\Theta_{Au} = 1$ Å = 0.8 monolayers; 1 monolayer = 7.6×10^{14} atoms/cm^2). Cr depositions of 0.5-15 Å were made immediately before a series of Au depositions (maximum Au coverage of 50-70 Å for each Cr interlayer thickness). All experiments were performed at pressures of 5×10^{-11} Torr ($\leq 6 \times 10^{-10}$ Torr during evaporation and at room temperature).

In Fig. 1 we show representative photoelectron energy distribution curves (EDC's) for the valence band of the Si-Cr-Au inerface at a fixed Au coverage of 20 Å as function of Cr interlayer thickness[14]. The topmost spectrum corresponds to Si(111)-Au without Cr, and the bottom-most shows the valence bands for a 20 Å Au film on an inert substrate[15]. The differences between these two, which reveal the formation of the Si-Au interface, have been explained as modifications of Au 5d-derived density of states features caused by Si atoms in the Au matrix, mainly through the change of the d-d overlap[12,14,16-17]. Taking such differences[17] as a fingerprint of Si-Au reaction, the systematics in Fig. 1 show that a Si-Au reaction can occur in the presence of a Cr interlayer but that the intermixing is sharply reduced when the interlayer thickness increases above 9 Å.

The presence of Si atoms in the Au matrix is clearly indicated by the topmost spectra of Fig. 1, but the valence band spectra show no evidence of Cr at the surface. To determine whether Cr intermixes with the Au overlayer, we measured the integrated Cr-derived 3p core emission as a function of Au coverage for two different Cr interlayer thicknesses, as shown at the top of Fig. 2. These results show an exponential attenuation of the Cr emission. Comparison to a simple exponential attenuation calculated with an escape depth of 5 Å (dashed line), indicates Si outdiffusion through the interlayer into the Au layer. This Si outdiffusion is quantitatively more important for an interlayer thickness of 2 Å than for an interlayer thickness of 8 Å. There is no evidence of Cr outdiffusion into the Au film.

In a series of previous papers we showed that the deposition of Cr onto the Si(111) surface produces a Si-Cr mixed surface phase of variable composition[11]. When the Si-Cr-Au junction is formed, we now see that Si diffuses into the Au matrix and, in principle, this might imply Si-depletion in the Si-Cr region. However, the results shown in Fig. 2 indicate that this is not the case. The Si-Cr reaction which occurs during deposition of Cr on Si(111) produces a chemical shift as large as 0.3 eV for the Cr 3p cores (mid-section of Fig. 2), but there is no variation of this chemical shift when Au is deposited onto the surface (lowest section). Hence, within experimental uncertainty of ~0.1 eV, the average local chemical environment of Cr atoms in the Si-Cr phase remains the same after the Si-Au interdiffusion. This implies that as Si enters the Au overlayer, a corresponding amount must enter the Si-Cr phase to maintain the local stoichiometry. The net results of this diffusion is transport of Si atoms through a stable Si-Cr interface phase.

In Fig. 3 we show very important results which demonstrate that a picture in which the Si-Cr phase acts as a diffusion barrier for atomic interdiffusion of Si and Au is inadequate. By measuring the attenuation of the Si 2p core emission as a function of Au overlayer for different Si-Cr interlayer thickness, we can show that the attenuation is not proportional to the thickness of the Cr "barrier." In Fig. 3 the emission intensity at a given Au coverage and interlayer thickness is normalized to the initial Si 2p emission from the Si-Cr phase. For comparison, results for the Si(111)-Au interface without Cr deposition are also shown (dot-dashed line from Ref. 14). The best representation of the net effect of the presence of the interlayer on the Si-Au reaction is done by a normalization which takes into account the initial emission from Si surface atoms since this varies with interlayer thickness. Three different ranges of interlayer thickness are clearly evident. First, for Cr coverage of 0-1 Å monolayer (lower section of Fig. 3), the interlayer weakly affects Si-Au intermixing and reduces Si outdiffusion. Second, for Cr coverages between 1.4 and 9 Å, the interlayer promotes Si outdiffusion. Finally, for Cr coverages above 10 monolayers the interlayer sharply reduces Si outdiffusion. The maximum promotion effect is obtained at about 2 Å of interlayer thickness, and the trend reverses itself in the monolayer Cr coverage range. Clearly, these results cannot be interpreted with only a simple diffusion barrier model. Hence, there must be a correlation to changes in the local chemistry and energy balance.

In studies of Si(111)-Cr interface formation we suggested[11,18] that there are three formation stages for the Si-Cr interface. In the first

($\Theta_{Cr} \lesssim 1 - 1.5$ ML), small core chemical shifts and the relatively rapid attenuation of the substrate emission suggest little or no interdiffusion and weak adatom-substrate interaction ("weak chemisorption"). For $1.5 \lesssim \Theta_{Cr} \lesssim 9$-10 ML, the core levels shift, there is slower attenuation of substrate emission, and the valence states evolve, thus providing evidence of Si-Cr interdiffusion ("reactive interdiffusion"). For $\Theta_{Cr} > 10$ ML, the Si-Cr reaction is completed and further metal deposition produces an unreacted Cr film on top of the reacted Si-Cr phase ("fully reacted").

The clear one-to-one correspondence between the different stages of Si-Cr interface reaction and the different regimes of catalytic effect of the Cr atoms in the interlayer is compelling and allows us several conclusions. The promotion catalytic effect of the interlayer on the Si-Au interdiffusion is related to an increase in reactivity of the Si surface atoms that occurs during the reactive interdiffusion stage of the Si-Cr system. The maximum promotion effect occurs just above the onset of the reactive interdiffusion; the interlayer reduces Si outdiffusion below the onset of the Si-Cr reaction. The sharp transition suggests that during the weak chemisorption stage the presence of Cr atoms leaves the energy content of the Si-Si bonds relatively unchanged while the metal atoms "cover up" the ordered surface and act as a thin barrier against the Si-Au interaction. When Cr and Si begin to react, the average Si-Si binding energy is reduced and the broken surface bonds[18] represent sites where the chemically driven Si-Au intermixing may start. The chemically activated character of this interdiffusion process is emphasized by the substantial reduction of intermixing that occurs for interlayer thickness above 9 Å, showing that even very thin layers of unreacted Cr on top of the Si-Cr phase act as an effective diffusion barrier for the intermixing.

The gradual reduction in Si outdiffusion that takes place for increasing Cr coverage between 2 and 9 Å is related either to a progressive change of the reactivity of the Si atoms in the Si-Cr phase due to changes in bonding character or to the growing importance of a rate-limiting mass transport step in the intermixing process. Discrimination between the two mechanisms will require temperature dependent studies (to vary the importance of mass transport in limiting the rate of reaction) or procedure whereby the actual concentration profile of Si atoms in the Si-Cr interlayer can be determined. The latter would allow a quantitative correlation of data such as ours with the effective surface concentration of Si atoms and therefore obtain the specific activity of the Cr catalyst as a function of interlayer thickness.

In summary, we have shown that interface reaction at the Si-Au junction can be modulated by Cr atoms at the interface, that these Cr atoms can promote or reduce Si outdiffusion from the bulk, and that the non-monotonic catalytic trend is related to the existence of several non-equivalent microscopic structures for the Si-Cr binary system at the interface. These results point to the importance of investigations of the influence of the interface catalytic effect on the formation of the Schottky barrier because such studies could clarify the relative importance of metal-induced defects and the formation chemistry of the extended interface region in determining the Schottky barrier heights.

ACKNOWLEDGEMENTS

This work was supported by the U.S. Army Research Office under Grant No. DAAG29-83-K-0061. We gratefully acknowledge the support of E. M. Rowe and coworkers at the Wisconsin Synchrotron Radiation Center supported by NSF DMR 78-21080.

REFERENCES

1 For an extensive review see: L. J. Brillson, Surf. Sci. Reports 2, 123 (1982).

2 G. Margaritondo, Solid State Electron. (in press) and references therein.

3 K. N. Tu in Thin films interdiffusion and reactions, edited by J. M. Poate, K. N. Tu, and J. W. Mayer (Wiley, Chichester, England, 1978).

4 I. Abbati, L. Braicovich, and A. Franciosi, Phys. Lett. 80A, 69 (1980).

5 K. Oura, S. Okada, and T. Hanawa, Appl. Phys. Lett. 35, 705 (1979); K. Oura, S. Okada, Y. Kishikawa, and T. Hanawa, ibid. 50, 138 (1982).

6 J. L. Freeouf, J. Vac. Sci. Technol. 18, 910 (1981); P. E. Schmid, P. S. Ho, H. Foll, and G. W. Rubloff, Phys. Rev. Lett. 18, 937 (1981).

7 L. J. Brillson, R. Z. Bachrach, R. S. Bauer, and J. McMenamin, Phys. Rev. Lett. 42, 397 (1979); L. J. Brillson, G. Margaritondo, and N. G. Stoffel, ibid. 44, 667 (1980).

8 L. J. Brillson, R. S. Bauer, R. Z. Bachrach, and G. Hansson, Phys. Rev. B23, 6204 (1981); C. F. Brucker, and L. J. Brillson, J. Vac. Sci. Technol. 19, 617 (1981).

9 J. M. Andrews and J. C. Phillips, Phys. Rev. Lett. 35, 56 (1975); G. Ottaviani, K. N. Tu, and J. W. Mayer, Phys. Rev. Lett. 44, 284 (1980).

10 L. Braicovich, C. M. Garner, P. R. Skeath, C. Y. Su, P. W. Chye, I. Lindau, and W. E. Spicer, Phys. Rev. B20, 5131 (1979).

11 A. Franciosi, D. J. Peterman, J. H. Weaver, and V. L. Moruzzi, Phys. Rev. B25, 4981 (1982).

12 I. Abbati, L. Braicovich, A. Franciosi, I. Lindau, P. R. Skeath, C. Y. Su, and W. E. Spicer, J. Vac. Sci. Technol. 17, 930 (1980) and references therein.

13 T. Narusawa, K. Kinoshita, W. M. Gibson, and A. Hiraki, J. Vac. Sci. Technol. 18, 272 (1981); P. Perfetti, S. Nannarone, F. Patella, C. Quaresima, A. Savoia, F. Cerrina, and M. Capozi, Solid State Commun. 35, 151 (1980).

14 Further valence band results can be found in A. Franciosi, D. G. O'Neill, and J. H. Weaver, J. Vac. Sci. Technol. (in press).

15 Gold film thicknesses of 10, 20, and 100 Å or more on oxidized tantalum gave results which were identical within experimental uncertainty and were representative of bulk Au.

16 I. Abbati, L. Braicovich, and A. Franciosi, Solid State Commun. 33, 881 (1980); these modifications have been quantitatively discussed through DOS calculations for a model Au_3Si silicide in O. Bisi, C. Calandra, L. Braicovich, G. Rossi, I. Abbati, I. Lindau, and W. E. Spicer (to be published).

17 With increasing Si content in Au, the Au atoms become more "atomic," shifting the 5d valence states to higher binding energy, decreasing the splitting of the two main 5d features toward the atomic spin-orbit value (1.5 eV), and affecting the localized d-d antibonding states so that the sharp d-edge of bulk Au 2 eV below E_F is displaced to higher binding energy and only a shoulder is seen in the spectra of reacted Au.

18 K. Oura, S. Okada, and T. Hanawa, Proc. 8th Int. Vacuum Congress, Cannes (France), Sept. 22-26, 1980, Suppl. to "Le vide, les Couches Minces," 201, 181 (1981).

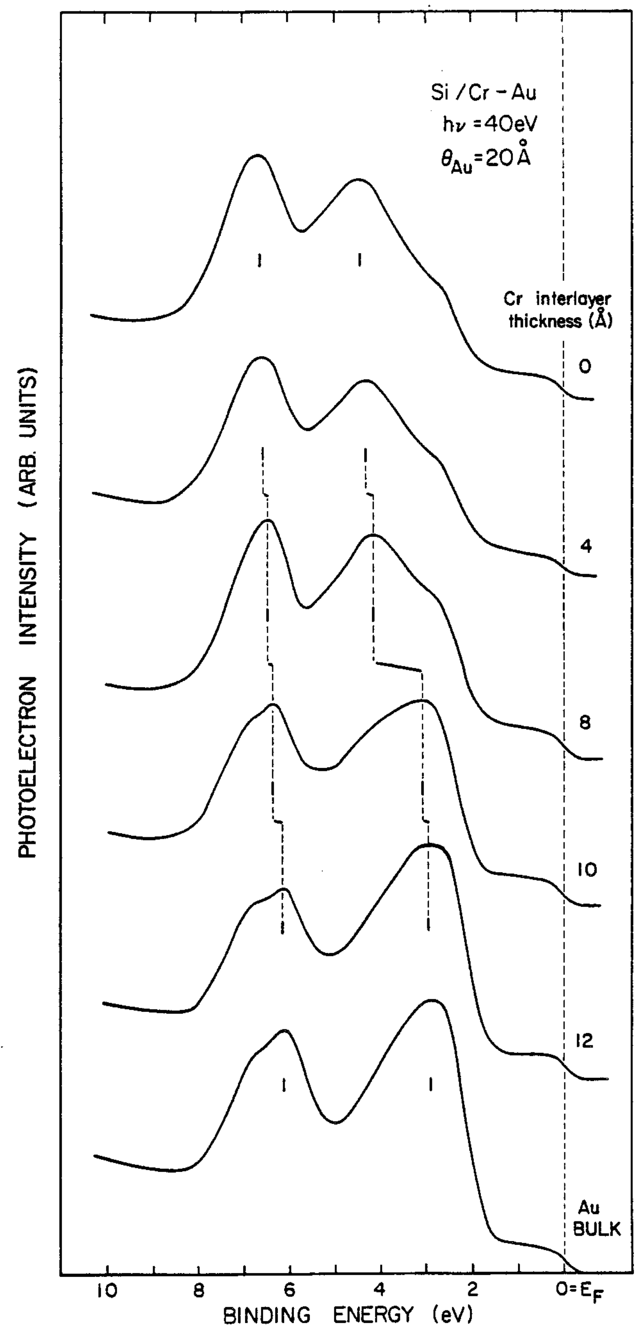


Fig. 1 Valence band EDC's of the Si-Cr-Au system as a function of the Cr interlayer thickness at a fixed Au coverage of 20 Å. The variation of the energy separation of the Au 5d main features (vertical lines) emphasizes that Si-Au intermixing takes place in the presence of the interlayer for Cr coverage below 9 Å but is sharply reduced for higher interlayer thickness.

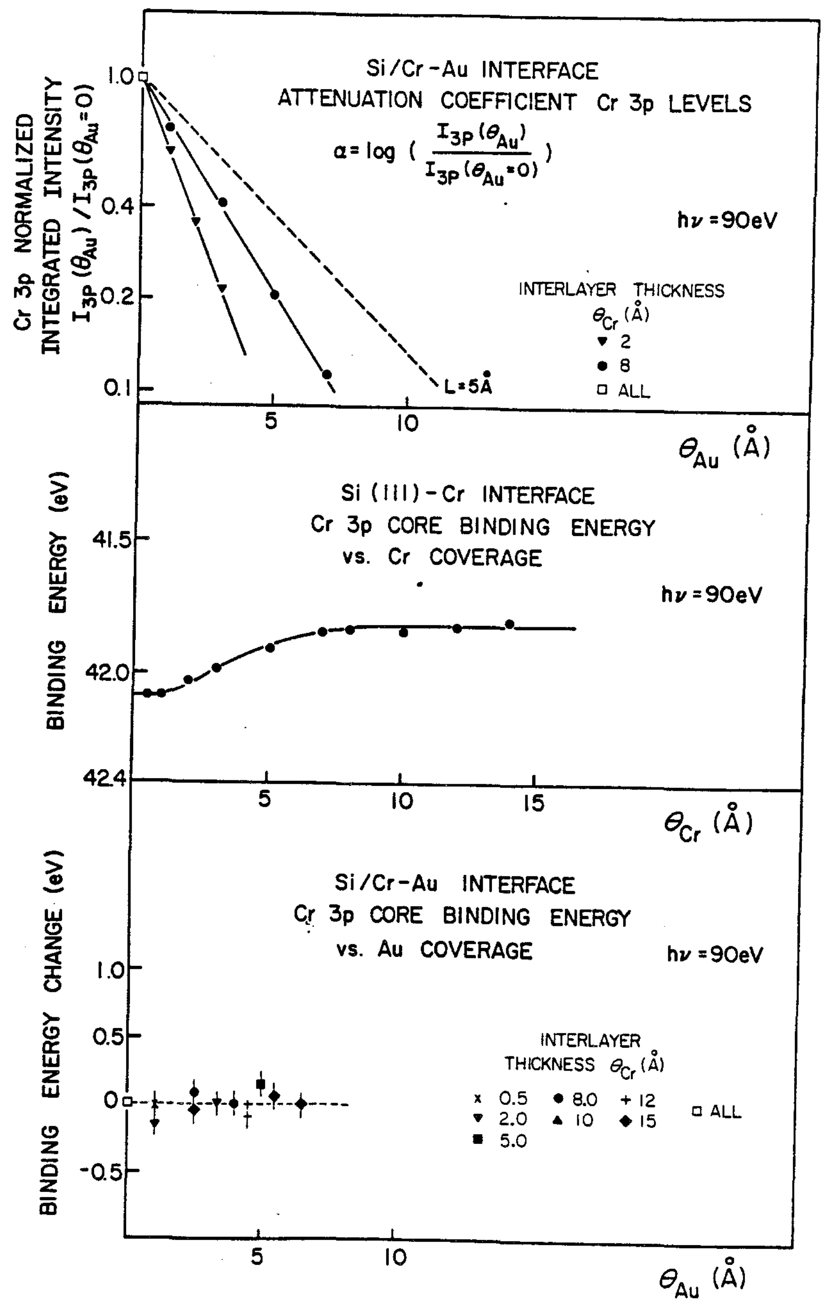


Fig. 2 Top: attenuation coefficient for the integrated emission intensity of the Cr 3p cores as a function of Au coverage on the Si-Cr surface. We show experimental data for Cr interlayer thicknesses of 2 and 8 Å. The dash line represents the theoretical result for expoential attenuation of the 3p based on an electron escape depth L = 5 Å.

Mid-section: binding energy of the Cr 3p cores as a function of Cr coverage for Cr-Si(111)2x1. The Si-Cr reaction corresponds to a chemical shift of ~0.3 eV for the Cr 3p cores.

Bottom: variation of the 3p core binding energy for the Cr atoms in the interlayer as a function of Au coverage. Within experimental uncertainty the average chemical environment of the Cr atoms remains the same after the Si-Au interdiffusion.

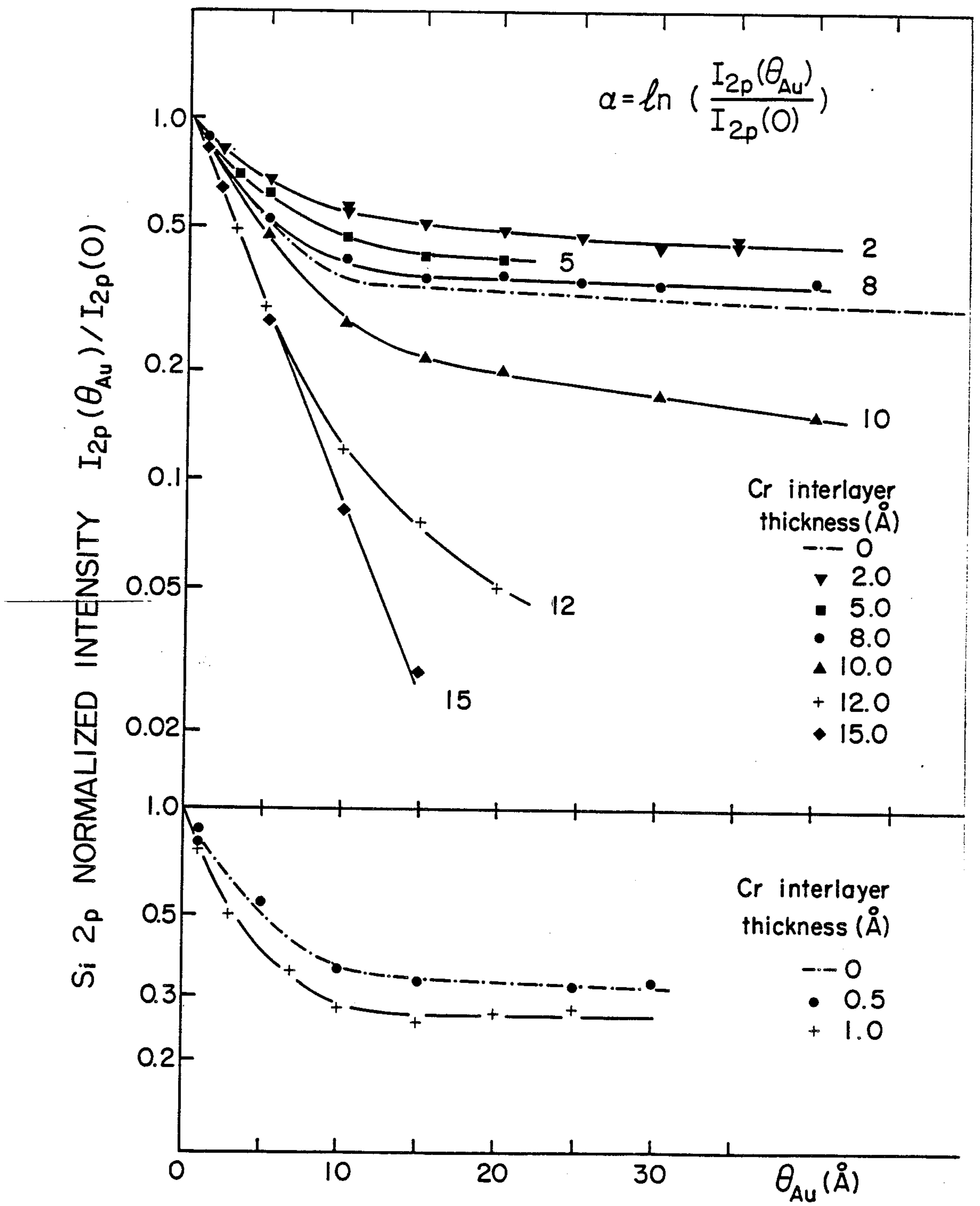


Fig. 3 Attenuation coefficient of the Si 2p core emission as a function of Au coverage on the Si-Cr phase. The data show the effect of the presence of the interlayer on Si-Au interdiffusion. The dot-dashed line gives the corresponding result for the Si(111)-Au interface (Ref. 14). For Cr coverage in the monolayer range (lower section), the interlayer slightly reduces Si outdiffusion. For Cr coverages between 2 and 9 Å, the interlayer promotes Si outdiffusion, while for all Cr coverages above 10 monolayers the interlayer sharply reduces the Si-Au reaction.

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

A SYSTOLIC DESIGN RULE CHECKER

Microelectronic and Information Sciences Center

Technical Report #03

R. Kane
Computer Science
University of Minnesota

S. Sahni
Computer Science
University of Minnesota

A SYSTOLIC DESIGN RULE CHECKER

MEIS Technical Report #03

R. Kane
S. Sahni

The research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

[1984]

ABSTRACT

We develop a systolic design rule checker (SDRC) for retilinear geometries. This SDRC reports all width and spacing violations. It is expected to result in a significant speed up of the design rule check phase of chip design.

This research was supported in part by the Office of Naval Research under contract N00014-80-C-0650 and in part by Microelectronic and Information Sciences Center at the University of Minnesota.

1. Introduction

Rapid advances in technology are making it possible to fabricate circuits of an ever increasing complexity. This increase in circuit complexity poses a severe challenge to the algorithms presently in use in design automation tools. One of the ways to meet the challenge is to develop new computer architechures capable of running these design automation algorithms efficiently. Another approach is to develop yet faster algorithms.

Several new architectures and corresponding algorithms have recently been proposed for design automation. Blank et al [BLAN81] describe a bit map processor architecture suitable for boolean operations, wire routing using Lee's algorithm, and for some design rule check (DRC) functions such as shrink and expand. Mudge et al [MUDG82] describe Cytocomputer architecture adapted for DRC and Lee type wire routing. Yet another DRC architecture is described in [SEIL82]. Some other references for special purpose architectures and associated algorithms for wire routing are [DAMM82] and [NAIR82]. A parallel processing approach for logic module placement has been developed by Ueda et al [UEDA83]. Simulation has also been the focus of several new architectural studies. The most popular such development is the Yorktown Simulation Engine ([PFIS82], [DENN82], and [KRON82]). Another logic simulation machine is described by Abramovici et al [ABRA82]. In this paper, we shall be concerned with the design of a systolic system for design rule checks. Our design differs from all earlier work on special purpose architectures for design automation in that ours is the first systolic design. Of course, systolic designs have been studied for quite some time. A valuable reference is [KUNG82]. Our systolic system for DRC's differs from earlier work on hardware assisted DRC's in that it is edge based rather than bit map based. Consequently, it has the potential of being much faster than earlier designs.

Specifically, our systolic design rule checker (SDRC) checks for spacing and width errors. The design may be extended to include other design rule checks. Our design points out the potential for systolic systems in design automation applications.

2. Polygons and Errors

In arriving at our SDRC, we made several assumptions on the nature of the polygon to be handled and also on the type of errors to be checked for. First, we assume that polygons are composed of horizontal and vertical edges only. Hence, only right angled bends are permitted. Polygons may contain holes. These holes are also restricted to be polygons with right angled bends. Figure 1 shows two example polygons that satisfy these restrictions.

This restriction on the edges composing a polygon allows a compact representation of each polygonn. This representation consist of the following :

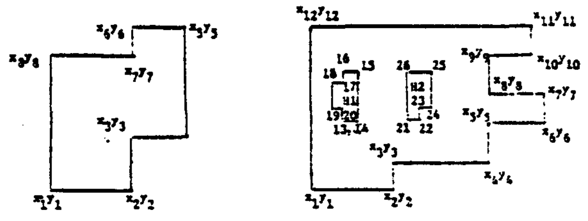
1. *Polygon number*. Each polygon is assigned a unique number. Holes within a polygon are assigned the same number as the enclosing polygon.
2. *A sequence of polygon vertices*. This sequence begins at the lowermost left hand vertex of the polygon and is obtained by traversing the polygon so that its interior lies to the left of the edge being traversed. Since all edges are either horizontal or vertical, the polygon vertices (except the first) may be described by providing a single coordinate. Thus , the polygon of Figure 1(a) is represented as:

p, n, x_1, y_1, x_2, y_3, x_4, y_5, x_6, y_7, x_8, y_1.

The first symbol p identifies this as an enclosing polygon. n is the polygon number. In case of a hole, an h is used in place of the p. Holes are traversed such that the the interior is to the left of each edge traversed. The representation for the polygon and holes of Figure 1(b) is:

p, n, x_1, y_1, x_2, y_3, x_4, y_5, x_6, y_7, x_8, y_9, x_{10}, y_{11}, x_{12}, y_1
h, n, x_{13}, y_{13}, x_{14}, y_{15}, x_{16}, y_{17}, x_{18}, y_{19}, x_{20}, y_{13}
h, n, x_{21}, y_{21}, x_{22}, y_{23}, x_{24}, y_{25}, x_{26}, y_{21}

The SDRC assumes that the polygons are well formed. Specifically, open polygons (Figure 2(a)); polygons with shared edges (Figure 2(b)); polygon



(a) No holes (b) Two holes H1 and H2

Figure 1 Examples of polygons

overlaps (Figure 2(c)); and polygons sharing an edge with a hole (Figure 2(d)) are not permitted. While this assumption of well-formedness is not essential to our disscussion, it enables us to concentrate on spacing and width issues. A minor modification to our design allows the SDRC to check for above malformations. Also, these inconsistencies need to be explicitly checked before one can apply bit map based width and spacing checks.

Let d denote the minimum allowable feature width. Figure 3 gives examples of polygons with width error. Note that many designers do not regard Figure 3(c) as an error unless the distance e is less than d. Our SDRC is easily changed to account for this variation. Note that The polygons of Figure 4 have no width error even though they contain some edges less than d.

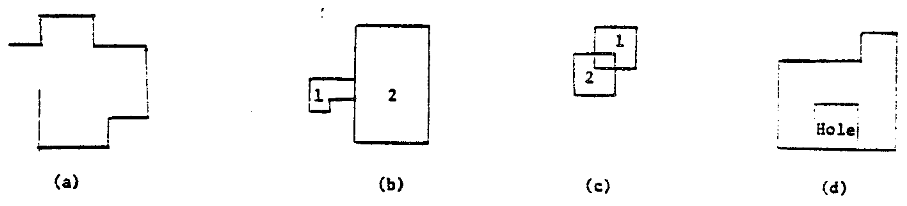


Figure 2 Malformed Polygons

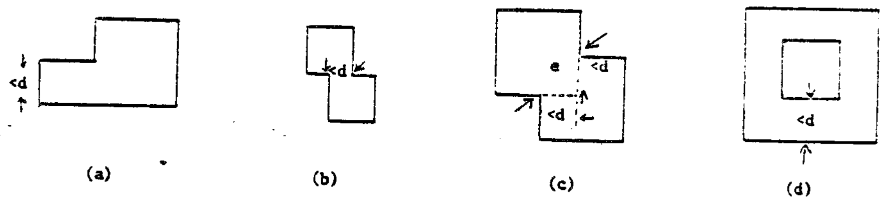


Figure 3 Polygons with width errors

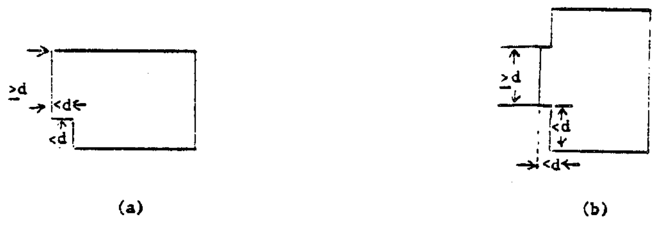


Figure 4 Polygons with no width errors

Let s denote the minimum allowable spacing between polygons. The ploygons of Figure 5 have space errors at the points marked *.

As in the case of Figure 3(c), the configuration of Figure 5(c) is often not considered erroneous unless the distance labeled e is less than s. This change is also easily made in the SDRC design.

3. SDRC Architecture

The SDRC is a hardware device that may be attached to a computer system as a peripheral (Figure 6) or directly to the CPU as in case of a floating point processor.

A block diagram of the SDRC appears in Figure 7. The major components of an SDRC are two systolic sort arrays (SAX and SAY), controllers for these sort arrays, and a design rule checker (DRC). Let us assume the configuration of Figure 6. When design rule checks are to be performed, the CPU sends the compact descriptions of the polygons to the SDRC. This description is transformed into explicit edges by the controlleers for SAX and SAY. Horizontal edges are created by the cotroller for SAX and inserted into SAX. Vertical edges formed by the controller for SAY and inserted into SAY. The sort arrays sort the edges into

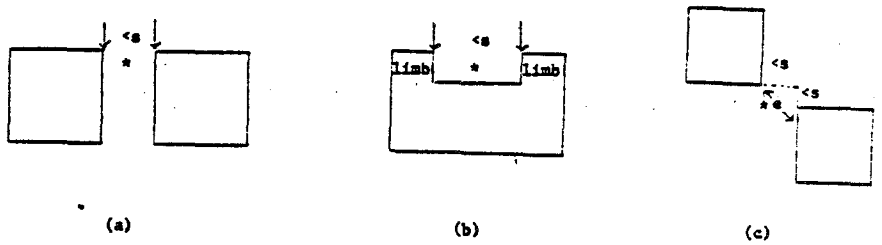


Figure 5

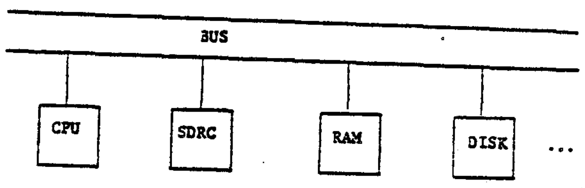


Figure 6

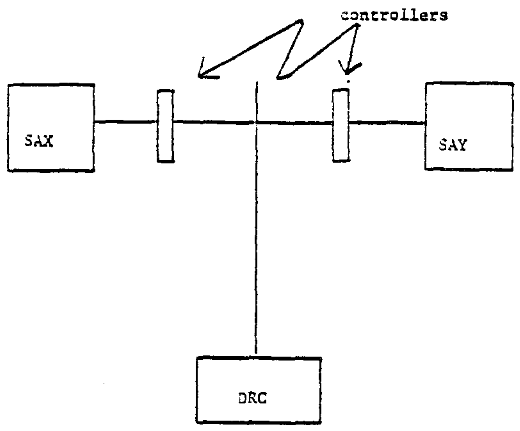


Figure 7 SDRC Architecture

lexical order. Thus, the SAX sorts edges by y - coordinates and within y - coordinate by x - coordinate. Recall that we have assumed that there are no overlapping edges. So, even though every horizontal edge has two x - coordinates, there is a unique lexical ordering for the horizontal edges. Similarly there is a unique ordering for the vertical edges.

As we shall see in the next section, the SAX and SAY are simply systolic priority queues. Consequently, as soon as the edges have been formed and entered into the SAX and SAY, they may be transmitted in lexical order to the DRC. First SAX sends its edges to the DRC, which examines them for width violations in the y direction and spacing violations in the x direction. All detected errors are transmitted back to SAX. Next SAY transmits its edges to the DRC which examines them for width errors in the x direction and spacing errors in the y direction. These errors are sent back to SAY. The errors collected in SAX and SAY may then be communicated back to the CPU.

Clearly, by using two DRCs, the horizontal and vertical edge processing may be effectively overlapped. Further, by providing a data path for the errors to go directly from the DRC to the CPU, the use of the SDRC may be pipelined.

4. Edge Forming

The descriptor for each edge formed in sort array controllers consists of 5 fields as shown in Figure 8. The terminology used in this Figure is with respect to the horizontal edges. y is the y - coordinate for the edge; x_l the left x coordinate; x_r the right coordinate; p# the polygon number; and ud (up-down) is 0 if the interior of the polygon is above this edge and 1 otherwise. In case the DRC sends errors back to the SAX (rather than directly to CPU) then each edge descriptor will have two additional bits to record the error. For vertical edges we may use the terminology of Figure 9 where x is the x coordinate of the edge; y_b and y_t are, respectively, the bottom and top y coordinates; p# is the polygon

y	x_l	x_r	P#	ud

Figure 8

number; and lr (left right) is 0 if the polygon interior is to the left of the edge and is 1 otherwise. The p# field is used only to identify polygons with errors. This field may be omitted and the detected errors can be associated with polygons by performing a search at the end.

Example 1: The edge descriptors for the horizontal edges of the polygon of Figure 10 are :

$y_1, x_1, x_2, 1, 0$
$y_7, x_7, x_8, 1, 1$
$y_{16}, x_{16}, x_{15}, 1, 0$
$y_{10}, x_{10}, x_9, 1, 0$
$y_{11}, x_{11}, x_{12}, 1, 0$
$y_6, x_6, x_5, 1, 1$
$y_{14}, x_{14}, x_{13}, 1, 0$
$y_4, x_4, x_3, 1, 1$

The descriptors for the vertical edges are:

$x_2, y_2, y_3, 1, 0$
$x_8, y_8, y_9, 1, 1$
$x_{12}, y_{12}, y_{13}, 1, 1$
$x_{10}, y_{10}, y_{11}, 1, 1$
$x_{15}, y_{15}, y_{14}, 1, 0$
$x_5, y_5, y_4, 1, 1$

x	y_b	y_t	P#	lr

Figure 9

$x_7, y_7, y_{16}, 1, 1$
$x_1, y_1, y_8, 1, 1$

The transformation from the compact polygon representation to the edge descriptors is relatively straightforward.

5. The Sort Arrays

While the sorting algorithms have been considered for hardware implementation ([THOM82]), priority queues appear tp be best suited for our sort application. Two systolic implementations of priority queues appear in literature. One is due to Leiserson [LEIS79], and the other due to Guibas and Liang [GUIB82]. While design of [GUIB82] is simpler than that of [LEIS79], it permits an insert/delete every four cycles as opposed to once every two cycles for the design of [LEIS79].

The systolic priority queue of [LEIS79] is a linear array of processors (PEs) each having two registers A and B (Figure 11). Each register in the priority queue is large enough to hold edge descriptor. The array of processors pulsates in regular cycles with instructions:

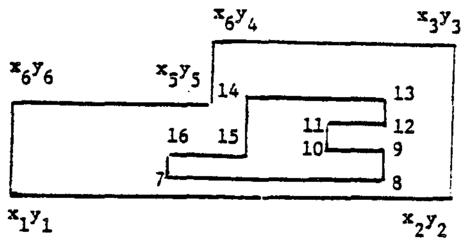


Figure 10

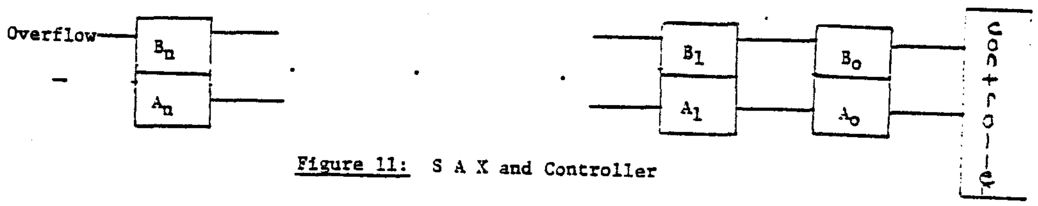


Figure 11: S A X and Controller

Figure 11

1. $B_i \leftarrow B_{i-1}$

2. Order A_{i-1}, A_i, B_{i-1} so that

   $A_{i-1} \leq A_i \leq B_i$

being performed for odd i in odd cycles and for even i (i $\neq 0$) in even cycles. A new edge can be inserted in the array just before every odd cycle by setting B_0 to the edge descriptor and A_0 to $-\infty$.

When all the insertions have been performed, the edges can be extracted in the lexical order by setting A_0 and B_0 to $+\infty$. It takes two cycles to extract each edge. The edges can be sent to DRC one by one as extracted, thereby overlapping the extraction process and DRC operation.

The remaining details for SAX and SAY may be found in [LEIS79].

The DRC

The DRC is invoked once for horizontal edges and once for vertical edges. Since the processing that occurs with horizontal edges is the same as that for vertical edges, our discussion of the DRC is confined to the case of horizontal edges.

As mensioned earlier, when processing the horizontal edges, the DRC checks for width violations in the y direction and spacing violations in the x direction. In addition, the spacing and width checks of Figure 12 are also performed.

The DRC (Figure 13) is a linear systolic array with the same organization as the priority queue of Figure 11. The A and B registers of each PE are ,however larger. In describing the fields of a register, we shall use the notation A[i].x to mean field x of register A of PE i. Each register in the DRC has all the fields

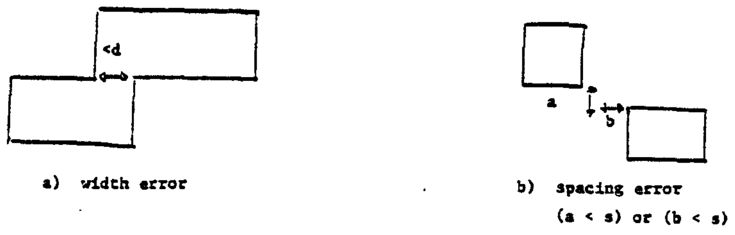


Figure 12

necessory to describe an edge(Figure 8). In addition, the following fields are also present:

PR .. This is a two bit priority field used to control the flow of data in the A and B registers. The four possible values assignable to PR have the following interpretation:

PR = 11: This signifies an empty register. If ud = 0, then this is an empty register to the right of the rightmost edge(i.e. edge 2.1 of Figure 14) in the DRC. If ud = 1, then this is an empty register to the left of the rightmost edge in the DRC.

PR = 10: The register contains an edge that has yet to settle in its place.

PR = 01: This value is possible only for an A register edge. It denotes an edge that has settled.

PR = 00: Denotes an edge for which an error has been detected.

WE .. A 1 bit width error field. It is set to 1 if a width error involving this edge has been detected.

SE .. A 1 bit space error field that is set to 1 when a spacing error involving this edge is detected.

rightok .. A 1 bit field. This is used only for edges with ud = 0. Let X, Y ∈ {A, B}.
X[i].rightok = 1 iff there is a j such that
(X[i].P# = y[j].P# and X[i].x_r = Y[j].x_l and
Y[j].ud = 0)

y_{right} .. Used in conjunction with rightok. Gives the y-value of the edge that satisfies the condition of rightok

leftok .. A 1 bit field that is used only for edges with ud = 1. Let x ∈ {A, B}.
X[i].leftok = 1 iff there is a limb at the leftand of the edge (Figure 5(b)).

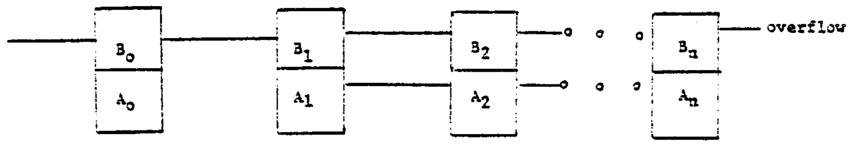


Figure 13

x_{ext} .. When leftok = 1, x_{ext} gives the leftmost point of the edge. Since edges may get split during processing, x_{ext} may not equal x_l (x_l wil be the current left end of the split edge. Since the rightok and y_{right} fields are used only when ud = 0 while the leftok and x_{ext} fields are used only when ud = 1, these fields may use the same physical register space.

It is assumed that all polygons are to be embeded on a rectangular chip (figurre 14). Thus during processing for horizontal edges, the edges 1.1, 1.2 , 2.1, and 2.2 are loaded in the SAX.The edges 1.1 and 1.2 come out of SAX before any other edges in the layout; whereas the edges 2.1, and 2.2 come out in the end. The DRC is initially loaded with the edge 2.1 for processing edges from SAX and the edge 3.1 for processing edges from the SAY.

At the start of each cycle of the DRC, an edge is inserted in B_0. This edge has PR = 01, and WE = SE = 0. Since edges come from SA (or SAY) only once every two cycles, the cycle time of the DRC must be at least twice that of the sort arrays. Once the edge enters the DRC at B_0, it moves towards the right until it finds its correct position with respect to the edges in the A registers. The A register edges are ordered by their x_l values. As the B register edges move to the right, width and spacing checks are performed against the A register edges in the PEs adjacent to the one the edge is to settle into. Once all the hoizontal edges have been entered into the DRC, we set B[0].PR = 11, B[0].UD = 1 and A[0].PR = 11. This will cause the detected errors to move to the left of the DRC from where they may be removed and sent back to SAX or the CPU.

The basic cycle of the DRC is described in procedure cycle.

Before specifying the details of the step 'PROCESS_IN_EACH_PE' ,we describe a few procedures used for this purpose.

6.1 Procedures Used For Width and Spacing Checks

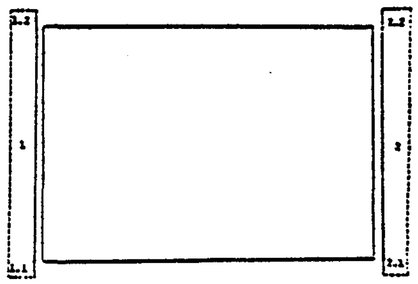


Figure 14

```
procedure cycle
  {pulsating cycle of the systolic DRC}
  repeat
    { shift B edges right }
    for every PE i, i < n do
        B[i+1] ← B[i]
    B[0] ← new edge
    B[0].leftok ← 0
    A[0].(PR, $x_l$, $x_r$, WE, SE, UD) ← (00, -∞, -∞, 0, 0, 1)
    PROCESS_IN_EACH_PE  { described later }
    { shift A edges as needed }
    for every PE i do
        if A[i].PR = A[i+1].PR = 11 and A[i+1].UD = 0
        then { mark i as right of rightmost edge }
          A[i].UD = 0
        end
        for odd i on odd cycles and
              even i on even cycles do
          if A[i].PR > A[i+1].PR
            then A[i] ←→ A[i+1] { interchange edges }
    end
  until false { infinite loop }
end cycle
```

Spacecheck 1.1

This is used by a PE that contains an edge in its A register that is to the right of the edge in its B register. Figure 15 depicts two of the situations when the check is performed.

```
procedure spacecheck 1.1
  if A.$x_l$ - B.$x_r$ < s
  then [A.SE ← 1;B.SE ← 1]
end spacecheck 1.1
```

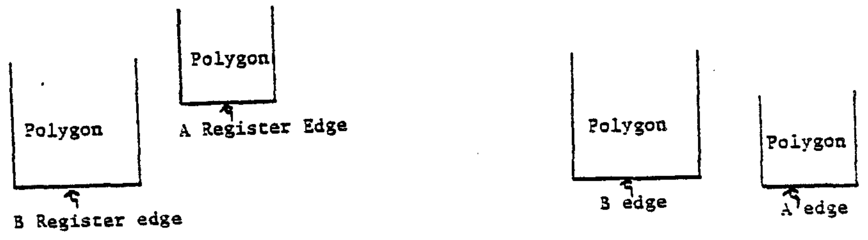


Figure 15

Spacecheck 1.2

This is similar to spacecheck 1.1 except that the B register edge is to the right of the A register edge.

```
procedure spacecheck 1.2
   if $B.x_l - A.x_r < s$
    then [A.SE ← 1;B.SE ← 1]
end spacecheck 1.2
```

Spacecheck2

This is used to check the interlimb distance in polygons (Figure 16). As edges progress through DRC, they may get broken. So, the edge in a register may actually be only a segment of a larger edge. The leftmost point on the original whole edge is 'remembered' in the field x_{ext} which takes the place of the y_{right} (x_{ext} is used when UD = 1 while y_{right} is used when UD = 0).

```
procedure spacecheck2
   if $B.x_r - B.x_{ext} < s$
   then B.SE ← 1
end spacecheck2
```

Widthcheck1

This is used when the A and B register edges in a PE belong to the same polygon; have some overlap; and A.UD = 0 and B.UD = 1. Figure 17 depicts a possible situation.

```
procedure widthcheck1
   if B.y -A.y < d
```

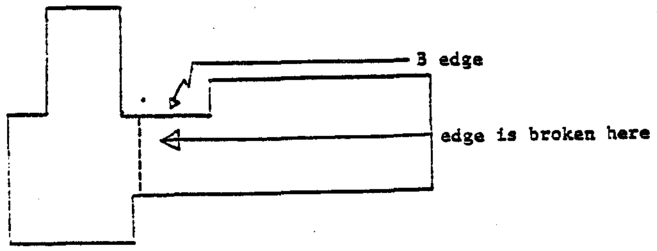


Figure 16 Interlimb distance

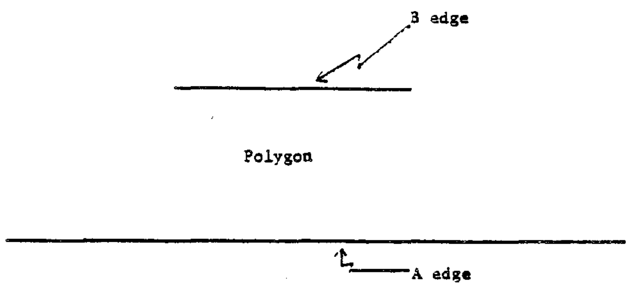


Figure 17

```
    then [A.WE ← 1; B.WE ← 1]
end widthcheck1
```

Widthcheck2

The widthcheck performed by this procedure is shown in Figure 18. The PE that performs this check has edges in A and B registers that have the same polygon numbers; A.UD = 0 and B.UD = 1; and A.rightok = 1.

```
procedure widthcheck2
    if $B.y - A.y_{right} < d$
    then B.WE ← 1
end widthcheck2
```

6.2 PROCESS_IN_EACH_PE

In this step of the cycle, each PE examines the edges in the A and B registers and performs the checks based on this. In order to understand the edge processing procedure to be outlined shortly, it is necesscry to keep the following in mind.

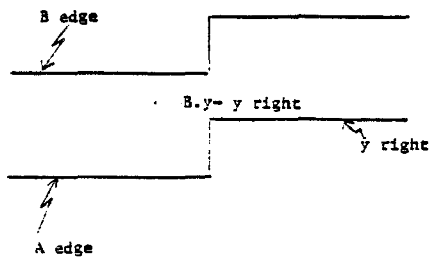


Figure 18

1. Edges may settle only in A registers. Thus,

   B.PR 01 for any PE.

2. Edges that have not yet settled must do so by moving to the right via B registers. So, the case A.PR = 10 is not possible.

3. Settled edges are ordered by their x values left to right in the A registers. The sequence of settled edges (i. e., PR = 01) may be interspersed with error edges (i. e., PR = 00) and empty edges (i. e., PR = 11).

4. A polygon edge may get split during processing. Figure 19(a) shows a polygon with a hole in it. When edge e is the B edge in the PE containing the edge acd in its A register, the acd adge is split into the three segments a, c, and d. The segments a and c are discarded. In the case of polygon in Figure 19(b), the edge e causes the edge ac to be split into segments a and c. The segment ais discarded as no new errors with respect to this segment are possible. All errors detected for the edge are retained by the segment.

In general, edge splits and discards are carried out so as to ensure that the set of active edges (i. e., PR= 01 or 10) have no overlap of their x coordinates.

The exact mechanism by which width and spacing errors are detected is best described using algorithmic notation below.

```
case A.PR of
  00 : { A edge has an error; do nothing }

  10 : { A adge hasn't settled. This is not possible. Only B edges
       may have PR = 10 }
```

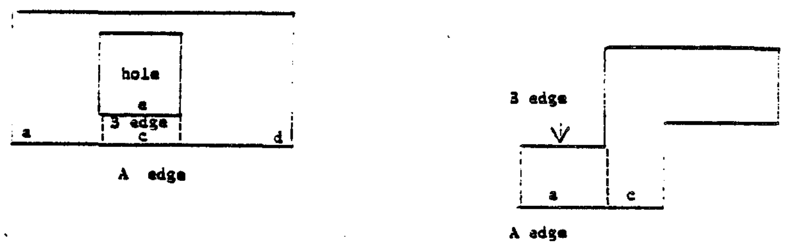


Figure 19

```
11 : { A register is empty }

   case B.PR of
     00: A ←→ B  { Move error edge to
                   empty A register }

     01: { Not possible as edges can
          settle only in A register }

     10: if A.UD = 0
        then {No edges to the right of PE }
        [B.PR ← 01; A ←→ B]
        { B edge must settle here }

     11: { do nothing }
   end case

01 : { A edge is in its correct place }

   casc B.PR of
     11 : { do nothing }

     00 and 01 :{ not possible }

     10 : case A.UD of

          0:

   { At this point A.PR = 01, B.PR = 10, A.UD = 0.
    The interior of the polygon is above the edge A.}
```

```
{ Determine the relationship between the A and B edges }
case

  1: $A.x_l \geq B.x_r$:

    { We have the situation of Figure 20 }
    if B.UD = 0
    then {Figure 20(a)}
      [if $B.x_r = A.x_l$
      then { By assumption on the polygons
                (Figuure 2) B.p# = A.p# }
        [B.rightok ← 1; $B.y_{right}$ ← A.y]
      else { B.P# <> A.P# or B and A are
                from two limbs of the same polygon}
        spacecheck1.1
      endif ]
    endif
    { This is B's place to settle }
    A.PR ← 10; B.PR ← 01; A $\overset{\bullet}{\longleftrightarrow}$ B
    { Note that when B.UD = 1, no checks need
     be performed as relevant checks were
     performed when the A edge was settle }
```

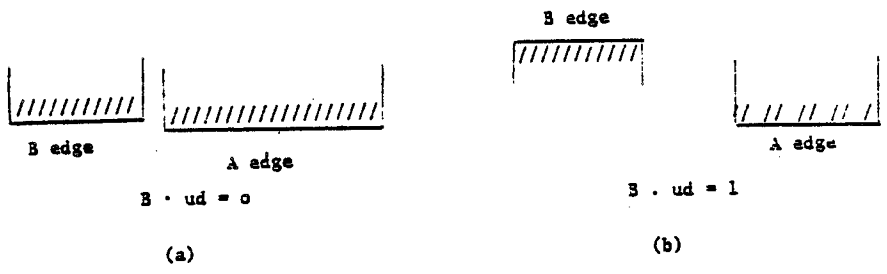


Figure 20

```
2: A.$x_r$ ≤ B.$x_l$:

  { This situation is depicted in Figure 21 }
  if B.UD = 0
  then {Figre 21(a)}
    if A.$x_r$ = B.$x_l$
    then { By assmption on polygons (Figure 2)
                B.p# = A.p# }
      [A.rightok ← 1; A.$y_{right}$ ← B.y]
    else spacecheck1.2
    endif
  else {Figure 21(b)}
    if A.$x_r$ = B.$x_l$ and not B.leftok
    then { Figure 21(c). Set leftok and $x_{ext}$
               in case limb test is needed. B edge
               may get split later}
      [B.leftok ← 1; B.$x_{ext}$ ← B.$x_l$]
    endif
    if A.rightok
    then { Figure 21(c). A width check is needed. }
      [if (B.y - A.y < d) and
          (B.$x_l$ - A.$x_r$) < d
       then B.WE ← 1]
    endif
  endif

3: else : { A and B edges have some overlap and so
            must be part of the same polygon. Hence
            B.UD = 1.Figure 22 - 26 show some cases.
            Note that A.$x_l$ ≤ B.$x_l$ < A.$x_r$.

            The case B.$x_l$ < A.$x_l$ is not
            possible as this would have caused
            caused B edge to be split earlier,
            leaving B.$x_l$ = A.$x_l$  }
```

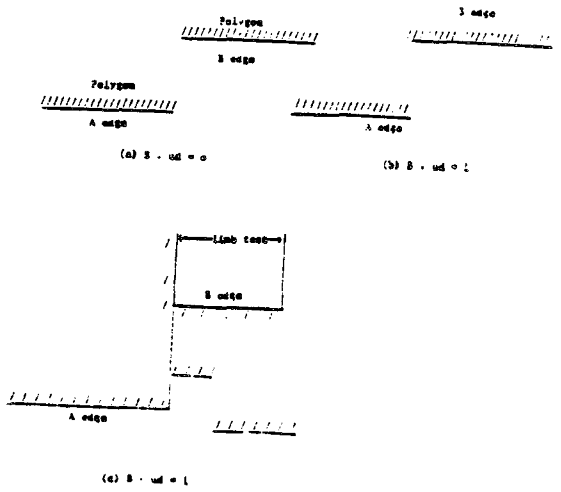


Figure 21

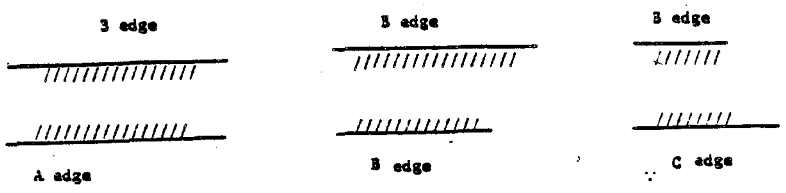


Figure 22

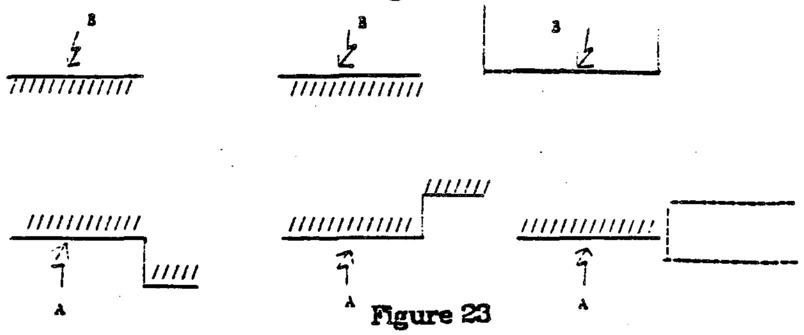


Figure 23

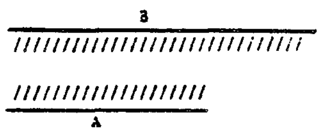


Figure 24

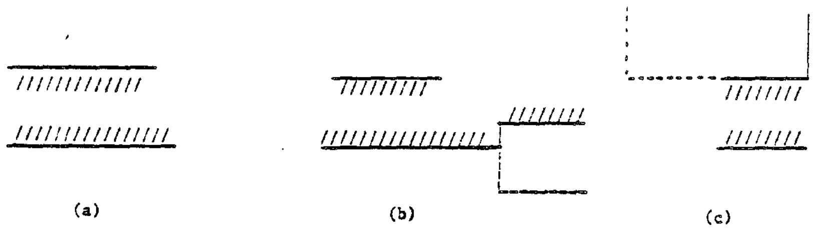


Figure 25

```
3.1: $A.x_l = B.x_l$ : {Figure 22}

  3.1.1: $A.x_r = B.x_r$ :{Figure 22(a)}

      if A.rightok
      then { Figure 23 }
         [widthcheck2
          if B.leftok
          then { Figure 23(c) }
            spacecheck2]

      { change status of A edge }
      if A.WE or A.SE
      then  A.PR ← 00
      else [A.PR ← 11; A.UD ← 1]

  3.1.2: $A.x_r < B.x_r$ :{Figure 24}

      { split B edge and put left part in A;
        note that if there is a left limb of B,
        B.leftok and $B.x_{ext}$ were set in case 2.
        (see Figure 21(b)}
      A.UD ← 1; A.y ← B.y; $B.x_l$ ← $A.x_r$
```

```
3.1.3: $A.x_r > B.x_r$ : {Figure 25}

    if A.rightok
    then {Figure 25(b)}
      [ widthcheck2]
    if B.leftok
    then {Figure 25(c)}
      [ spacecheck2 ]

    { split A edge }
    $A.x_l \leftarrow B.x_r$
    { This is B's place to settle }
    $A \leftrightarrow B$; $A.PR \leftarrow 01$; $B.PR \leftarrow 10$


3.2: $A.x_l < B.x_l$ : { Figure 26 - 28}

  { There is an upward limb at the left of B}
  $B.leftok \leftarrow 1$; $B.x_{ext} \leftarrow B.x_l$

  3.2.1: $A.x_r = B.x_r$ :{Figure 26}
      if A.rightok
      then {Figures 26(a) and (b)}
          [widthcheck2; spacecheck2]
      endif
      { split A edge }
      $A.x_r \leftarrow B.x_l$; $A.rightok \leftarrow 0$
```

```
3.2.2: $A.x_r > B.x_r$ : {Figure 27(a) and (b)}
    { The situation depisted in Figure 27(c)
      is not possible as the A edge would
      have been split at $B.x_r$ when
      edge c went over it }
    **if** A.rightok
    **then** { Figure 27(a)}
        widthcheck2
    **endif**
    spacecheck2 { must be a limb }
    { split A edge discarding the segment
      $A.x_l$ to $B.x_l$ }
      $A.x_l \leftarrow B.x_r$

3.2.3: $A.x_r < B.x_r$ : {Figure 28}

    { split A and B retaining segments
      b and d, (Figure 28)}
    A.rightok ← 0
    $(A.x_r, B.x_l) \leftarrow (B.x_l, A.x_r)$

{ This ends the case A.UD = 0}

{ Begin last case to consider }
4: A.UD = 1:
  { At this time, A.PR = 01, B.PR = 10, and A.UD = 1}
```

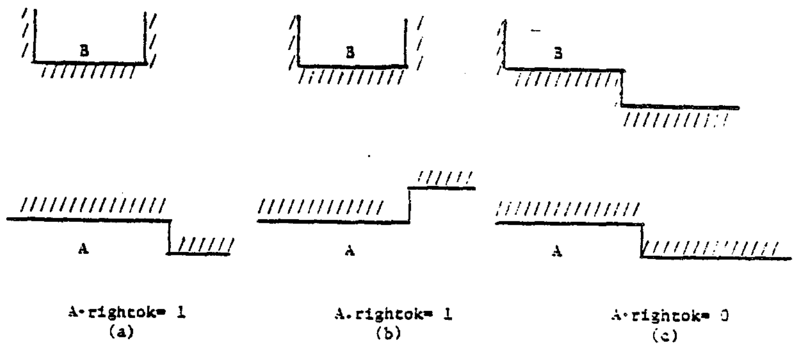


Figure 26

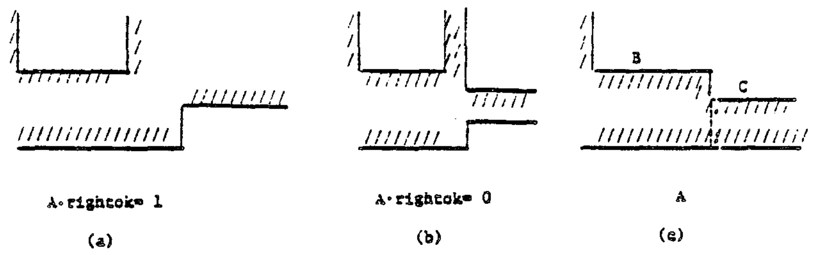


Figure 27

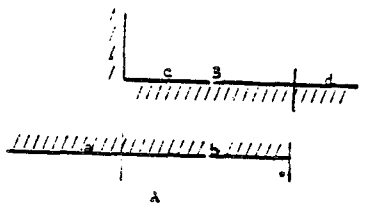


Figure 28

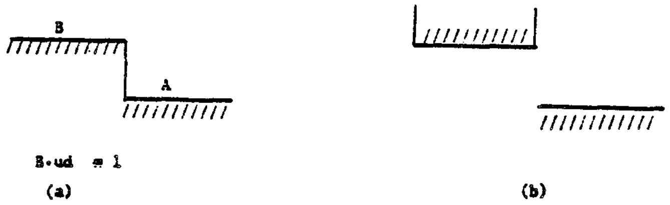


Figure 29

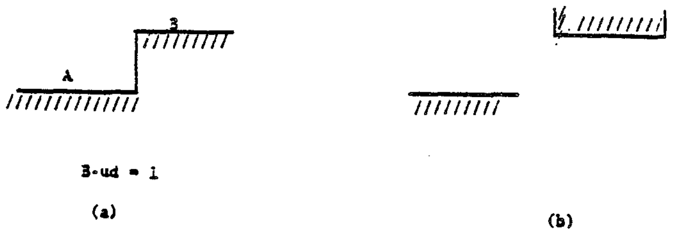


Figure 30

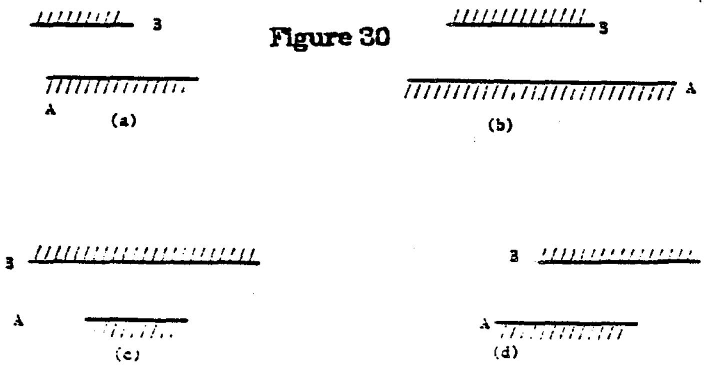


Figure 31

```
if $B.y - A.y \geq s$

then { remaining edges are too far
        from A to cause errors}
    [if  A.WE or A.SE
     then A.PR $\leftarrow$ 00
     else  A.PR = 11]
else
case

   4.1:$A.x_l \geq B.x_r$: {Figure 29}
       if not [(B.UD=1 and $B.x_r = A.x_l$)
            or $A.x_l - B.x_r \geq s$]
       then
           [A.SE $\leftarrow$ 1; B.SE $\leftarrow$ 1]
       { This is B's place to settle }
       A $\leftrightarrow$ B; A.PR $\leftarrow$ 01; B.PR $\leftarrow$ 10

   4.2:$A.x_r \leq B.x_r$ : {Figure 30 }
        if not [B.UD = 1 and $B.x_l = A.x_r$
            or $B.x_l - B.x_r \geq s$]
        then
           [A.SE $\leftarrow$ 1; B.SE $\leftarrow$ 1]

   4.3:else: { Partial overlap (Figure 31). So, B.UD = 0}
          A.SE $\leftarrow$ 1; B.SE $\leftarrow$ 1
          case

          4.3.1: $B.x_r < A.x_r$:{Figure 31(a) and (b) }
                { split A}
                $A.x_l \leftarrow B.x_r$
                A $\leftrightarrow$ B; A.PR $\leftarrow$ 01; B.PR $\leftarrow$ 10

          4.3.1: $B.x_r \geq A.x_r$ :{Figure 31(c) and (d)}
                A.PR $\leftarrow$ 00
                { The remaining spacing errors involving the left
                  part of the A edge in Figure 31(b) and (d) will
be detected when handling vertical edges }
                **end case**
        **end case**
    **end**
```

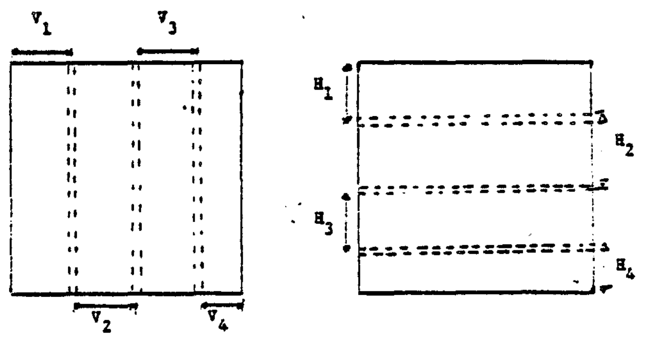
6.3 Performance

Under the assumption that the sort arrays and DRC are large enough to accommodate all the edges, the sort time and the DRC time is linear in the number of the edges in all the polygons. Furthermore the time spent extracting the errors from the sort arrays is effectively overlapped with the DRC processing.

In practice, of course, no matter how large the sort arrays and DRC, there will be times when the number of the edges to be handled will exceed the capacity of the systolic arrays. In these circumstances, the layout may be partitioned into vertical slices for SAX and horizontal slices for SAY (Figure 32). By ensuring that adjacent slices overlap by at least max{s, d} we ensure that no erroneous reporting will occur. The checks may then be performed for each slice independently.

7. Conclusions

We have demonstrated the potential of systolic architectures in the design automation field. While our design of a DRC several simplifying assumptions,



a) Partitioning into vertical slices

b) Partitioning into horizontal slices

Figure 32

these may be relaxed at the expense of the increased complexity. In particular, the assumptions about well formed polygons (Figure 2) and Manhatten vs Euclidean distance (Figure 5)are trivially removabe.

8. References

[ABRA82] M. Abramovici, Y. H. Levendel, and P. R. Menon,"A Logic Simulation Machine" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 65-73*

[BLAN81] Tom Blank, Mark Stefik, William vanCleemput "A Parallel Bit Map Processor Architecture for DA Algorithms" *ACM IEEE Eighteenth Design Automation Conference Proceedings pp 837-845*

[DENN82] M. M. Denneau, "The Yorktown Simulation Engine" *CM IEEE Nineteenth Design Automation Conference Proceedings pp 55-59*

[GUIB82] Leo J. Guibas, Frank M. Liang, "Systolic Stacks, Queues and Counters" *1982 Conference on advanced Research in VLSI, M. I. T.*

[KRON82] E. Kronstadt and G. Pfister, "Software Support for the Yorktown Simulation Engine" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 60-64*

[KUNG82] H. T. Kung, "Let's Design Algorithms for VLSI Systems" *CMU-CS-79-151 Department of Computer Science, Carnegie Mellon Universit*

[MUDG82] T. N. Mudge, R. A. Ratenbar, R. M. Lougheed, and D. E. Atkins, "Cellular Image Processing Techniques for VLSI Circuit Layout Validation and ROuting" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 537-543*

[NAIR82] R. Nair, S. Jung, S. Liles, and R. Villani, "Global Wiring on a Wire Routing Machine" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 224-231*

[LEIS79] C. E. Leiserson, "Systolic Priority Queues" *Proceedings of Conference on VLSI: Architecture, Design, Fabrication California Institute of Tachnology Jan 79 pp 199-214*

[PFIS82] G. F. Pfister, "The Yorktown Simulation Engine, Introduction" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 51-54*

[SEIL82] L. Seiler, "A Hardware Assisted Design Rule Check Architecture" *ACM IEEE Nineteenth Design Automation Conference Proceedings pp 232-238*

[THOM82] C. D. Thompson, "The VLSI Complexity of Sorting" *UCB ERL M82/5 Electronics Research Laboratory, College of Enineering, Berkeley, California*

[UEDA83] Kazuhiro Ueda, Tsutomu Komatsubara and Tsutomu Hosaka, "A Parallel Processing Approach for Logic Module Placement" *ACM IEEE Transactions on Computer Aided Design Vol. CAD-2 No.1 Jan.83 pp 39-47*

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER TR 83-13	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) "A Systolic Design Rule Checker"		5. TYPE OF REPORT & PERIOD COVERED July 1983 Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Rajiv Kane, Sartaj Sahni		8. CONTRACT OR GRANT NUMBER(s) N00014-80--650
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department University of Minnesota 136 Lind Hall, 207 Church Street. S.E. Mpls,MN.		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of the Navy Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE July 1983
		13. NUMBER OF PAGES 26
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Design Rule Checks, feature width, spacing rectilinear geometries, systolic systems.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We develop a systolic design rule checker (SDRC) for rectilinear geometries. This SDRC reports all width and spacing violations. It is expected to result in a significant speed up of the design rule check phase of chip design.

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

MODELLING OF CHEMICAL VAPOR DEPOSITION REACTORS FOR THE FABRICATION OF MICROELECTRONIC DEVICES

Microelectronic and Information Sciences Center

Technical Report #04

K.F. Jensen
Department of Chemical Engineering
and Materials Science
University of Minnesota

MODELLING OF CHEMICAL VAPOR DEPOSITION REACTORS FOR THE FABRICATION OF MICROELECTRONIC DEVICES

MEIS Technical Report #04

K.F. Jensen

The research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

C 1984

ABSTRACT

The modelling of reactors for chemical vapor deposition (CVD) of thin solid films is reviewed. Both production and experimental reactor systems are considered. The discussion of production equipment centers on conventional horizontal, barrel, and tubular reactors while the treatment of experimental systems focuses on rotating disk and stagnation point flow reactors. The analogies between CVD and heterogeneous catalysis are pointed out and also illustrated through a modelling study of the multiple-wafer-in-tube low pressure CVD reactor. The reactor model is shown to be equivalent to a fixed bed reactor model. The deposition of polycrystaline Si from SiH_4 is considered as a specific example. The model predicts experimental observations and provides quantitative comparison with experimental data from reactor studies.

Chemical vapor deposition (CVD) is one of the fundamental processes in the microelectronics industry where it is used to deposit stable, thin solid films. The terms "chemical" and "vapor" derive from the fact that the solid film is deposited by chemical reactions of gaseous components. This distinguishes CVD from physical deposition processes such as sputtering and gives the process its flexibility. In the microelectronics industry CVD is used to grow a wide range of thin solid films which serve as dielectronics, conductors, passivation layers, dopant sources, and oxidation barriers. These films in turn form the basis for many different electronic devices including semiconductor memories, semiconductor lasers, light detectors, power transistors, and microprocessors. CVD is well established in the growth of the basic silicon derived electronic materials: polycrystalline Si, epitaxial (doped) Si, SiO_2, and Si_3N_4. In addition, the process is gaining importance in the formation of the next generation of materials based on III-V compounds (e.g. GaAs, InP, and AlGaAs) which currently find use in optoelectronic devices.

Chemical vapor deposition is not restricted to the microelectronics industry. It is the key process in the fabrication of optical fibers where it enables grading of the refractive index as a function of the radial position in the fiber (1). In the manufacturing industry the technique provides coatings with special properties such as high hardness, low friction, and high corrosion resistance. Examples of CVD reactions and processes are given in Table 1.

Table 1

TYPICAL CHEMICAL VAPOR DEPOSITION SYSTEMS

System	Overall Reaction
Halide transport	$WF_6 + H_2 \rightarrow W$ $SiCl_4 + H_2 \rightarrow Si$
Organometallic CVD	$Al(C_4H_9)_3 + H_2 \rightarrow Al$ $Ga(CH3)_3 + AsH_3 \rightarrow GaAs$
Low pressure CVD	$SiH_4 \rightarrow Si$ $SiH_4 + N_2O \rightarrow SiO_2$ $SiH_4 + NH_3 \rightarrow Si_3N_4$
Plasma enhanced CVD	$SiH_4 \rightarrow Si:H$ (amorphous) $SiH_4 + NH_3 \rightarrow Si_3N_4$
Photochemical vapor deposition	$SiH_4 + N_2O \xrightarrow[Hg]{h\nu} SiO_2$ $SiH_4 + NH_3 \xrightarrow[Hg]{h\nu} Si_3N_4$

The widespread use of CVD is due to its many advantages. (i) Gaseous reactants are easier to handle and keep pure than are liquids or melts. (ii) The substrate can be cleaned before deposition by a slight etch with a reactive gas. (iii) The film is formed at temperatures well below its melting point, especially in plasma or photo-enhanced CVD, where film temperatures typically are a few hundred degrees. (iv) The growth of an epitaxial layer on top of another single crystalline material with differing lattice parameters (i.e. heteroepitaxy) can be realized by varying the composition of the growing film so that the lattice parameters change smoothly across the interface between the two materials. (v) Similarly, donor/acceptor concentrations in the film may be controlled by varying the gas phase composition. (vi) In contrast to sputtering methods, CVD provides good step coverage, which is particularly important since patterns are etched and subsequently covered by new layers during the processing of a device.

To effectively exploit the above advantages it is necessary to have a detailed understanding of the chemical and physical rate mechanisms underlying CVD. These encompass mass transport in the gas phase by convection and diffusion, homogeneous as well as heterogeneous reactions, and heat transfer by both convection and radiation. The situation is further confounded by complex flow fields and boundary conditions. The formation of SiO_2 from $SiCl_4$ exemplifies the importance of knowing the role of each rate process. The heterogeneous reactions are necessary to grow thin Si films for bipolar devices, while homogeneous nucleation of SiO_2 is essential in the production of optical fibers.

There exists a considerable literature on CVD (2) but relatively few attempts have been made to combine chemical and physical rate processes to give a complete representation of the deposition process. Most CVD studies have focused on demonstrating the growth of a particular material or crystal structure. However, the combined analysis is necessary in order to design CVD reactors where it is possible to deposit thin films of constant thickness and uniformity across an entire wafer. This is particularly important in the realization of submicron feature sizes for Very Large Scale Integrated Circuits. The further development of devices based on III-V compounds also depends on CVD reactor design improvements since the composition and thus the electronic properties of these materials vary considerably with process conditions.

Chemical vapor deposition and heterogeneous catalysis share many kinetic and transport features, but CVD reactor design lags the corresponding catalytic reactor analysis both in level of sophistication and in scope. In the following we review the state of CVD reactor modelling and demonstrate how catalytic reactor design concepts may be applied to CVD processes. This is illustrated with an example where fixed bed reactor concepts are used to describe a commercial "multiple-wafers-in-tube" low pressure CVD reactor.

CVD Reactors

Figure 1 illustrates conventional CVD reactors. These reactors may be classified according to the wall temperature and the deposition pressure. The horizontal, pancake, and barrel reactors are usually cold-wall reactors where the wall temperatures are considerably cooler than the deposition surfaces. This is accomplished by heating the susceptor by external rf induction coils or quartz radiant heaters. The horizontal multiple-wafer-in-tube (or boat) reactor is a hot-wall reactor in which the wall temperature is the same as that of the deposition surface. Therefore, in this type of reactor, the deposition also occurs on the reactor walls which presents a potential problem since flakes from the wall deposit cause defects in the films grown on the wafers. This is avoided in the cold-wall reactors, but the large temperature gradients in those reactors may induce convection cells with associated problems in maintaining uniform film thickness and composition.

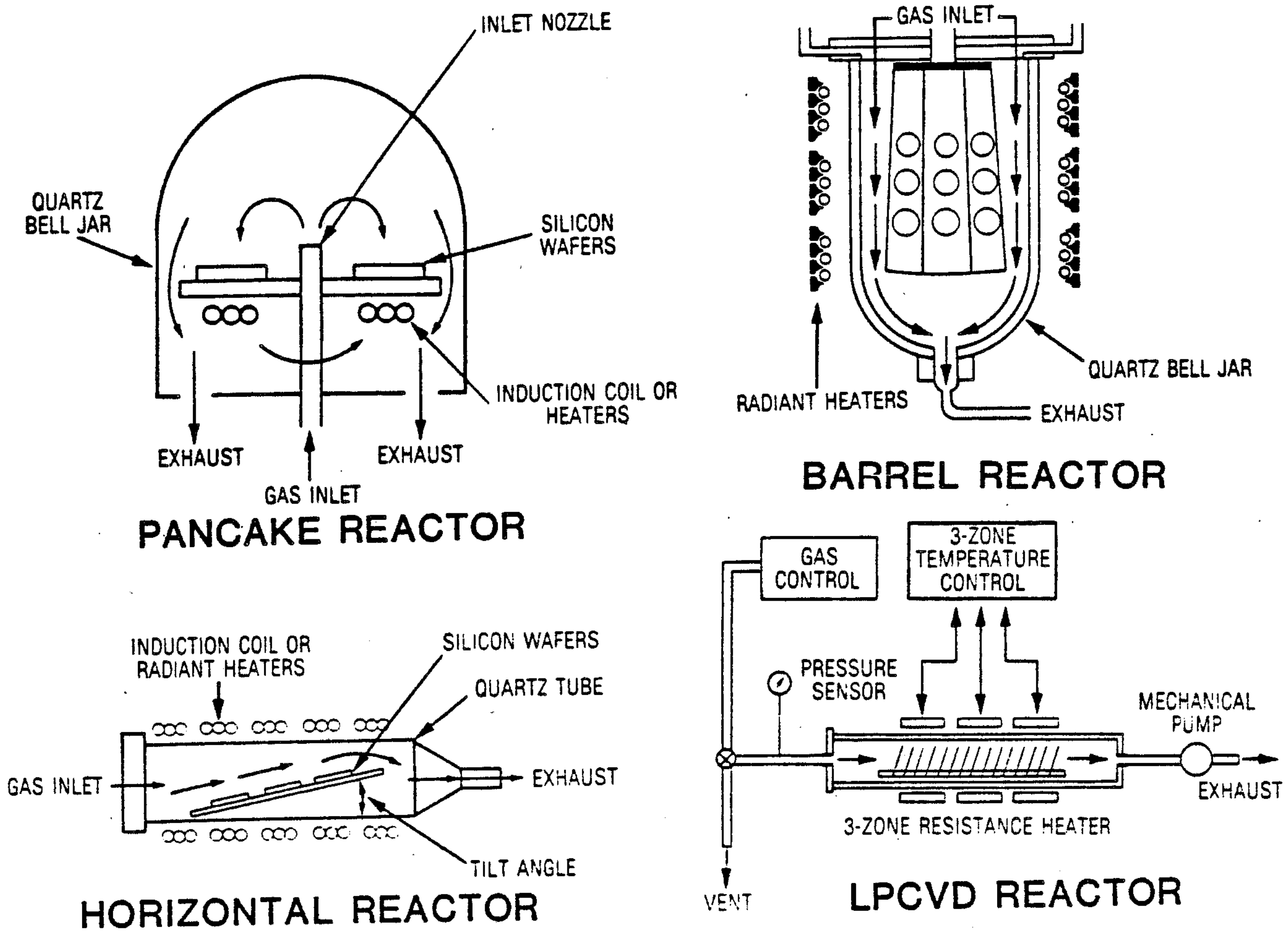


Figure 1. Typical CVD Reactors.

There have been a number of modelling studies; in particular horizontal and barrel reactors have been considered. However, the usefulness of these models in the optimization of reactor design and operating conditions is limited by many simplifying assumptions which bring empirical constants into the modelling equations. Shepherd (3) described the deposition of Si from $SiCl_4$ in a horizontal reactor assuming a parabolic velocity profile, linear temperature variation, and diffusion to the susceptor surface while neglecting the change in gas phase concentration in the flow direction due to the depletion of reactants. Rundle (4,5) included this axial depletion of reactant, but assumed a plug flow and neglected temperature variations.

Bradshaw (6) and Eversteyn et al. (7,8) considered a stagnant layer of fluid adjacent to the susceptor coupled with a well-mixed main flow region between the upper end of this layer and the reactor wall. Eversteyn et al. combined the diffusion equation for the stagnant layer with a plug flow model for the main flow region. An empirical relation was used to predict the thickness of the layer as a function of the gas velocity. The major result of this so-called stagnant layer model was the prediction that if the susceptor were tilted at a small angle to the horizontal, the uniformity of silicon growth rates through the reactor would be substantially improved. This improvement is due to a stabilization of the flow and a thinning of the boundary layer. The latter effect only enters into the modelling equations through the empirical correlation of layer thickness and gas velocity. In fact, the stagnant layer model incorrectly predicts that the reactor efficiency passes through a minimum and then increases with increasing gas velocity while the efficiency should approach zero in the limit of very large flow rates.

The stagnant layer concept originated from flow visualization experiments with TiO_2 particles showing an almost particle free layer close to the susceptor and natural convection cells in the main gas stream. This was interpreted as evidence for a stagnant layer, but because of the large temperature gradients in the reactor, the particle free layer more likely arises from thermodiffusion of TiO_2 particles away from susceptor (9). Takahashi et al. (10) have also observed natural convection structures in a horizontal CVD reactor. They present finite difference simulations of velocity and gas phase concentration profiles for pure laminar flow and for the spiralling flow caused by natural convection effect. Unfortunately, as was the case with previous investigators, they limit the analysis to cases where the surface reaction rate is very large so that mass transfer to the surface controls the deposition process. However, both kinetic and transport effects are important in the majority of CVD processes.

The two major modelling approaches based on either boundary layer approximations or well developed laminar flow have also been applied to the barrel reactor. Dittman (11) used a Chilton-Colburn analogy for flow over a flat plate to predict Si growth rates, but such an approach is limited to the mass transfer controlled regime and the specific reactor for which the correlation is developed. Fujii et al. (12) split the annular flow region into three concentric layers where convection dominates in the central one and diffusion governs the transport in both the layers next to the wall and the susceptor. By adjusting the relative thickness of the layers the authors could match experimental data. Again, this type of approach only applies to the reactor at hand and provides no new insight into the deposition process. Manke and Donaghey (13) assumed fully developed laminar flow and treated the mass transfer in the annular region between reactor wall and susceptor as an extended Graetz problem neglecting all kinetics.

The so far most complete analysis of the barrel reactor is given by Juza and Cermak (9,14) making use of the 2D momentum, mass, and energy balances to demonstrate the development of the transverse velocity, concentration, and temperature profiles along the susceptor. They include overall surface kinetics for the surface reaction: $SiCl_4 + 2H_2 \rightarrow Si + 4HCl$, and by comparison with the previous studies they demonstrate the importance of kinetic effects. They also show that thermodiffusion should be included because of the steep thermal gradients and the large difference in molecular weight between H_2 and Si-species. The gas phase reactions associated with Si deposition from $SiCl_4$ (cf. (15)) are not included in the analysis although they influence the growth rate predictions.

Experimental Reactor Systems

There have been several experimental studies using the horizontal reactor, but most of these have been limited to simple measurements of film thickness. Only a few studies have been performed on the gas phase, notably by Sedgwick, Ban, and their coworkers. Sedgwick et al. (16) used laser Raman scattering techniques to measure the transverse concentration and temperature profiles in a horizontal reactor. Ban et al. (15,17) determined gas phase contractions by mass spectrometry. However, the horizontal reactor is not suitable for mechanistic studies of CVD since it is virtually impossible to separate physical and kinetic rate processes for this complex reactor geometry. On the other hand, rotating disk or stagnation point flow reactors are attractive for detailed investigations since the hydrodynamics are characterized. Thus, the use of these configurations makes it possible to decouple the transport effects more accurately from the chemical kinetics.

The rotating disk has long been used to study electrochemical reactions and has also found early use in CVD (18). Sugawara (19) has presented an analysis of epitaxial growth of Si from $SiCl_4$. His treatment includes natural convection but is limited to mass transfer controlled deposition and equilibrium distributions in the gas phase. Pollard and Newman (20) detail Si deposition on a rotating disk treating the multicomponent mass and heat transfer problem and including simultaneous homogeneous and heterogeneous reactions. In addition, the physical parameters vary across the deposition zone. Their treatment clearly demonstrates that both homogeneous and heterogeneous reactions are important in CVD of Si, but they observe some discrepancy between model predictions and experiments presumably because of poorly known kinetic constants and natural convection effects. Hitchman et al. (21,22) also consider the rotating disk reactor and find that the inlet gas flow perturb the ideal fluid flow pattern. They hypothesize that the difficulty of rotating disk CVD experiments to confirm to theoretically predicted flow patterns could be caused by this inlet effect which is not apparent in liquid phase systems.

The stagnation point flow reactor, where the flow impinges on the heated substrate, appears to be an attractive alternative to the rotating disk configuration. There is no ambiguity in the way the gas is introduced into the deposition region. Moreover, natural convection driven instabilities may be avoided by inverting the reactor so that the buoyancy force and the inlet gas velocity point in the same direction. Donaghey (23) discusses the use of the stagnation point flow in crystal growth and reviews several CVD applications. The contributions by Wahl (24,25) seem to be the most significant. He solves the two-dimensional momentum, species, and energy balances for an incompressible gas flow in an enclosed stagnation point flow reactor by using finite differences. The model predictions compare well with flow visualization experiments.

The conventional stagnation point configuration, where the gas flows downward, has been used in several CVD studies. However, none of these researchers seem to have accounted for the fact that in a region close to the flow axis, the surface of the stagnation plate is "equally accessible" to transport implying that the film growth rate is independent of the radial position. Graves and Jensen (26,27) have recently analyzed this case detailing the transformation of the general partial differential modelling equations to a set of ordinary differential equations. These are in turn solved by a Galerkin finite element technique. The authors consider various general chemical mechanisms for film growth to demonstrate how the stagnation point flow configuration may be used to distinguish whether homogeneous or heterogeneous reactions dominate the overall deposition process.

Low Pressure CVD Reactors

Low pressure CVD (LPCVD) has become a dominant process in the growth of thin films of microelectronic materials. It is widely used to deposit thin films of polycrystalline Si, SiO_2, and Si_3N_4. In addition it has been demonstrated for deposition of metals, specifically Al and W. The process is carried out in tubular, hot wall reactors where the wafers are placed perpendicular to the flow direction as illustrated in Figure 2. The very large packing densities that can be realized in LPCVD reactors without adverse

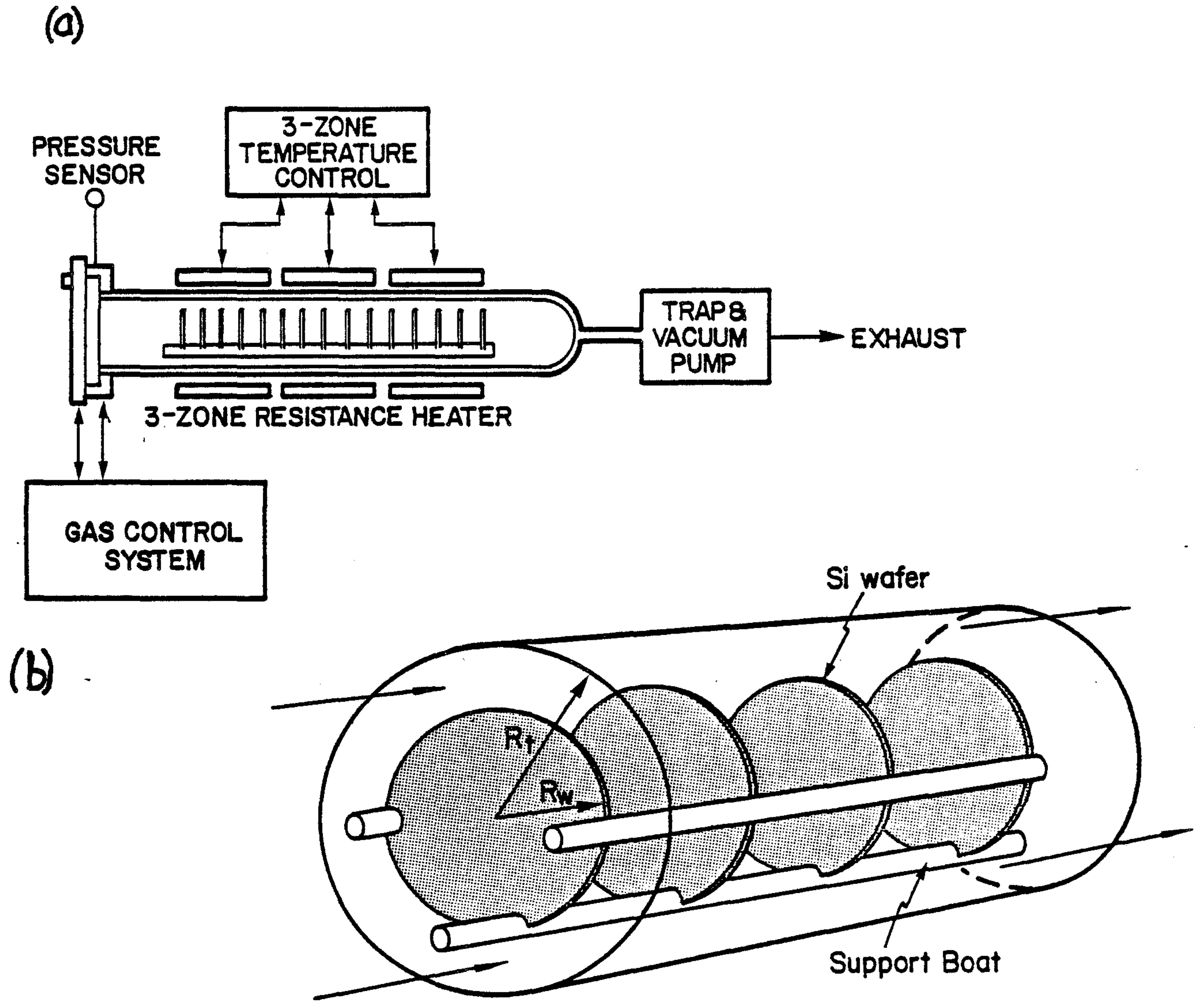


Figure 2. LPCVD Reactor System.

effects in film uniformity are possible because of the reduced pressure (~ 1 torr) where the diffusion coefficients are three orders of magnitude larger than at atmospheric pressure. This implies that the chemical reactions at the surface of the wafers are rate controlling rather than mass transfer processes. Moreover, in spite of the low pressures, rates of deposition in LPCVD reactors are only an order of magnitude less than those obtained in atmospheric CVD since the reactants are used with little or no dilution in LPCVD whereas they are strongly diluted in the conventional cold wall processes reviewed above.

There appear to have been few modelling efforts for hot-wall LPCVD reactors. Gieske et al. (28) and Hitchman et al. (29) present experimental data and discuss flow fields, mass transfer effects, and possible kinetics in rather general terms. A recent model by Kuiper et al. (30) cannot account for diffusion in the spaces between the wafers and the significant volume expansion commonly associated with LPCVD processes. Furthermore, it is restricted to isothermal conditions and plug flow in the main flow region in spite of the large diffusivities associated with LPCVD.

In the following we present a detailed model of the commercial, multiple-wafer-in-tube reactor illustrated in Figure 2. We have selected the LPCVD as an example because of its central role in the microelectronics industry and because it nicely demonstrates the analogies to heterogeneous catalytic reactors, in particular the fixed bed reactor.

LPCVD Reactor Model

The modelling approach behind the LPCVD reactor model is not restricted to any specific deposition kinetics. However, to limit the algebraic complexity and to be able to compare model predictions with experiments we consider the simplest major deposition process, the deposition of polycrystalline Si from SiH_4. The model is based on the following kinetic mechanism:

$$SiH_4(g) \rightleftarrows SiH_2(g) + H_2(g)$$

$$SiH_2(g) \rightleftarrows SiH_2(ads)$$

$$SiH_2(ads) \rightarrow Si(s) + H_2(g)$$

and it is assumed that the surface reaction is rate controlling and follows the rate expression:

$$\mathcal{R} = \frac{k\, P_{SiH_4}}{1 + K_1 P_{H_2} + K_2 P_{SiH_4}} \qquad (1)$$

This particular form may be justified by the experimental observations: the rate is inhibited by H_2, first order in SiH_4 at low SiH_4 partial pressures and approaches zero order in SiH_4 at high partial pressures (31). The details of the development of the model are given in ref. (32) and follow the same approach as commonly used in modelling fixed bed reactors. The complete description of the physicochemical processes in the tubular LPCVD reactor entails partial differential equations in the radial and axial coordinates. However, at low pressures the time scale associated with deposition is of the same order or larger than that for diffusion so that one may assume perfect radial mixing in the narrow annular flow region. Similarly, since the wafer spacing is small compared to the radius of the wafer, we may neglect variations in the axial direction within each cell formed by adjacent wafers. Under those conditions the reactor equation takes the form:

$$D_o \frac{d}{dz} \left(\psi \frac{d\chi}{dz}\right) + v_o \frac{d\chi}{dz} - \frac{2}{(R_t^2 - R_w^2)c_{10}} \left[R_t(1+\alpha) + \frac{R_w^2}{\Delta} \eta\right] \mathcal{R}(\chi(z)) = 0 \qquad (2)$$

where $\psi = -\frac{(1+\varepsilon)}{(1+\varepsilon\chi)^2} \left(\frac{T}{T_o}\right)^{0.65}$

and the boundary conditions are:

$$D_o \left.\frac{d\chi}{dz}\right|_0 = v_o(1 + \chi\varepsilon)\chi \quad ; \quad \left.\frac{d\chi}{dz}\right|_L = 0 \qquad (3)$$

Here χ is the conversion of SiH_4. ψ combines the effect of the molar expansion in the deposition process as well as the change in the volumetric flow and the dispersion coefficient, D, with temperature. At low pressures and small Re in LPCVD reactors the dispersion occurs mainly by molecular diffusion, therefore, we have used $(D/D_o) = (T/T_o)^{1.65}$. ε is the expansion coefficient and the stoichiometry implies that $\varepsilon = (x_1)_0$, the entrance mole fraction of SiH_4. The expansion coefficient, ε is introduced as originally described by Levenspiel (33). The two reaction terms refer to the deposition on the reactor wall and wafer carrier and that on the wafers, respectively. The remaining quantities in these equations and the following ones are defined at the end of the paper. The boundary conditions are equivalent to the well known Danckwerts' boundary conditions for fixed bed reactor models.

The reaction and diffusion of SiH_4 between the wafers is governed by the continuity equation:

$$D_o c_o \left(\frac{T}{T_o}\right)^{0.65} \frac{1}{r} \frac{d}{dr} \left(\frac{r}{1+\varepsilon\chi} \frac{d\chi}{dr}\right) + \frac{2}{\Delta} \mathcal{R}(\chi(r)) = 0 \qquad (4)$$

with the boundary conditions:

$$\chi(r = R_w) = \chi_b(z) \quad ; \quad \left.\frac{d\chi}{dr}\right|_0 = 0 \qquad (5)$$

method has been applied successfully to fixed bed reactor problems (37 and references within) and the solution to the LPCVD reactor case is similar (32). Here we turn to the comparison of model predictions with experimental observations.

Model Predictions

Since detailed kinetic data for the deposition of Si from SiH_4 are scarce and somewhat contradictory, the kinetic parameters are evaluated from experimental data in ref. (31). Figure 3 shows the experimental data and the predicted growth rates. The model predicts the decrease in growth rates along the reactor length due to depletion of SiH_4 as well as the inhibiting effect of increased H_2 concentration.

In addition to providing a fit to specific experimental data, a good mathematical model should also predict process performance over a range of conditions. Therefore, to test our LPCVD model, the same kinetic parameters were used in the analysis of deposition data obtained by Rosler (36) with a different LPCVD reactor and at two orders of magnitude larger SiH_4 concentrations. Figure 4 illustrates the measured growth rates and the predicted ones for a ± 15% change in SiH_4 feed rates around the base case of 47 SCCM pure SiH_4 feed. The reactor temperature is increased along the length of the reactor by using a three-zone furnace to produce a nearly uniform thickness profile in the base case. The model accurately predicts the trends in the experimental data. When the feed rate of SiH_4 is stepped up 15%, the growth rates increase along the length of the reactor sice there is an excess of reactant available relative to the base case. A flat thickness profile could clearly be achieved by lowering the temperature gradient along the length of the reactor. Analogously, if the flow of SiH_4 is dropped, the rates of deposition tail off. The slight wavelike appearance of the model predictions arises from the compensation of decreasing reactant by a piecewise linearly increasing reactor temperature.

The ultimate goal in LPCVD deposition of polycrystalline Si is to grow Si films with constant thickness and material properties. Because of the distributed nature of the LPCVD reactor, it is necessary to vary the wafer temperature along the length of the reactor to at least approach this goal. A continuously stirred tank reactor would give uniform films, but there are particulate and wafer loading problems associated with the design and operation of a CSTR reactor. On the other hand, these problems are avoided in the tubular LPCVD reactor. By recycling a fraction of the reactor effluent to the reactor inlet one can combine the benefits of the CSTR and the conventional LPCVD reactor. This modification provides highly uniform films as shown in (32) along with additional comparison of model predictions and experimental data.

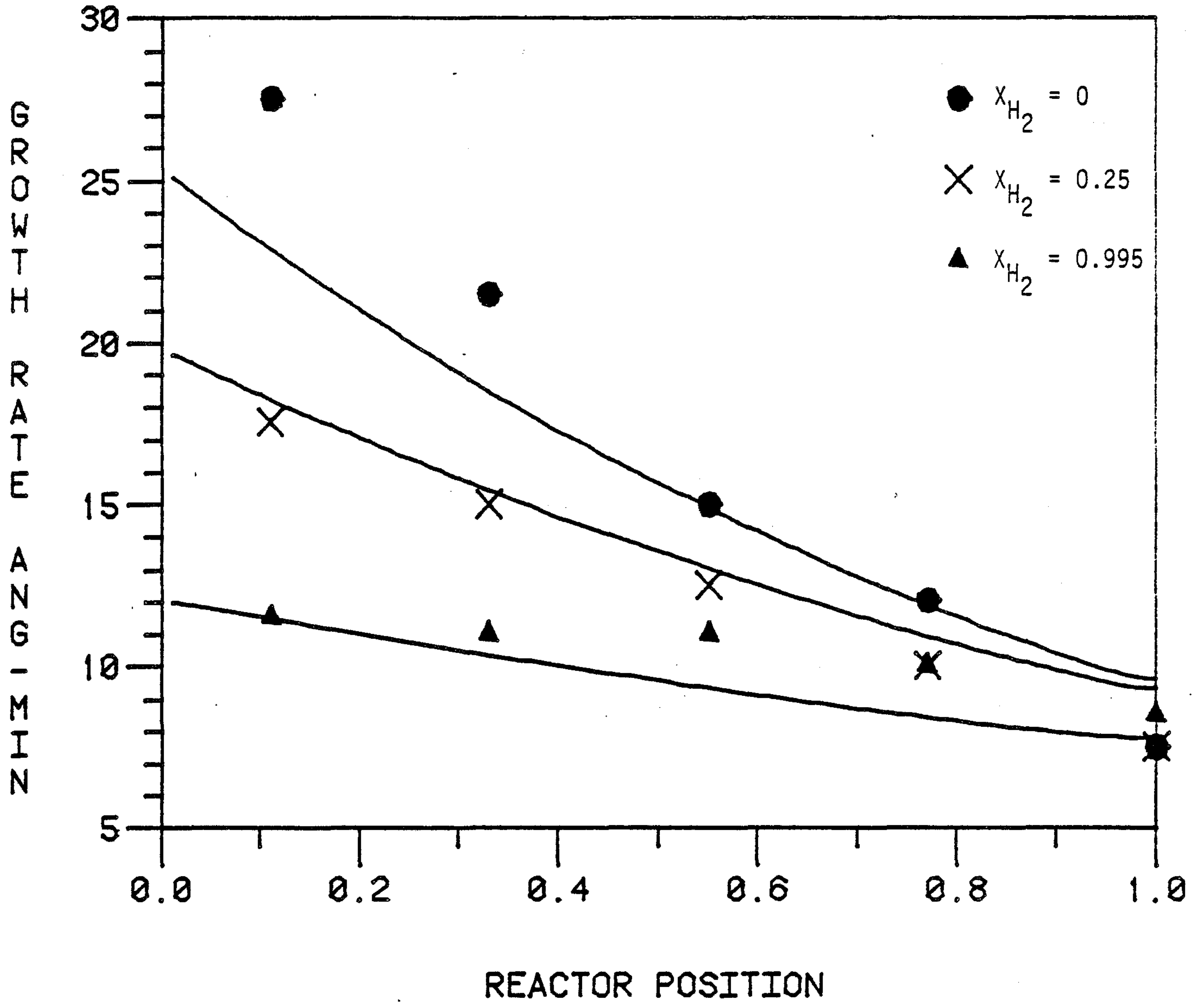


Figure 3. Model Predictions (solid line) vs. Experimental Data of Growth Rate Profiles as Function of Inlet H_2 Concentration. x_{SiH_4} = 0.0047 (32).

where the center boundary condition is the usual symmetry condition. The factor $1/(1+\varepsilon\chi)$ comes about because of the increase in the number of moles in the deposition reaction. The quantity η, connecting the descriptions for the flow region (2) and the wafers (4) is defined as:

$$\eta = \frac{2}{R_w^2} \int_0^{R_w} r \cdot \frac{\mathcal{R}\left[\chi(r)\right]}{\mathcal{R}(\chi_b)} \, dr \tag{6}$$

It plays the same role as the effectiveness factor in heterogeneous catalysis and is a measure of the film thickness uniformity. It represents the ratio of the total reaction rate on each pair of wafers to that we would obtain if the concentration in the cell formed by the two wafers were equal to the bulk concentration everywhere. Thus, if the surface reaction is the rate controlling step, $\eta = 1$, whereas if the diffusion between the wafers controls, $\eta < 1$. In the limit of strong diffusion resistance the deposition is confined to a narrow outer band of the wafers and a strongly nonuniform film results.

As in the analysis of catalytic reactions we make the modelling equations dimensionless and establish the characteristic dimensionless parameter combinations associated with the LPCVD process. By defining:

$$Da_1 = \frac{2L\ R_t\ (1+\alpha)}{v_o(R_t^2 - R_w^2)c_{10}} \mathcal{R}(\chi{=}0) \quad , \quad Da_2 = \frac{2L\ R_w^2/\Delta}{v_o(R_t^2 - R_w^2)c_{10}} \mathcal{R}(\chi{=}0)$$

$$g(\chi) = \frac{\mathcal{R}(\chi)}{\mathcal{R}(\chi{=}0)} \quad , \quad Pe = \frac{v_o\ L}{D_o} \tag{7}$$

$$\zeta = \frac{z}{L} \quad , \quad \xi = \frac{r}{R_w} \quad , \quad \Phi^2 = \frac{2R_w^2 \mathcal{R}(\chi{=}0)}{\Delta c_{10} D_{1m0} (T/T_o)^{0.65}}$$

the reactor equation (2) then takes the form:

$$\frac{d}{d\zeta}\left(\psi \frac{d\chi}{d\zeta}\right) + Pe \frac{d\chi}{d\zeta} - Pe(Da_1 + \eta\ Da_2)g(\chi) = 0 \tag{8}$$

with boundary conditions:

$$\left.\frac{d\chi}{d\zeta}\right|_0 = Pe\big(1 + \chi(0)\varepsilon\big)\chi(0) \qquad \text{and} \qquad \left.\frac{d\chi}{d\zeta}\right|_1 = 0 \tag{9}$$

and

$$\eta = 2 \int_0^1 \xi \frac{g\left[\chi(\xi)\right]}{g(\chi_b)} \, d\xi \tag{10}$$

The reaction-diffusion problem (4) governing the wafers becomes:

$$\frac{1}{\xi}\frac{d}{d\xi}\left(\frac{\xi}{1+\varepsilon\chi}\frac{d\chi}{d\xi}\right) + \Phi^2 g\big(\chi(\xi)\big) = 0 \qquad (11)$$

with boundary conditions

$$\chi(\xi=1) = \chi_b(\zeta) \quad , \quad \left.\frac{d\chi}{d\xi}\right|_0 = 0 \qquad (12)$$

In these equations the Damköhler numbers Da_1 and Da_2 represent the ratios of reactor space time to the characteristic time for deposition on the reactor wall and on the wafers, respectively. Pe is the axial Peclet number which represents the ratio of the time constant for convective transport to that for diffusive transport. The parameter, Φ, is equivalent to the Thiele modulus used extensively in analysis of heterogeneous reactions. It denotes the ratio of the characteristic time for diffusion in between the wafers to the characteristic time for deposition of Si on the wafer surfaces. Thus, if Φ is large the deposition is hindered by diffusion and a nonuniform film results. This effect is completely analogous to that of a large Thiele modulus for a porous catalyst. In fact, the modelling equations (8-12) are completely equivalent to those for a heterogeneous one-dimensional dispersion model for a fixed bed reactor with intraparticle resistance in the catalyst phase (cf. (34)). Thus, the extensive literature on the behavior of fixed bed reactors can readily be applied to the LPCVD reactor. For example, for large Pe the reactor will behave as a plug flow reactor and large differences in film thickness between the front and the back of the reactor may be expected. While for small Pe the reactor resembles a continuous stirred tank reactor (CSTR) and the film thickness should then be nearly uniform through the reactor.

The similarity between the wafer problem (11-12) and the cylindrical catalyst particle problem has already been mentioned. For an isothermal first order reaction with negligible volume change it is possible to solve (11-12) analytically and evaluate η (35, p. 114). Hence, one can determine a priori the necessary wafer spacing to obtain uniform film growth across each wafer, i.e. $\eta \doteq 1$. In commercial reactors the spacing is typically 3/16" (4.7 mm) which corresponds to the spacings in the cassettes used to store the wafers between process steps. This spacing exceeds the one necessary to ensure uniform growth in polycrystalline Si deposition. At the low pressures encountered in LPCVD, Knudsen diffusion becomes important with wafer spacings < 2 mm. For faster surface reactions than those involved in Si deposition, the diffusion of reactants between wafers could be rate controlling with standard 3/16" spacings. This appears to be the case in some SiO_2 and Si_3N_4 deposition schemes (36).

The modelling equations (8-12) form two nonlinear boundary value problems which we solve by orthogonal collocation. This

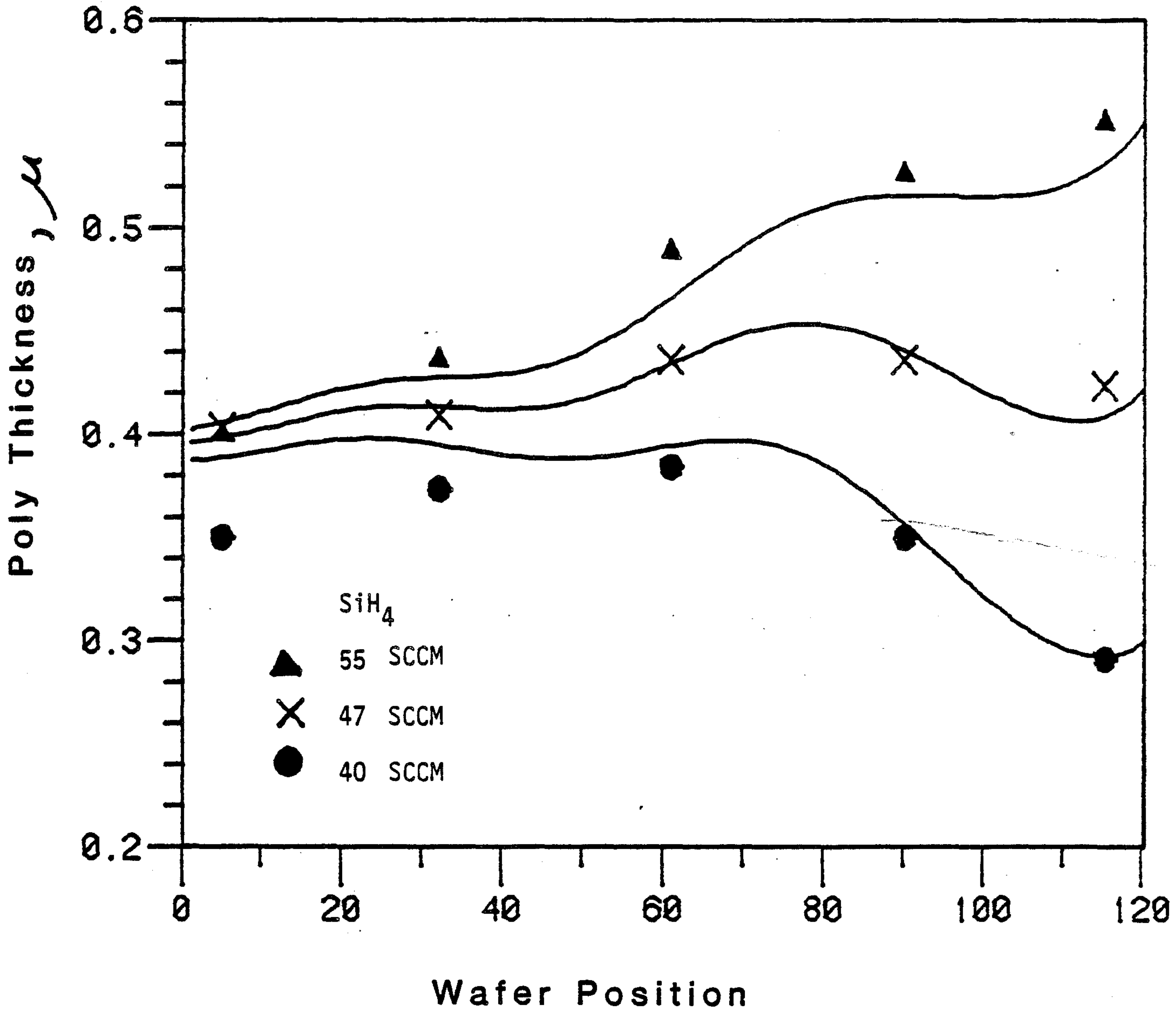


Figure 4. Model Predictions (solid line) vs. Experimental Data of Film Thickness Profiles as a Function of (pure) SiH_4 Flowrate (32).

Conclusions

Chemical vapor deposition is a key process for thin film formation in the development and manufacture of microelectronic devices. It shares many kinetic and transport phenomena with heterogeneous catalysis, but CVD reactor design has not yet reached the level of sophistication used in analyzing heterogeneous catalytic reactors. With the exception of the tubular LPCVD reactor, conventional CVD reactors may be viewed as variations on the original horizontal reactor. These reactors have complex flow fields and it is consequently difficult to control and predict the effect of operating conditions on the film thickness and composition.

The LPCVD reactor example illustrates that chemical reaction engineering concepts can readily be applied to CVD processes. Moreover, the similarity between the LPCVD reactor model and fixed bed reactor models means that it is possible to predict qualitatively the effect of operating conditions on the film growth without solving the modelling equations. The good quantitative agreement of model predictions with experimental observations further supports the modelling approach. The model also provides the ability to estimate kinetic parameters and to predict process conditions where variations in film thickness and composition are minimized.

The treatment in this paper has focused on Si derived materials since these are the most widely used materials in the microelectronics industry. However, the modelling approach can be also applied to the new III-V semiconductors. In fact, because of high demands on film thickness and composition uniformity in these systems this promises to be an area where chemical reaction engineering could play a major role.

Acknowledgments

This work was supported by a seed grant from the Microelectronic and Information Sciences Center at the University of Minnesota.

Legend of Symbols

c_o	total inlet concentration
D	dispersion coefficient $\doteq D_{1m}$
D_{im}	diffusivity of component i in the gas mixture
Da_1, Da_2	Damköhler numbers see eqn. (7)
$g(\chi)$	dimensionless rate see eqn. (7)
k, K_1, K_2	rate constants see eqn. (1)
L	reactor length
Pe	Peclet number see eqn. (7)
r	radial coordinate
R	gas constant
$\mathcal{R}$	reaction rate see eqn. (1)

R_w	radius of wafer
R_t	radius of reactor tube
T	temperature
v	linear velocity in annulus
x_i	mole fraction component i
z	axial coordinate
α	area of wafer carrier relative to reactor tube area
Δ	wafer spacing
ε	volume expansion coefficient
ζ	dimensionless axial coordinate see eqn. (7)
η	effectiveness factor see eqn. (6)
ξ	dimensionless radial coordinate see eqn. (7)
Φ	Thiele modulus see eqn. (7)
χ	conversion
ψ	see eqn. (2)

Subscripts

b	bulk conditions
1	SiH_4

Literature Cited

1. Walker, K. L.; Harvey, J. W.; Geyling, F. T.; Nagel, R. S. J. Am. Cer. Soc. 1980, 63, 69.
2. Hawkins, D. T., "Chemical Vapor Deposition 1960-1980, A Biography"; IFI/Plenum: New York, 1981.
3. Shepherd, W. H. J. Electrochem. Soc. 1965, 112, 988.
4. Rundle, P. C. Int. J. Electron. 1968, 24, 405.
5. Rundle, P. C. J. Crystal Growth 1971, 11, 6.
6. Bradshaw, S. E. Int. J. Electron 1967, 23, 381.
7. Eversteyn, F. C.; Severin, P. J. W.; van den Brekel, C. H. J.; Peek, H. L. J. Electrochem. Soc. 1970, 117, 925.
8. Eversteyn, F. C.; Peek, H. L. Philips Res. Rep. 1970, 25, 472.
9. Juza, J.; Cermak, J. J. Electrochem. Soc. 1982, 129, 1627.
10. Takahashi, R.; Koga, Y.; Sugawara, K. J. Electrochem. Soc. 1972, 119, 1406.
11. Dittman, F. W. Adv. Chem. Ser. 1974, 133, 463.
12. Fujii, E., Nakamura, H., Haruna, K., Koga, Y. J. Electrochem. Soc. 1972, 119, 1106.
13. Manke, C. W.; Donaghey, L. F. J. Electrochem. Soc. 1977, 124, 561.
14. Juza, J.; Cermak, J. Chem. Eng. Sci. 1980, 35, 429.
15. Ban, V. J. Electrochem. Soc. 1978, 125, 317.
16. Sedgwick, T. O.; Smith, J. E.; Ghez, R.; Cowher, M. E. J. Crystal Growth 1975, 31, 264.
17. Ban, V. S.; Gilbert, S. L. J. Crystal Growth 1975, 31, 284.
18. Orlander, D. R. Ind. Eng. Chem. Fundam. 1967, 6, 188.
19. Sugawara, K. J. Electrochem. Soc. 1972, 119, 1749.
20. Pollard, R.; Newman, J. J. Electrochem. Soc. 1980, 127, 744.

21. Hitchman, M. L.; Curtis, B. J. J. Crystal Growth 1982, 60, 43.
22. Hitchman, M. L.; Curtis, B. J.; Brunner, H. R.; Eichenberger, V. in "Physicochemical Hydrodynamics"; Spalding, D. B., Ed.; Advance Publications: London, 1977; Vol. 2.
23. Donaghey, L. F. in "Crystal Growth"; Pamplin, B. R., Ed.; Pergamon Press, 1980; p. 65.
24. Wahl, G. Thin Solid Films 1977, 40, 13.
25. Wahl, G. Rev. Int. Hautes. Temper. Refract., Fr. 1980, 17, 7.
26. Graves, D. B.; Jensen, K. F. Proc. 4th Euro. Conf. CVD, Eindhoven, The Netherlands, 1983.
27. Graves, D. B.; Jensen, K. F. Chem. Eng. Sci., submitted.
28. Gieske, R. J.; Mcmullen, J. J.; Donaghey, L. F Proc. 6th Int. Conf. CVD, Electrochem. Soc. Princeton, 1977, p. 183.
29. Hitchman, M. L.; Kane, J.; Widmer, A. E. Thin Solid Films 1979, 59, 231.
30. Kuiper, A. E. T.; van den Brekel, C. J. H.; de Groot, J.; Veltkamp, G. W. J. Electrochem. Soc. 1982, 129, 2288.
31. Claassen, W. A. P., Bloem, J.; Valkenburg, W. G. J. N.; van den Brekel, C. H. J. J. Crystal Growth 1982, 57, 259.
32. Jensen, K. F.; Graves, D. B. J. Electrochem. Soc. 1983, 130, 1950.
33. Levenspiel, O. "Chemical Reaction Engineering"; Wiley: New York, 1972; 2nd Ed.
34. Hlavacek, V.; Votruba, V. "Chemical Reactor Theory, A Review"; Amundson, N. R.; Lapidus, L., Eds.; Prentice Hall: Englewood Cliffs, NJ, 1977; p. 314.
35. Aris, R. "The Mathematical Theory of Diffusion and Reaction in Permeable Catalysts"; Oxford, 1975.
36. Rosler, R. S. Solid State Tech. 1977, 20 (4), 63.
37. Jensen, K. F.; Ray, W. H. Chem. Eng. Sci. 1982, 37, 199.

Figure 1. Typical CVD reactors.

Figure 2. LPCVD reactor system.

Figure 3. Model predictions (solid line) vs. experimental data of growth rate profiles as function of inlet H_2 concentration. $x_{SiH_4} = 0.0047$.

Figure 4. Model predictions (solid line) vs. experimental data of film thickness profiles as a function of (pure) SiH_4 flowrate.

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

TUNNELING ANOMALIES AND THE COEXISTENCE OF FERROMAGNETISM AND SUPERCONDUCTIVITY IN $ErRh_4B_4$ FILMS

Microelectronic and Information Sciences Center

Technical Report #05

A.M. Goldman
School of Physics and Astronomy
University of Minnesota

A.M. Kadin
School of Physics and Astronomy
University of Minnesota

L-J. Lin
School of Physics and Astronomy
University of Minnesota

C.P. Umbach
Department of Chemical Engineering
and Materials Science
University of Minnesota

TUNNELING ANOMALIES AND THE COEXISTENCE OF FERROMAGNETISM AND SUPERCONDUCTIVITY IN $ErRh_4B_4$ FILMS

MEIS Technical Report #05

A.M. Goldman
A.M. Kadin
L-J. Lin
C.P. Umbach

The research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

ABSTRACT

Single-particle tunneling measurements have been used to study the paramagnetic, superconducting and ferromagnetic phases of polycrystalline films of the reentrant superconductor $ErRh_4B_4$. Anomalous features of the tunneling conductance in the ferromagnetic phase which persist into the superconducting state have been observed. These features seem intimately connected with the onset of ferromagnetism in the region of coexistence of ferromagnetism and superconductivity just above T_{c2}.

$ErRh_4B_4$ (ERB) is a remarkable compound which at different temperatures is a normal paramagnet ($T > T_{c1} \approx 8$ K), a superconductor ($1\ K \approx T_{c2} < T < T_{c1}$) or a normal ferromagnet ($T < T_{c2}$).[1] Neutron scattering experiments in single crystal ERB have recently revealed that uniform ferromagnetic order and superconductivity coexist near, but above T_{c2}.[2] Furthermore, peaks in the low-angle scattering were interpreted as evidence for a spatial modulation of the alignment of the Er^{3+} moments with a wavelength of about 100 Å, which disappears when superconductivity vanishes at T_{c2}. This so-called spin periodic phase[3] has also been observed by the same technique in polycrystalline samples of ERB,[4] as well as in the other reentrant superconductor $HoMo_6S_8$.[5]

However, the interpretation and understanding of these results near T_{c2} have been strongly dependent on a set of theoretical assumptions, which have not yet been independently verified. It would therefore be highly useful to investigate in detail the same regime using alternative modalities. In this letter we present single-particle tunneling measurements on thin film ERB-Lu_xO_y-Sb junctions, over a temperature range (0.6 K - 15 K) that spans all three regimes. In addition to features that might be expected in a reentrant superconductor, the conductances G(V) of these junctions exhibit a series of very narrow anomalous features near T_{c2}, which have not previously been reported. There is as yet no theory to account for these sharp resonant peaks in the tunneling spectrum. However, the very existence of such well-defined characteristics in a number of junctions (5 junctions on two ERB films), strongly suggests that they are intimately connected with the nature of ERB and its reentrance from the superconducting to the ferromagnetic state. Towards the end of the paper we will speculate on some possible explanations of these features.

Single-particle tunneling provides a probe of the electronic density of states and of the excitations which couple to the electrons. It must be emphasized that the volume probed lies near the surface up to a depth of the coherence length (≈ 150 - 200 Å in ERB) when superconducting, and less when normal. The junctions reported here, with the exception of their (normal) Sb counterelectrodes, were identical to ERB-Lu_xO_y-In junctions used to study the Josephson effect.[6,7] Their ERB electrodes, which were almost single phase, were about 4000 Å thick and had T_{c1} = 8.1 K and T_{c2} = 1.0 K. The widths of these transitions were the order of 0.1 K. Below T_{c2} the normal electrical resistances of the ERB electrodes returned to 96% of their values above T_{c1}.

In Fig. 1 we plot the conductance G(V) = dI/dV for one junction (#JB2) with normal-state resistance R_N = 23 Ω, for T between 0.6 K and 15 K, showing the onset, growth, and collapse of superconductivity. The conductance is seen to be very nonlinear near zero bias even in the paramagnetic normal state above T_{c1}. The peak associated with the opening of the superconducting gap develops on top of this anomalous normal state of conductance. The estimated maximum value of the energy gap is Δ(1.6 K) ≈ 1.3 meV (after correction for thermal smearing[8]). This is in rough agreement with the vacuum tunneling of Poppe[9] on a single crystal of ERB, but much larger than the result of Rowell _et al._,[10] suggesting a damaged ERB surface or incompletely oxidized barrier in the latter case. The sub-gap conductance in Fig. 1 never goes to zero in the superconducting state, due to some combination of factors including barrier leakage, normal minority phase crystallites, and sub-gap states caused by magnetic pair-breaking. However, all junctions showed similar features at similar voltages, suggesting that the background conductance can be treated as a constant addendum. Below about 1.5 K, the sub-gap conductance begins to rise significantly, but the location of the gap peak does not appear to change much until it disappears rather abruptly at T = 1 K. This is suggestive of a first order phase transition at T_{c2}, as has been predicted theoretically.[11] These results are consistent with studies of the Josephson effect in ERB.[6] The systematic variation of G(V) with T and H are discussed elsewhere.[12]

A more detailed plot of G(V) is shown in Fig. 2 for T = 0.7 K, together with a higher derivative $d^2V/dI^2 \propto -dG/dV$ to bring out sharp (but small) features in the conductance. In addition to the broad peak at low voltage ($V \approx 0.7$ mV), note the sharper features out at $\approx \pm 3$ mV. This feature, corresponding to a resonant peak in the conductance, is actually far narrower than the plot suggests, probably narrower than the thermal smearing (≈ 50 μV). The much larger apparent width in d^2V/dI^2 is an artifact of the measuring process, limited by the ac modulation (≈ 500 μV) needed to give a large signal-to-noise ratio. As many as three additional similar features on either side may be visible, depending on the voltage and the temperature. At low temperatures, heating limits the accessible voltage range.

Another remarkable aspect of these narrow resonances was their extreme sensitivity to small magnetic fields. A field of 40 gauss suppressed the features significantly, and one of 100 gauss obliterated them entirely. This is surprising in that internal fields in ERB at 0.6 K should be greater than 1000 gauss.[13] Finally, these features not only continue into the ferromagnetic state to as low a temperature as we can measure, but they also persist into the superconducting phase for $T > T_{c2}$. In Fig. 3 we show the temperature dependence of these resonance peaks, and for comparison the T-dependence of G(0), the gap peak, and the broad ferromagnetic peak (in d^2V/dI^2) as well. The resonant peaks appear to be, at least approximately, integral multiples (2x, 3x, and 4x) of the first, and they extrapolate to zero at temperatures $\approx 1.4 - 1.5$ K. Perhaps significantly, this is close to the temperature at which G(0) exhibits a minimum, and also that at which the Josephson current in ERB-In junctions reaches a maximum.[6] The resonant peaks cannot be followed all the way up to that temperature, as they become smaller and eventually become lost in the background.

Before further discussion of these resonances, we briefly refer to some of the theoretical approaches to the problem of coexistence of ferromagnetism and superconductivity. In what is generally considered the standard theory,[3] uniform ferromagnetism cannot coexist spatially with superconductivity. However, the spin-periodic phase <u>can</u> coexist with superconductivity, in separate spatial domains from the normal ferromagnetic phase. In an alternative approach, Tachiki <u>et al</u>.[14] have emphasized that surfaces behave differently from the bulk, and in particular that superconductivity can coexist with uniform ferromagnetism within a magnetic penetration depth of the surface. Finally, based on earlier microscopic approaches,[15,16] Machida[17] has suggested that coexistence may entail the presence of a spatially modulated superconducting order parameter, commensurate with a spin density wave. Such a spatially inhomogeneous superconducting state (SISC) would be characterized by a highly anisotropic density of states, and might not even be superconducting in the usual sense.

The significance of the resonance peaks in the tunneling spectrum is uncertain within any of these theories. However, the extreme sensitivity to small magnetic fields would by itself appear to rule out phonons, electronic transitions, crystal field levels, charge density waves, and even magnons, as the cause of these features. A surface state can of course not be ruled out, and would seem to be relevant. It is difficult to see how magnetic domains could be responsible for these features, and uncertain how the spin-periodic phase would by itself couple to the electronic tunneling spectrum. On the other hand, some sort of anisotropic density of states showing sharp features relating to a SISC state would show up naturally in the tunneling spectrum of a randomly oriented microcrystalline sample. Furthermore, such a phase might persist into the "normal" ferromagnetic state. More theoretical guidance would clearly be helpful.

In summary, we have carried out single-particle tunneling studies in polycrystalline ERB films which reveal features in the paramagnetic, superconducting, and ferromagnetic phases of ERB. Anomalous sharp features which are thus far without explanation have been observed in the conductance of the ferromagnetic phase below T_{c2} and persist into the superconducting state. Their onset seem intimately associated with the onset of magnetic order and the destruction of superconductivity, and with the possibility of coexistence.

The authors would like to thank C.M. Varma, K. Machida, R. Klemm, G. Grest, and K. Levin for useful discussions. This work was supported by the ONR under contract N00014-78-C-0619, by the NSF under grants NSF/DMR-8011948 and 8015310 and by the Graduate School of the University of Minnesota.

References

1. W.A. Fertig, D.C. Johnston, L.E. DeLong, R.W. McCallum, M.B. Maple, and B.T. Matthias, Phys. Rev. Lett. 38, 987 (1977).

2. S.K. Sinha, G.W. Crabtree, D. G. Hinks, and H. Mook, Phys. Rev. Lett. 48, 950 (1982), and references therein.

3. H.S. Greenside, E.I. Blount, and C.M. Varma, Phys. Rev. Lett. 46, 49 (1981); E.I. Blount and C.M. Varma, Phys. Rev. Lett. 42, 1079 (1979).

4. D.E. Moncton, D.B. McWhan, P.H. Schmidt,G. Shirane, W. Thomlinson, M.B. Maple, H.B. MacKay, L.D. Woolf, Z. Fiske, and D.C. Johnston, Phys. Rev. Lett. 45, 2060 (1980).

5. J.W. Lynn, G. Shirane, W. Thomlinson, and R.N. Shelton, Phys. Rev. Lett. 46, 368 (1981); J.W. Lynn. R. Pynn, J. Joffrin, J.L. Ragazzoni, and R.M. Shelton, Phys. Rev. B 27, 581 (1983).

6. C.P. Umbach and A.M. Goldman, Phys. Rev. Lett. 48, 1433 (1982).

7. C.P. Umbach, A.M. Goldman, and L.E. Toth, Appl. Phys. Lett. 40, 81 (1982).

8. C.P. Umbach, L.E. Toth, A.M. Goldman, and E.D. Dahlberg, Physica (Utrecht) 108B, 803 (1981).

9. U. Poppe, Physica (Utrecht) 108B, 805 (1981).

10. J.M. Rowell, R.C. Dynes, and P.A. Schmidt, in Superconductivity in d- and f-band Metals, edited by H. Suhl and M.B. Maple (Academic Press, New York, 1980), p. 409.

11. Liam Coffey, K. Levin, and G.S. Grest, Phys. Rev. B 27, 2740 (1983).

12. C.P. Umbach, L.-J. Lin, and A.M. Goldman, in Superconductivity in d- and f-band Metals 1982, edited by W. Buckel and W. Weber (Kernforschungszentrum Karlsruhe GmbH, Karlsruhe, 1982) p. 209.

13. G. Cort, R.D. Taylor, and J.O. Willis, Physica (Utrecht) 108B, 809 (1981).

14. M. Tachiki, T. Koyama, and S. Takahashi, J. Mag. and Magn. Mat. 35, 43 (1983).

15. A.I. Larkin and Yu. N. Ovchinnikov, Zh. Eksp. Teor. Fiz. 47, 1136 (1964) [Sov. Phys. JETP 20, 762 (1965)].

16. P. Fulde and R.A. Ferrel, Phys. Rev. 135, A550 (1964).

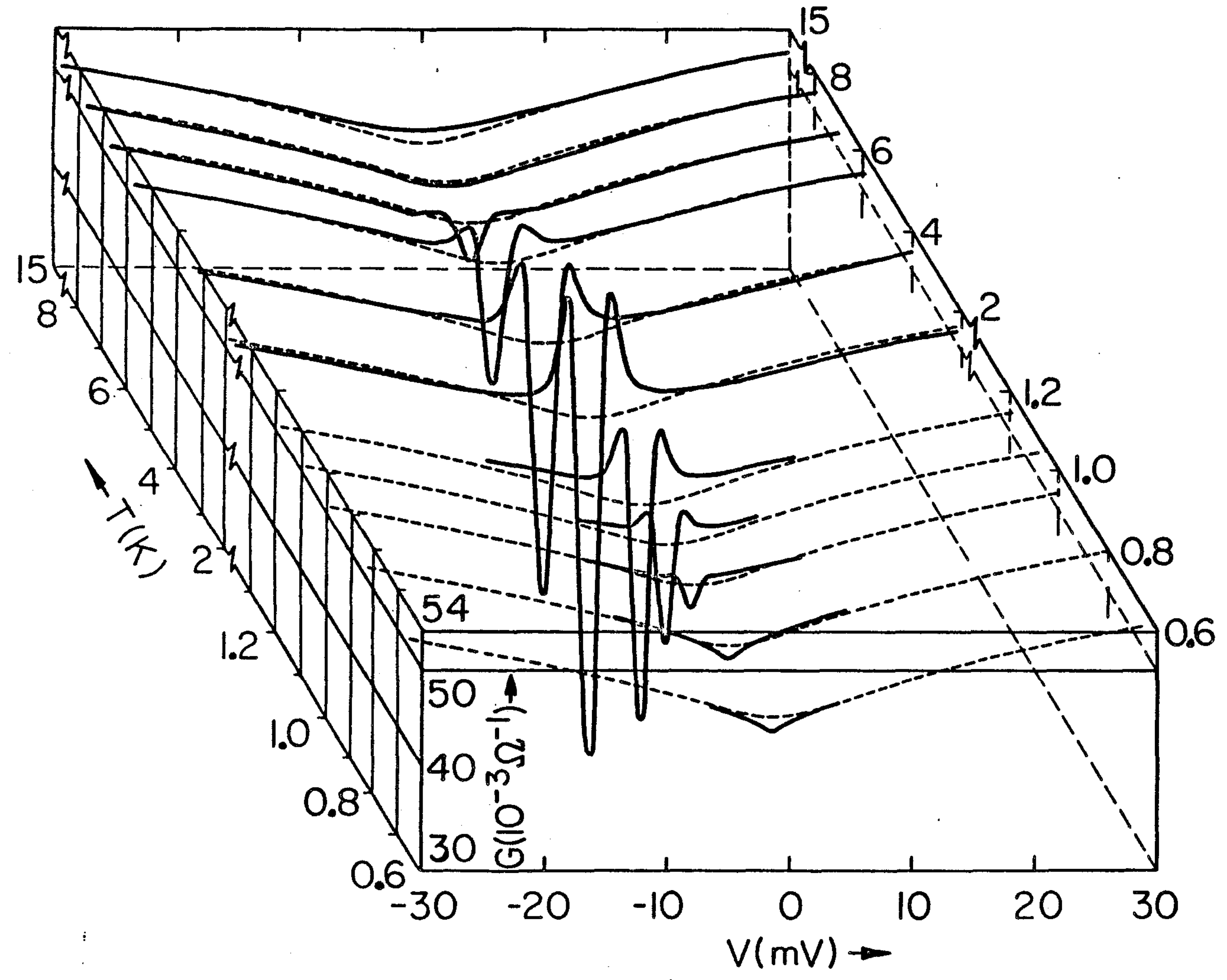
17. K. Machida, J. Phys. Soc. Japan 51, 3462 (1982).

Figure Captions

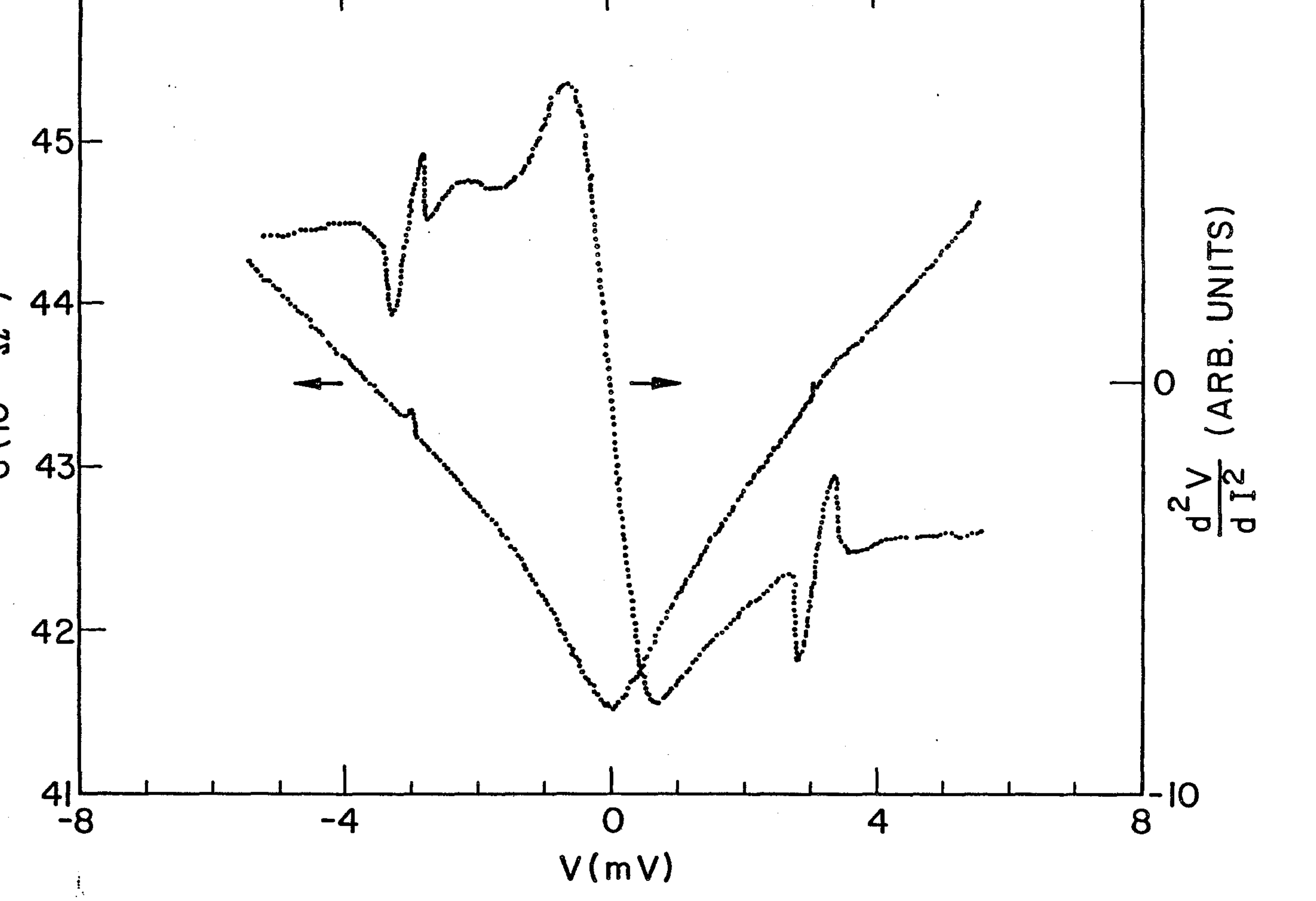
Fig. 1. Conductance G(V) vs. voltage for an $ErRh_4B_4$-Lu_xO_y-Sb tunnel junction (# JB2) at various temperatures. The dashed lines, presented for reference, are G(V) for T = 9 K. Data were measured using a constant current bridge and synchronous detection to obtain dV/dI, which was inverted numerically. For the lowest temperatures, the range of voltages was limited by heating.

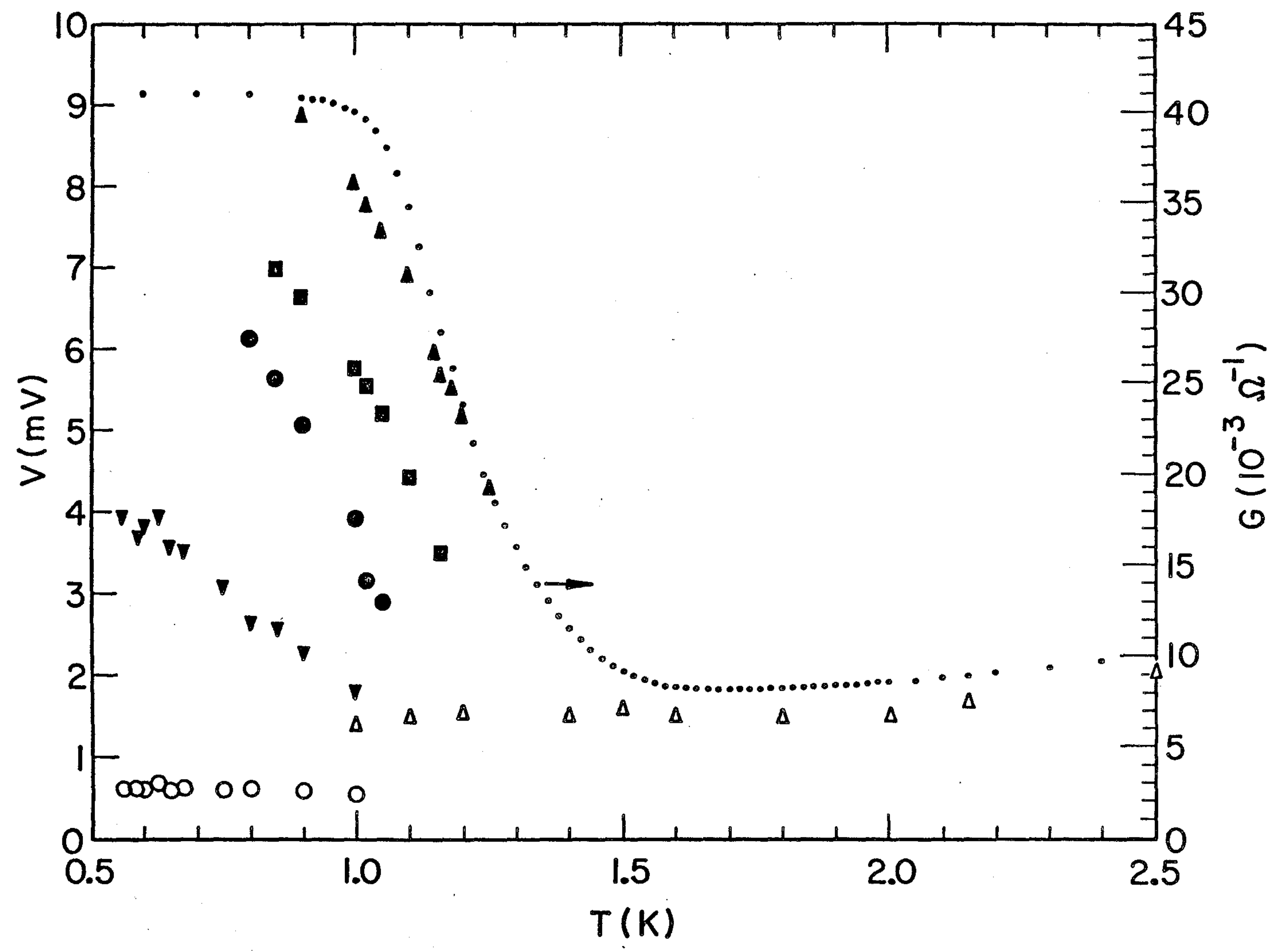
Fig. 2. Conductance G(V) and $d^2V/dI^2 \propto -dG/dV$ vs. V at T = 0.7 K for the junction JB2. The features of d^2V/dI^2 are broadened by the large modulation amplitude (460 μV as opposed to 10 μV for G).

Fig. 3. Summary plot of temperature dependences of various features in G(V) and d^2V/dI^2 for junction JB2. The zero-bias conductance G(0) is represented by small dots, the gap peak in G by Δ, the low-voltage peak in d^2V/dI^2 below T_{c2} by ○, and the four sharp resonances by large solid symbols. All features correspond to the left (Voltage) axis, except for G(0).



15
8
6
4
2
1.2
1.0
0.8
0.6
54
50
40
30
G(10^{-3}Ω^{-1})
T(K)
-30
-20
-10
0
10
20
30
V(mV)


G (10^{-3} Ω^{-1})
46
45
44
43
42
41
-8
-4
0
4
8
V(mV)
10
0
-10
$\frac{d^2V}{dI^2}$ (ARB. UNITS)


V(mV)
G (10^{-3} Ω^{-1})
T(K)
0.5
1.0
1.5
2.0
2.5
0
1
2
3
4
5
6
7
8
9
10
0
5
10
15
20
25
30
35
40
45

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

NATURE OF THE SMECTIC-A-SMECTIC-C TRANSITION NEAR A NEMATIC-SMECTIC-A - SMECTIC-C MULTICRITICAL POINT

Microelectronic and Information Sciences Center

Technical Report #06

C. C. Huang
School of Physics and Astronomy
University of Minnesota

S. C. Lien
School of Physics and Astronomy
University of Minnesota

NATURE OF THE SMECTIC-A-SMECTIC-C TRANSITION NEAR A NEMATIC-SMECTIC-A - SMECTIC-C MULTICRITICAL POINT

MEIS Technical Report #06

C. C. Huang
S. C. Lien

This research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

C1058[illegible]

ABSTRACT

A mean-field Landau model with a sixth-order term is employed to analyze the x-ray and heat-capacity data along the smectic-A - smectic-C (AC) transition line near one nematic-smectic-A - smectic-C (NAC) multi-critical point. Even though this model gives a satisfactory fitting to all data along the AC transition line, the coefficients obtained for this model suggest that the fluctuations near the NAC point may be important.

I. INTRODUCTION

Liquid crystals are organic molecules, anisotropic in shape, which have one or more mesophases between the crystalline state and the isotropic liquid. Unlike most materials which have a strong first order melting transition, the phase transitions between different mesophases, in many cases, can be continuous or weakly first order. Liquid crystals therefore provide a rich variety of orderings and are excellent systems for studying the three dimensional (d=3) aspects of phase transitions or multicritical points.

Among the various mesophases, three of them are relevant to our discussion here, namely, nematic (N), smectic-A (A) and smectic-C (C) phases. In the nematic phase, on average the long axis of each molecule (molecular director) is oriented along a preferred direction. The center of mass of each molecule is, however, free to diffuse throughout the system so that translational invariance is preserved. The smectic-A and the smectic-C phases are characterized by a one-dimensional density wave whose vector is along (A) or tilted with respect to (C) the molecular director. The molecules maintain translational invariance within the smectic planes. Between these three phases three phase transitions are possible, i.e., NA, AC and NC transitions. In 1973 de Gennes[1] proposed three theoretical models for these three transitions, respectively. Since then, the NA transition has been the most studied. So far the nature of the NA transition is still controversial. On the other hand, before the x-ray study by Safinya et al.[2] and our heat capacity study[3] it was generally believed that the continuous AC transition had helium-like exponents (n=2,d=3).[4] Our high-resolution heat capacity measurement demonstrated that the data could be uniquely described by a mean-field Landau model with an anomalously large sixth-order term.[3] Subsequently, x-ray, light scattering and heat capacity studies on the AC transition of another liquid crystal compound[5] confirmed our proposed model. Among these three transitions, the one least studied is the NC transition which has been found to be first order. In appropriate binary liquid-crystal mixtures it is possible to have lines of second order NA and AC transitions crossing over to a line of first order NC transitions. The triple point where these three transition lines intersect in the temperature-composition phase diagram is the NAC multicritical point.[6] Figure 1 shows the phase diagram for mixtures of octyl- and heptyloxy-p-pentylphenyl thiolbenzoate ($\bar{8}S5_{1-x}$ - $\bar{7}S5_x$).[7] Extensive experiments have been carried out on this mixture system in order to understand the nature of fluctuations near the NAC point.[8-11]

Recently there has been considerable interest in phase transitions associated with the Lifshitz point (LP),[12] i.e., a multicritical point connecting three distinct phases. The phases are, respectively, characterized by an order parameter, $\vec{M}(\vec{r})$, which is zero (disordered phase), finite but spatially uniform

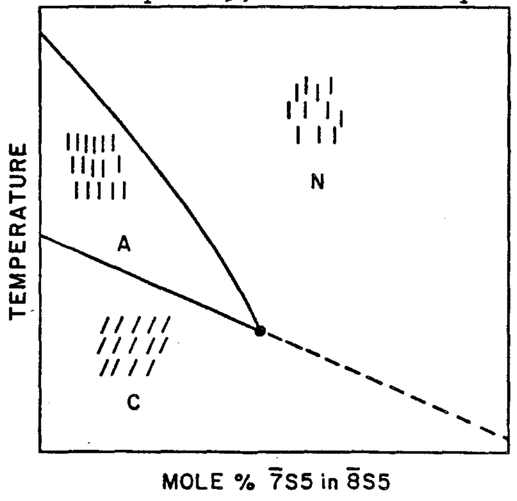


Fig. 1. Phase diagram near the NAC point. Two solid lines indicate continuous NA and AC transitions. The dotted line is the first-order NC transition line.

(uniformly ordered phase), and spatially varying (modulated phase). m denotes the number of spatial dimensions in which $\vec{M}$ can vary in the modulated phase. The realization of m=2 LP at a NAC point[13] has stimulated both experimental and theoretical interest. Theoretically it has been proven that d=3 is the lower critical dimensionality for m=2 LP.[14] Consequently, the NAC point is very different from m=1 LP which has been studied extensively on MnP.[15] Here we would like to report results obtained along the AC transition line near one NAC point.

II. DATA ANALYSIS AND DISCUSSION

Since there can be no long-range order on the uniformly ordered-modulated phase boundary in the n=m=d-1=2 LP problem, there can be no long-range (layered) positional order on the boundary between the A and C phases.[14] Here n is the dimensionality of the order parameter. However, in both pure[3,5] and mixture[16] systems, it has been found that the AC transition can be unambiguously described by the following mean-field Landau free energy with Ψ being the tilt-angle in the C phase:

$$F = F_0 + at\Psi^2 + b\Psi^4 + c\Psi^6. \tag{1}$$

Here F_0 is the nonsingular part of free energy, $t = (T-T_c)/T_c$ and T_c is the AC transition temperature. a, b and c are positive constants for a continuous transition. After minimizing the free energy with respect to Ψ, Ψ and anomalous part of the heat capacity ΔC are zero for $T > T_c$. In the case of $T < T_c$, one obtains

$$\Psi^2 = \frac{b}{3c}\left((1 + 3t'/t_0)^{1/2} - 1\right) \tag{2}$$

and

$$\Delta C = \frac{a^{3/2}\,T}{2(3c)^{1/2}\,T_c^{3/2}}(T_m - T)^{-1/2} \tag{3}$$

where $t'=-t$, $t_0=b^2/ac$ and $T_m=T_c(1+t_0/3)$. Actually, the full-width at half-height of the ($\Delta C/T$) vs. T curve is t_0 in the reduced temperature scale. Furthermore, from Eq. (2), if $t' \ll t_0$, then Ψ is proportional to $(t')^{1/2}$ which represents the critical behavior for a second order mean-field transition. But if $t' \gg t_0$, then Ψ is proportional to $(t')^{1/4}$ which is a characteristic of a tricritical point. Therefore, the locus of t_0's will separate the second order transition region from the tricritical region in the phase diagram.

Using high-resolution x-ray scattering, Safinya *et al.*[8,9]

have measured both tilt-angles and layer-spacings in the C phase just below the AC transition line of ($\bar{8}$S5 - $\bar{7}$S5) mixtures. Here we calculated the ratio (R)[15] of the measured tilt-angle to the quantity $((1+3t'/t_0)^{1/2} - 1)^{1/2}$ for each data point. By varying both T_c and t_0, the optimum values of T_c and t_0 are obtained by minimizing the relative deviation σ=D/M. Here D is the root-mean-squares deviation of the data from the calculated average ratio (M) for a given set of data. This was done with the experimental weighting factor for each data point. The same procedure was used to obtain T_c and t_0 for the 2-theta shift data (corresponding to layer spacings) which are proportional to Ψ^2. This kind of fitting scheme was found to be more satisfactory than a simple power law fitting in which, in general, one has to discard a good fraction of data with large tilt-angles.[16] The optimum values of T_c and t_0 for $\bar{8}$S5 and the four mixtures are listed in Table I along with other relevant physical parameters. The T_{AC}'s obtained by this method are consistent with the reported T_{AC}'s from a power law fitting.[9]

TABLE I. The parameters related to the NA and AC transition lines

Mole % of $\bar{7}$S5 in $\bar{8}$S5	(a)	T_{NA}(K)	T_{AC}(±5mK)$^{(b)}$	$T_{AC}^{(c)}$	$t_0 \times 10^3$(c)	T_{AC}/T_{NA}	$\sigma^{(d)}$
0	A	336.085	328.170	328.171	6.4	.9765	.044
	B			328.183	7.3		.105
15	A	331.990	326.620	326.618	4.5	.9838	.040
31	A	326.172	322.600	322.603	1.8	.9891	.049
	B			322.612	4.7		.066
41	A	324.163	323.258	323.259	0.62	.9972	.016
	B			323.271	1.6		.049
42	A	323.030	322.726	322.727	0.32	.9991	.020
	B			322.737	0.58		.069

a) type of measurements. A: tilt angle; B: 2-theta shift

b) these values obtained from the power law fitting (Ref. 9)

c) optimum values from our fitting

d) the minimum σ for the given set of data

The fact that heat-capacity data show abrupt jumps at the T_{AC}'s and no discernible fluctuations above the T_{AC}'s also suggests mean-field-like AC transitions. Moreover, from the full -widths at half-height of the heat-capacity anomalies, one obtains t_0's which are shown in Fig. 2 as open circles. A reanalysis of x-ray data also gives t_0 for each mixture as shown with solid dots. These results clearly demonstrate that as T_{AC}/T_{NA} increases toward 1, t_0 monotonically decreases toward zero and the tricritical region (the region with $t' >> t_0$) monotonically increases.

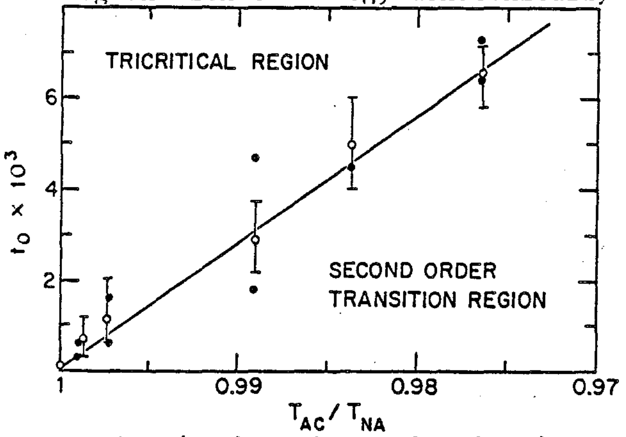


Fig. 2. t_0 vs (T_{AC}/T_{NA}). The solid line is a straight line through the data.

Figure 2 appears to suggest the existence of a mean-field tricritical point in the vicinity of the NAC point. A combination of x-ray[9] and heat-capacity[7] data allow one to gain further insight into the nature of the NAC point along the AC transition line. From $t_0 = b^2/ac$, $R^2 = b/3c$ and $\Delta C_J = a^2/2bT_c$ (the value of the heat-capacity jump at T_c), one can calculate the coefficients a, b and c associated with the Landau free energy. The relevant parameters and the results for a, b and c are shown in Table 2. Except for the 42% mixture, along the AC transition line the co-efficient a and c do not vary too much and b decreases by about a factor of 3 in the 2-theta shift measurement. In the 42% mixture a, b and c all show increases above the values for the 41% mixture, with the 41% mixture having the minimum value of b. Consequently, the large increase in a and c and not the decrease in b in the 42% mixture causes the further decrease in t_0 near the NAC point. This result cannot be easily explained by a Landau free energy with coupled order parameters.[16] The fluctuations[8-11] near the NAC point may be responsible for the large increase in a and the moderate increase in b and c. Surprisingly, the x-ray data for the 42% mixture are also well described by the mean-field expression (Eq. 2) (see σ's in Table 1). On the other hand, the monotonic decrease in b from pure $\overline{8}$S5 to 41% also suggest that along the AC line the system has the tendency to approach a mean-field tricritical point with b=0. Finally

TABLE 2. Coefficients of Landau free energy

Mole % of $\overline{7}$S5 in $\overline{8}$S5	ΔC_J(b)	(a)	R^2	$t_0 \times 10^3$	a(c)	b(c)	c(c)
0	5.61	A	.0461	6.4	1410	66	470
		B	.0363	7.3	2050	137	1260
15	5.97	B	.0289	4.5	1680	87	1010
31	6.61	A	.0106	1.8	2000	114	3580
		B	.0151	4.7	1410	56	1240
41	10.75	A	.0101	0.62	1180	24	800
		B	.0198	1.6	1560	42	710
42	36.5	A	.0073	0.32	2890	42	1950
		B	.0109	0.58	3470	62	1880

(a) type of measurements: A: tilt angle; B: 2-theta shift

(b) units in R_0 (the gas constant), Ref. 7

(c) units in Joul/mole

the large discrepancy in t_0 and c from tilt-angle and 2-theta shift measurements in the 31% mixture may be due to the presence of impurities because T_{AC} for this mixture is lower than that of both the 15 and 41% mixtures.[9] Simultaneous measurements on the tilt-angle and heat-capacity would be invaluable to explore the nature of AC transitions near the NAC point.

ACKNOWLEDGEMENTS

We are grateful to C. Dasgupta, Y. Shapira and J. Viner for useful discussions and C. R. Safinya and R. J. Birgeneau for providing us unpublished x-ray data. This work was supported in part by MEIS University of Minnesota, National Science Foundation-Solid State Chemistry- Grant DMR8204219 and the Department of Energy under contract DE-AC02-79-ER-10461 (SCL).

NOTE: In the near future the entire x-ray data near this NAC point will be published by authors in Ref. 10.

REFERENCES

1. P.G. de Gennes, Mol. Cryst. Liq. Cryst. 21, 49 (1973).
2. C.R. Safinya, M. Kaplan, J. Als-Nielsen, R.J. Birgeneau, D. Davidov, J.D. Litster, D.L. Johnson and M.E. Neubert, Phys. Rev. B21, 4149 (1980).
3. C.C. Huang and J.M. Viner, Phys. Rev. A25, 3385 (1982).
4. C.C. Huang and J.M. Viner, "Liquid Crystal and Ordered Fluids," Vol. 4, edited by A.C. Griffin and J.F. Johnson.
5. R.J. Birgeneau, C.W. Garland, A.R. Kortan, J.D. Litster, M. Meichle, B.M. Ocko, C. Rosenblatt, J.J. Yu and J. Goodby, Phys. Rev. A27, 1251 (1983).
6. D. Brisbin, D.L. Johnson, H. Fellner and M.E. Neubert, Phys. Rev. Lett. 50, 178 (1983) and references found therein.
7. R. DeHoff, R. Biggers, D. Brisbin and D.L. Johnson, Phys. Rev. A25, 472 (1982) and references found therein.
8. C.R. Safinya, R.J. Birgeneau, J.D. Litster and M.E. Neubert, Phys. Rev. Lett. 47, 668 (1981).
9. C.R. Safinya, MIT Thesis (1981) unpublished.
10. C.R. Safinya, L.J. Martinez-Miranda, M. Kaplan, J.D. Litster and R.J. Birgeneau, Phys. Rev. Lett. 50, 56 (1983).
11. S. Witanachchi, J. Huang and J.T. Ho, Phys. Rev. Lett. 50, 594 (1983).
12. R.M. Horneich, M. Luban and S. Shtrikman, Phys. Rev. Lett. 35, 1678 (1975).
13. J.H. Chen and T.C. Lubensky, Phys. Rev. A14, 1202 (1976).
14. G. Grinstein, Phys. Rev. B23, 4615 (1981) and references found therein.
15. Y. Shapira, invited talk in this conference.
16. C.C. Huang and S.C. Lien, Phys. Rev. Lett. 47, 1917 (1981).

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

PERFORMANCE BOUNDS ON MULTIPROCESSOR SCHEDULES

Microelectronic and Information Sciences Center

Technical Report #07

Richard Y. Kain
Electrical Engineering Department
University of Minnesota

Abolghasem A. Raie
Tehran, Iran

PERFORMANCE BOUNDS ON
MULTIPROCESSOR SCHEDULES

MEIS Technical Report #07

Richard Y. Kain
Abolghasem A. Raie

This research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

ABSTRACT

Optimizing general uniprocessor task scheduling is an NP-hard problem. For multiprocessor systems, list scheduling is not optimum but is easily implemented. Bounds on the total finishing time of list scheduling algorithms are known. In this paper we prove bounds on the performance of a multiprocessor list scheduling algorithm compared to the optimum schedule for the particular problem instance. We consider situations of which the system performance is measured by the weighted mean task finish time or measured by payoffs that exponentially decrease as a function of task completion times. For each measure, bounds are determined for two cases: 1) all processors can initiate processing at the same time, and 2) each processor has a given start time. All four problems utilize similar proofs. The proofs and results apply when task execution times are known quantities or when they are random variables with known probability distributions; the latter case is not discussed in this paper.

1. INTRODUCTION

The scheduling problem concerns a set of n tasks and a set of m homogeneous processors; that is, any task could be performed on any processor with the same execution time. There is no cost associated with "moving" a task from one processor to another one for execution at the latter site. Though general task scheduling may be constrained by precedence relations defining logical dependencies among tasks, in this paper we consider a set of tasks without precedence constraints.

The functions $p_i(f_i)$ define the amount of "payoff" that accrues by virtue of finishing task i at finish time f_i. The cumulative payoff of a schedule S is the summation of the payoffs of the tasks completed by the system:*

$$P(S) = \Sigma \; p_i(f_i) \qquad (1)$$

The objective of scheduling is to maximize this payoff. In this paper we consider exponential payoff functions of the form

$$p_i = a_i \operatorname{Exp}(-r_i f_i) \qquad (2)$$

* Most summations in the paper are over either index i, denoting tasks, or index j, denoting processors. To save space we have not written index specifications on such summations.

Here a_i and r_i are parameters, and we restrict the exponential payoff functions in any problem instance to have the same inverse "time constant" r_i.

Another interesting performance measure is the weighted mean finish time of the schedule. Task i is assigned weight w_i, and the scheduling objective is to minimize the weighted mean finishing time w given by:

$$w = \Sigma \ w_i f_i \tag{3}$$

In this paper we consider non-preemptive schedules only. A non-preemptive schedule defines the start time for each task in terms of the finish times of tasks scheduled beforehand (this phrasing covers tasks with random execution times). With non-preemption the scheduler never decides to interrupt the execution of one task to perform another task, even though this can be attractive if tasks may arrive for processing while others are being executed or if the payoff functions exhibit strange behavior(3). Preemption can be attractive with other performance measures, such as the length of the schedule(1).

A list scheduling algorithm places all tasks in a list before any are executed and then selects the task at the head of the list for execution after another task has been completed. List scheduling algorithms are optimum for scheduling uniprocessor execution of tasks with linear or exponential payoff functions(3); these results apply whether the execution times are known or are random variables with known arbitrary distributions. In the latter case the expected value of the payoff function is used.

Despite the fact that list scheduling is optimum for uniprocessor scheduling, this is not the case for multiprocessor systems, even with a centralized scheduler that has complete information regarding the tasks to be executed. This paper considers multiprocessor list scheduling for several classes of task descriptions and bounds the preformance of centralized list scheduling policies in relation to optimum scheduling, which generally requires comparisons among many possible schedules (making the optimization problem NP-hard).

2. RESULTS PAST AND PRESENT

Previous results concerning optimum uniprocessor list scheduling and bounds on multiprocessor list scheduling algorithms relate to the new multiprocessor list scheduling bounds presented here. We summarize these results in this section; other results concerning scheduling deterministic tasks were surveyed by Gonzalez(2).

If the tasks have known execution times T_i and linear payoff functions, the optimum uniprocessor schedule is a list schedule ordering the tasks by the values of the ratios T_i/w_i, with the task having the smallest value of the ratio placed first in the list. Since the weighted mean finish time performance measure is a special case of linear payoffs, the same list scheduling policy can be applied. Similarly, the optimum schedule for exponential payoff functions (with identical time constants) is a list schedule using increasing values of the ratios

$$r_i = \frac{\mathrm{Exp}(rT_i) - 1}{a_i} \tag{4}$$

These ordering ratios are special cases of ratios for tasks with random execution times, see Rothkopf(5).

While list scheduling is optimum for uniprocessor systems, it is well known that multiprocessor schedule optimization is NP-hard; the difficulty arises in determining the assignment of tasks to processors, since once this assignment has been made, each processor's schedule can be optimized by list scheduling.

List scheduling is effective in producing "good" multiprocessor schedules with low values of the largest finish time (the "schedule length") and for meeting deadlines. The performance of these policies in multiprocessors has been bounded; see

Coffman(1) for a summary.

We consider list schedules designed to reduce the weighted mean finish time or increase the cumulative exponential payoff; we study four cases:

A. wmft with identical processor start times;

B. wmft with arbitrary processor start times;

C. exponential payoff functions with identical start times;

D. exponential payoff functions with arbitrary start times.

Some of our bounds depend on the number of processors, but all cases have reasonable overall bounds, showing maximum percentage errors for list schedules of 20.7%, 30.9%, 14.7% and 26.9% for cases A, B, C, and D, respectively. The remainder of the paper discusses proofs of these bounds.

3. PROOF TECHNIQUES

The major results in this paper place bounds on the performance of the list schedule compared to the optimum schedule for the same instance, where the instance is defined by the sets of parameter values associated with each task. Letting p_{list} and p_{opt} denote the two payoffs, we consider the fractional difference given by

$$g = \frac{p_{list} - p_{opt}}{p_{opt}} \qquad (5)$$

All proofs are based on finding properties of the problem instance with the smallest number of tasks which maximizes g. This instance will be called IMG (instance maximizing g). Each proof follows the same five steps:

1. Find a global parameter constraint;
2. Find general properties of IMG;
3. Find properties of the optimal schedule for IMG;
4. Constrain some IMG parameters;
5. Vary remaining IMG parameters to maximize g.

Many of the properties that we will prove are the same for all cases; they are (numbering by the steps in which they will be shown):

1a. All tasks have the same list rank value in IMG;

2a. All tasks in IMG fall into one of two nonempty groups:

   i) TGZ: a set of k tasks whose execution times are greater than zero;

   ii) TAZ: a set of tasks whose individual execution execution times approach zero, but whose aggregate execution time is fixed;

3a. In the optimal schedule, each TGZ task is performed on a different processor from every other TGZ task;

3b. In the optimal schedule, the blocks of time allocated for TAZ tasks ends at the same time on all processors that are executing tasks at that time;

4a. The TGZ task execution times are all equal.

Generally these results are shown by contradictions or by varying the instance to show that g decreases if the condition is not met. In this paper we graphically illustrate the changes in instances or outline the contradiction but do not provide details, which can be found in Raie(4).

All proofs will be separated into "steps" as indicated above, and all result numbers include a letter indicating the first case to which the result applies.

4. PROOFS

We consider case A in some detail and then discuss only the modifications to adapt the proofs to the other cases.

4.1 CASE A: Weighted Mean Finish Time and Identical Start Times

4.1.1 Step 1:

There is one lemma in this step:

<u>Lemma</u> A.1: In IMG all task ranks are equal.

Proof: Straightforward by examining cases and showing a contradiction otherwise.

Since the task ranks are all equal, let $R=w_i/2T_i$. During the proof we consider modifying IMG task parameters. We will describe such changes as changes in execution times. To conform to Lemma A.1, each change in T_i is accompanied by a proportional change in w_i to preserve the value of R.

Let c_j denote the completion time of processor j, and let s denote the common processor start time. Then the wmft is given by

$$w = R[\Sigma\, T_i^2 + \Sigma\, c_j^2 - ms^2] \qquad (6)$$

To show Equation (6) consider first one processor and number the tasks so that the task numbered i occupies the ith position in the schedule. Then

$$f_i = s + \sum_{k=1}^{i} T_k$$

From the definition of R, we have $w_i = 2T_iR$. The wmft is given by

$$\omega = 2R \sum T_i(s + \sum_{k=1}^{i} T_k)$$

From this relation Equation (6) is easily verified for one processor. For many processors, we sum the wmft's for each processor.

By simple algebraic manipulations, we can also express the wmft in terms of the average processor completion time c_{av} and the deviations (from this average) of each processor, d_j:

$$w = R[\sum T_i^2 + mc_{av}^2 + \sum d_j^2 - ms^2] \qquad (7)$$

Given a problem instance, changing the schedule changes the deviations d_j, but not the other values in this expression. Therefore, w is minimized if one makes all deviations zero. In other words, all processors finish at the same time. Of course, the task execution times in a particular problem instance may not be consistent with a non-preemptive schedule in which all processors finish at the same time. Nevertheless, to construct worst case instances we can adjust task execution times as desired to satisfy any desired condition.

Let an additional subscript o indicate a parameter of the optimal schedule. We can compute g by substituting Eq. (7) in Eq. (5), which gives

$$g = \frac{\Sigma d_j^2 - \Sigma\, d_{jo}^2}{\Sigma\, T_i^2 + mc_{av}^2 + \Sigma\, d_{jo}^2 - ms^2} \qquad (8)$$

4.1.2. Step 2.

These three Lemmas concern the list schedule for IMG and are proved by computations based on Eq. (8). Let c_e denote the earliest processor completion time in the list schedule.

Lemma A.2: In IMG, the processors start at time zero (s = 0).

Proof: Since changing the common start time also changes all completion times, let $c_{av} = s+A$, where A is determined by the tasks and the schedule. The denominator of Eq. (8) depends upon s in the two terms

$$mc_{av}^2 - ms^2 = m(s+A)^2 - ms^2 = 2msA + mA^2$$

Clearly increasing s increases the denominator, so the maximum value of g occurs when s = 0.

Lemma A.3: In the list schedule, every task in IMG that completes after c_e belongs to the set TGZ and begins execution at time c_e.

Proof: 1) If a task begins after c_e, it violates the list scheduling rules, since it could have been started earlier at c_e on some other processor. Therefore if a processor's completion time exceeds c_e, it must have been executing a single task since time c_e, and therefore that task is a TGZ task.

2) If a TGZ task starts before c_e, consider breaking it into two tasks, one ending at c_e and the other starting at c_e. This change would increase g. Therefore every TGZ task in IMG begins at exactly time c_e.

Lemma A.4: In the list schedule for IMG, time c_e.

Proof: One can increase g by dividing any task that completes before c_e into two tasks with equal execution times. This change reduces ΣT_i^2 but leaves the rest of Eq. (8) unchanged, so it increases g. By continually subdividing the tasks, we approach a very large number of very short tasks, with the set occupying all processors until time c_e. These are the TAZ asks.

As a consequence of these lemmas, we know that each processor performs at most one TGZ task. Therefore, the list schedule for IMG looks like the structure depicted in Figure 1, where the processors have been numbered in the order of decreasing completion times. The shaded area in the figure represents the time allocated for performing TAZ tasks.

4.1.3 Step 3

These three lemmas concern the structure of the optimal schedule for IMG. Like the list schedule lemmas, they are proved by instance variations and computations based on Eq. (8).

Lemma A.5: In the optimal schedule for IMG, tasks in TGZ are assigned to different processors.

Proof: If this is not the case, then two TGZ tasks are assigned to the same processor in the optimal schedule. By lengthening the longer one and shortening the shorter one by the same amount x, we increase g, which contradicts the statement that this instance is the IMG. To perform the detailed computation, suppose that tasks i and j are the two TGZ tasks scheduled on the same processor in the optimal schedule. As a consequence of lemmas A.3 and A.4, these tasks are scheduled on different processors, say processors i and j, in the list schedule. Without loss of generality, let $T_i > T_j$. Now consider a new instance in which $T_i' = T_i + x$ and $T_j' = T_j - x$, and with all other parameters unchanged. The change affects d_i and d_j, in addition to T_i and T_j; in particular $d_i' = d_i + x$ and $d_j' = d_j - x$. Let A and B denote the numerator and denominator, respectively, of Eq. (8) for the first instance. Then $A' = A + 2x(d_i - d_j) + 2x^2$ and $B' = B + 2x(T_i - T_j) + 2x^2$. Since $g<1$, $A<B$. Also, due to Lemma A.3, $d_i - d_j = T_i - T_j > 0$. So identical positive increments $[x(T_i - T_j) + 2x^2]$ are being added to A and B, and $g < g' < 1$. Therefore the modified instance has a larger error, and so tasks i and j must have been scheduled on different processors in the IMG optimal schedule.

Lemma A.6: In the optimal schedule for IMG, processors not performing tasks from TGZ are performing tasks from TAZ. All such processors finish at the same time and that finish time is the earliest processor finish time in the optimal schedule.

Proof: Any processor processing TAZ tasks must finish at c_{eo}, for otherwise some of its TAZ tasks could be added to the end of the schedule for the earlier finishing processor.

Lemma A.7: In the optimum schedule, each processor that handles a TGZ task does not handle any TAZ tasks.

Proof: By examining several cases; the details are in Raie(4).

These lemmas establish the structure of the optimal schedule for the IMG instance, which is diagrammed in Figure 2.

4.1.4 Step 4

Now we constrain the task execution times.

Lemma A.8: All IMG tasks in the set TGZ have the same execution time.

Proof: If not, construct a new problem instance by replacing two tasks having unequal execution times by a pair of tasks, each with an execution time equal to the average of the two original execution times. This change increases g, a contradiction to the assumption that the instance was IMG.

4.1.5 Step 5

The proof is completed by varying the completion times and the number of TGZ tasks to maximize g.

Theorem A: The list schedule's fractional error g for m processors in case A is less than

$$g_A(m) = \frac{\sqrt{k(2m-k)} - k}{2m} \qquad (9)$$

with $k = m(2-\sqrt{2})/2$. The limit is

$$g_A(m) = \frac{\sqrt{2}-1}{2} = 0.207 \qquad (10)$$

Proof: Use differentiation first to find the value of the ratio of the completion time of the TGZ processors to the completion time of the TAZ processors, and then differentiate to find the value of k, the number of TGZ processes that maximizes g. The details are presented by Raie(4).

For a particular value of m, a tighter bound can be found by evaluating Eq. (9) at the integer values of k that are the floor and ceiling of the irrational value given by the Theorem.

4.2 CASE B: Weighted Mean Finish Time and Arbitrary Start Times

This case is similar to case A; the major difference arises in determining the set of processor start times for IMG. Without loss of generality, we number the processors in order of their start times. Most of the lemmas are identical to case A, with the inclusion of varying start times and the following exceptions:

Lemma B.9: The start times of the processors handling the k tasks in TGZ are all zero, and the start times of the remaining processors are all equal to c_e.

Remark: This implies that in the list schedule the m-k processors starting at c_e never process any tasks. The structures of the list schedule and the optimal schedule are shown in Figure 3. This leads to Theorem B:

Theorem B: The list schedule's fractional error g for m processors in case B is less than

$$g_B(m) = \frac{\sqrt{5m^2 - 6m + 2} - m}{2(2m-1)} \qquad (11)$$

This limit is an increasing function of m, with a limit

$$g_B(\infty) = \frac{\sqrt{5} - 1}{4} = 0.309 \qquad (12)$$

4.3 CASE C: Exponential Payoffs and Uniform Start Times

This proof is similar in structure to the case A proof; the mathematical details are different, as shown in the following discussion of the differences.

Lemma C.1: In IMG all task ranks are equal.

Proof: By computation and contradiction. The equal task ranks imply a simple formula for g(m):

$$g_C(m) = \frac{\Sigma\, \mathrm{Exp}(-rf_j) - \Sigma\, \mathrm{Exp}(-rf_{jo})}{\Sigma\, \mathrm{Exp}(-rs) - \Sigma\, \mathrm{Exp}(-rf_{jo})} \qquad (13)$$

The time scale for the IMG instance can be modified so that r=1 without affecting the properties of the instance. Therefore we consider only r=1 in the following arguments. First revise g(m) for r=1:

$$g_C(m) = \frac{\Sigma\, \mathrm{Exp}(-f_j) - \Sigma\, \mathrm{Exp}(-f_{jo})}{\Sigma\, \mathrm{Exp}(-s) - \Sigma\, \mathrm{Exp}(-f_{jo})} \qquad (14)$$

When the conditions of the Lemmas analogous to A.2-A.8 are met, we find that the maximum g(m) is obtained if the TGZ task execution times approach infinity. Then $g_C(m)$ can be written in the form

$$g_C(m) = \frac{y - y^\alpha}{\alpha - y^\alpha} \qquad (15)$$

where

$$\alpha = \frac{m}{m - k}, \quad y = \mathrm{Exp}(-c_e) \qquad (16)$$

Theorem C: The list schedule's fractional error g_C for m processors in case C is less than the value of g(m) given by Eqs. (15) and (16) with $c_e = k/m$

and k determined from $\alpha \approx 2.1231$, the solution of Eq. (17) with $\alpha > 1$.

$$\alpha - \alpha^2 \operatorname{Exp}\left(-\frac{(1-\alpha)^2}{\alpha}\right) - (1-\alpha)\operatorname{Exp}(1-\alpha) = 0 \quad (17)$$

The limit is $g_C(m) = 0.1468$.

Proof: By differentiating Eq. (15) and finding solutions in the feasible range.

Since Theorem C specifies a transcendental value for k, the number of TGZ tasks, a tight bound for given m can be found by comparing the values given by Eqs. (15) and (16) with k either the floor or the ceiling of the transcendental value.

4.4 CASE D: Exponential Payoffs and Arbitrary Start Times

This proof is structurally similar to the case B proof. The k processors handling TGZ tasks start at t=0 and the remaining m-k processors start at $t=c_e$. After determining that the lengths of all TGZ tasks should approach infinity, we can write $g_D(m)$ in the form

$$g_D(m) = \frac{y - y^\alpha}{\alpha - y^\alpha + y - 1} \quad (18)$$

with α and y defined in Eq. (16).

The overall result is given by

Theorem D: The list schedule's fractional error g_D for m processors in case D is less than

$$g_D(m) = \frac{1}{\left(\frac{m}{m-1}\right)^m + 1} \quad (19)$$

Proof: Differentiating Eq. (18) and setting it equal to zero to find the extreme points, we find

$$y = \alpha^{1/(1-\alpha)}$$

Then we discover that $g_D(m)$ is a decreasing function of α (which must be greater than one). The minimum α giving an integer k is

$\alpha = \frac{m}{m-1}$. Algebra then gives the result.

The error bound is an increasing function of m bounded above by $g_D(\infty) = 1/(1+e) = 0.269$.

5. SUMMARY

We have proved bounds on the performance of multiprocessor list scheduling policies for situations with the payoff being the weighted mean (task) finish time or with a cumulative payoff being constructed by summing exponentially decreasing payoffs associated with each task. These bounds become presuppose some conditions on the problem that may not apply in real situations. First, the task execution times are known. Second, no cost is associated with moving tasks among processors. Since neither condition may apply in an actual system, we have studied the consequences for relaxing the conditions. We have generalized the analysis presented here to situations in which the probabilty distributions of the task execution times are known. The same basic analysis techniques can be applied when the execution time distribution is known; the details are given by Raie[4].

If costs are associated with task movement, the problem is very difficult, so we leave it as an interesting open problem, which is to find bounds on performance of decentralized scheduling policies. Any realistic version of the problem is complex because the entire problem is trivial unless costs are associated with moving tasks between processors.

6. BIBLIOGRAPHY

1. Coffman, E. G., Jr., (ed.), Computer and Job-shop Scheduling Theory, Wiley, 1974.

2. Gonzalez, M. J., Jr., "Deterministic Processor Scheduling", Computing Surveys 9, 173-204, 1977.

3. Kain, R. Y. Gouda, M. G., and Raie, A. A., "DPRE Final Report, Vol. 2", Report F0606-FR-V2, Honeywell Systems and Research Center, Minneapolis, 1978.

4. Raie, A. A., "Toward Distributed Schedulers for Distributed Computer Systems", Ph.D. Thesis, Univ. of Minn., 1982.

5. Rothkopf, M. H., "Scheduling with Random Service Times", Management Science 12, 707-713, 1966.

7. BIOGRAPHIES

Richard Y. Kain was educated at MIT, where he also served as an Assistant Professor. In 1966 he joined the Electrical Engineering Department at the University of Minnesota, where he is a Professor. He teaches, performs research, and consults on computer system design and analysis.

Abolghasem A. Raie received his undergraduate education in Iran and his graduate education in Electrical Engineering at the University of Minnesota. Upon completion of his graduate studies he returned to Iran.

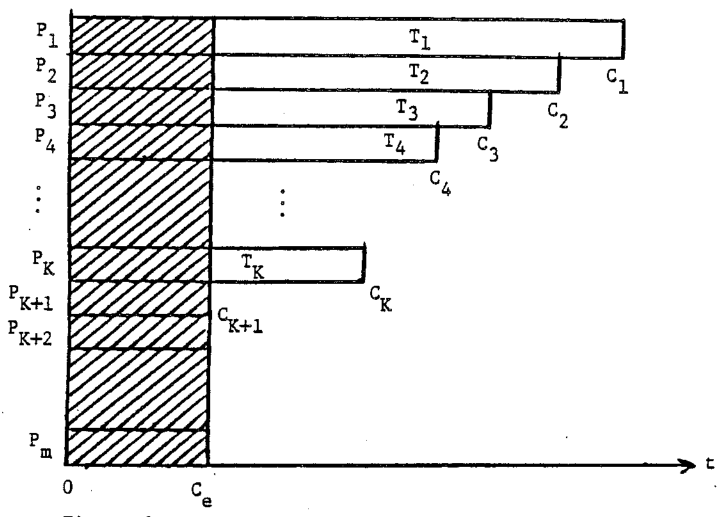


Figure 1: The List Schedule for the IMG Instance for Cases A and C.

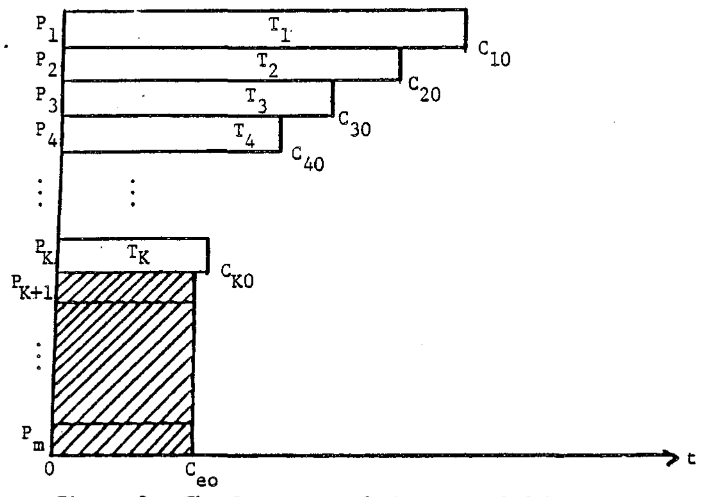
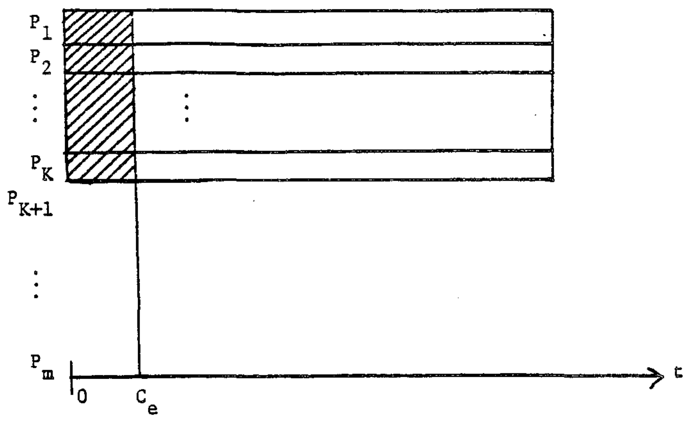
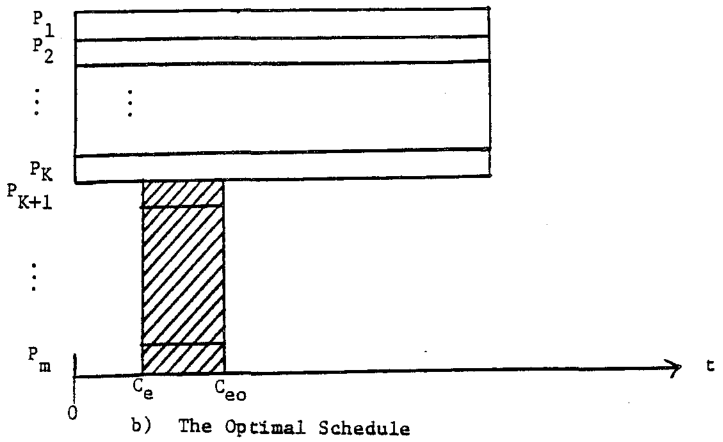


Figure 2: The Structure of the Optimal Schedule for IMG for Cases A and C.



a) The List Schedule



b) The Optimal Schedule

Figure 3: Schedules for Case B and Case D IMGs.

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

PARASITIC MESFET IN (Al,Ga)As/GaAs MODULATION DOPED FETs AND MODFET CHARACTERIZATION

Microelectronic and Information Sciences Center

Technical Report #08

Kwyro Lee
Department of Electrical Engineering
University of Minnesota

Michael Shur
Department of Electrical Engineering
University of Minnesota

Timothy J. Drummond
Department of Electrical Engineering and
Coordinated Science Laboratory
University of Illinois

Hadis Morkoc
Department of Electrical Engineering and
Coordinated Science Laboratory
University of Illinois

PARASITIC MESFET IN (Al,Ga)As/GaAs
MODULATION DOPED FETs
AND MODFET CHARACTERIZATION

MEIS Technical Report #08

Kwyro Lee
Michael Shur
Timothy J. Drummond
Hadis Morkoc

This research reported in this paper was supported in part by the Microelectronic & Information Sciences Center.

[1984]

ABSTRACT

A charge control model for n-channel Modulation Doped FET's (MODFETs) is extended to include the drain-to-source current through the doped (Al,Ga)As layer which becomes important for large positive gate voltages. This parasitic conduction leads to decreased device transconductances at high gate voltages. A unified and complete characterization technique for deducing the parameters of our model is introduced and used for the device characterization. Parameters, e.g., the saturation velocity, two-dimensional gas concentration at equilibrium, thickness of the doped (Al,Ga)As layer, etc., deduced using the model, are in good agreement with the independent calculations and measurements. However, the deduced values of the room temperature, low field mobility of the two dimensional electron gas are considerably smaller than those measured by Hall effect and in long gate MODFETs. This model is in good agreement with the characteristics of high current normally-on MODFET's. The maximum measured current swing of 300 mA/mm gate is reported.

I. INTRODUCTION

Recently, Modulation Doped FETs (MODFETs) - also called High Electron Mobility FETs (HEMTs) and Two-dimensional Electron Gas FETs (TEGFETs) - have emerged as one of the strongest contenders for superfast integrated circuits. Ring oscillator delays of 12.2 ps at 300 K were obtained [1] with delays as low as 5 ps at 77 K possible. We have achieved intrinsic device transconductances as high as 350 mS/mm gate at 300 K (and 565 mS/mm at 77 K) with currents as high as 300 mA/mm gate [2] and predicted much higher 300 K transconductances (up to 1000 mS/mm gate at 300 K) in devices with an optimized design [3]. Further progress in MODFETs cannot be possible unless realistic device models and a thorough understanding of device physics are achieved.

We have recently developed a simple charge control model [4] (a generalization of a charge control model proposed in Reference [5]), which allowed the calculation of the I-V and C-V characteristics [6] of (Al,Ga)As-GaAs MODFETs. In this paper the model is extended to include the parallel conduction in the (Al,Ga)As layer which takes place at gate biases for which this layer may become partially undepleted. Based on this model and on a new method of determining the source and drain series resistance [7], we propose a new characterization technique for MODFETs which makes possible an accurate determination of the device parameters from measured I-V characteristics.

II. EXTENDED CHARGE CONTROL MODEL

A schematic cross-sectional diagram of a MODFET is shown in Figure 1. The transistor metalization pattern for source, drain and gate are identical to that of a conventional split source MESFET with gate length L = 1 μm and gate width Z = 290 μm. The difference is that the primary conduction path is the first ~ 150 Å of GaAs away from the heterojunctions. The electrons in the channel have transferred from the large bandgap (Al,Ga)As to the smaller bandgap GaAs. In an ideal normally-on FET at zero bias, charge transfer across the heterojunction and the Schottky gate will deplete just that thickness of (Al,Ga)As between the gate and the GaAs layer. In the following analysis the effect of a layer of undepleted (Al,Ga)As resulting from a positive gate is considered. It will be shown that the presence of this "parasitic" MESFET can be taken advantage of to characterize normally on MODFETs.

A qualitative sketch of the interface carrier density in a MODFET as a function of the gate voltage is shown in Fig. 2. V_{t2} is the device threshold voltage. At gate voltages larger than V_{t2} but smaller than

$$V_{t3} = V_{t2} + (V_{po})_{2D} \tag{1}$$

where

$$(V_{po})_{2D} = \frac{qn_{so}(d + \Delta d)}{\varepsilon} \tag{2}$$

the charge control model proposed in the references applies [4,5]. Here V_{t3} is the threshold voltage for the parasitic MESFET formed by the gate and doped (Al,Ga)As layer, $(V_{po})_{2D}$ is the pinch-off voltage of the two dimensional electron gas, q is the electronic charge, n_{so} is the equilibrium two dimensional interface carrier concentration calculated in Reference [8],

$$d = d_d + d_i \tag{3}$$

where d_d is the thickness of doped (Al,Ga)As layer, d_i is the thickness of undoped (Al,Ga)As layer, and $\Delta d \simeq 80$ Å is the effective width of the two dimensional gas [4].

According to the charge control model [4] represented by a dashed line in Fig. 2:

$$qn_{s2} = C_{2D}(V_g - V_{t2}) \tag{4}$$

where $n_{s2}(V_g)$ is the interface carrier density of the two dimensional electron gas, V_g is the gate voltage,

$$C_{2D} = \varepsilon/(d_d + d_i + \Delta d). \tag{5}$$

is the capacitance between gate the the two dimensional electron gas given in Ref. [9]. Here ε is the dielectric permittivity of (Al,Ga)As. The value of d_d can be measured from C-V characteristics of a long gate MODFET or calculated as follows [4,5]:

$$d_d = [2\varepsilon(\phi_b - \Delta E_c/q - V_{t2})/(qN_d)]^{1/2} \tag{6}$$

where ϕ_b is the Schottky barrier height, ΔE_c is the conduction band discontinuity at the heterointerface and N_d is the doping density in (Al,Ga)As.

At gate voltages higher than V_{t3}, $n_{s2} \simeq n_{so}$ and the charge control model proposed in References [4,5] does not apply. The parallel conduction from the (Al,Ga)As MESFET has to be included. The charge qn_{s3} of the electrons in the MESFET channel is given by

$$qn_{s3} = C_{3D}(V_g - V_{t3}) \tag{7}$$

$$C_{3D} = \frac{\varepsilon}{t_d} \tag{8}$$

and

$$t_d = d_d - \frac{n_{so}}{N_d} \tag{9}$$

Eq. (7) is applicable when

$$V_g - V_{t3} << (V_{po})_{3D} \tag{10}$$

where

$$(V_{po})_{3D} = \frac{qN_d t_d^2}{2\varepsilon} \tag{11}$$

is the MESFET pinch-off voltage (see Fig. 2).

III. I-V CHARACTERISTICS AT LARGE GATE BIASES

In this Section the extended charge control model of Section II is applied to calculate the I-V characteristics of a MODFET for large gate biases. The qualitative distribution of electrons in the two dimensional electron gas and in the doped (Al,Ga)As layer is shown in Figure 3. In the gate voltage range

$$V_{t2} < V_g < V_{t3}, \qquad (12)$$

the two dimensional gas is partially depleted everywhere in the channel (see Fig. 3a). Previous models [4,6] can be applied only in this regime.

In the gate voltage range

$$V_{t3} \lesssim V_g \lesssim V_{t3} + V_{ds} \qquad (13)$$

where V_{ds} is the drain-to-source voltage, the electron distribution in the channel looks like the distribution shown in Fig. 3b. (We limit ourselves to $V_{ds} \lesssim V_{ds}^s$ where V_{ds}^s is the drain-to-source saturation voltage.) Below we calculate the drain-to-source saturation current for the gate voltage range given by Eq. (13) (where we assume $V_{ds} = V_{ds}^s$). In this case the current $(I_{ds})_{2D}$ carried by the two dimensional gas in Region I ($0 < x < L_1$) is given by

$$(I_{ds})_{2D} = qZ\mu_2 n_{so} \frac{V(L_1)}{L_1} = I_2 \cdot V(L_1)/V_1 \qquad (14)$$

where

$$I_2 = qn_{so}v_{s2}Z, \qquad (15)$$

$$V(L_1) = V_g - V_{t3} \qquad (16)$$

and

$$V_1 = v_{s2}L_1/\mu_2 \qquad (17)$$

Here v_{s2} and μ_2 are the electron saturation velocity and the low field mobility of the two dimensional gas. On the other hand, drain saturation current $(I^s_{ds})_{2D}$ in Region II ($L_1 < x < L$) can be found using our two piece model [4],

$$(I^s_{ds})_{2D} = I_2 \frac{[(V_{po})^2_{2D} + V_2^2]^{1/2} - V_2}{(V_{po})_{2D}} \tag{18}$$

where

$$V_2 = \frac{v_{s2}L_2}{\mu_2} . \tag{19}$$

and $(V_{po})_{2D}$ is the pinch-off voltage of the two dimensional electron gas.

The value of L_1 (or V_1) can be found by equating Eqs. (18) and (14) and by noting that $L = L_1 + L_2$, or that

$$V_1 + V_2 = V_0 \tag{20}$$

where

$$V_0 = \frac{v_{s2}L}{\mu_2} . \tag{21}$$

The resulting expression for V_1 is

$$V_1 = V(L_1) \cdot \frac{V_0 + [V_0^2 + 2(V_{po})_{2D}V(L_1) + (V_{po})^2_{2D}]^{1/2}}{(V_{po})_{2D} + 2V(L_1)} \tag{22}$$

Substitution of Eq. (22) into Eq. (14) yields

$$(I^s_{ds})_{2D} = I_2 \cdot \frac{(V_{po})_{2D} + 2V(L_1)}{V_0 + [V_0^2 + 2(V_{po})_{2D}V(L_1) + (V_{po})^2_{2D}]^{1/2}} \tag{23}$$

The saturation drain-to-source current I^s_{ds} and the device transconductance in the saturation regime g^s_m vs. the gate voltage are shown in Fig. 4a and 4b. The solid curves show the dependences predicted by the two-piece charge control model which does not take into account the finite value of n_{so} [4,5,6]. As pointed out above for a small voltage, e.g. $V_g < (V_{po})_{2D} + V_t$, these curves coincide with the dependences given by the present model. At higher biases the transconductance decreases, and finally, at $V_g = V_t + (V_{po})_{2D} + V_0$ the maximum value of the current, I_2, through the two dimensional gas is reached:

$$I_2 = qn_{so}v_sZ \tag{15}$$

This maximum current carried by the two dimensional gas was analyzed in Reference [3]. Any further small increase in current must only be due to the conduction through the doped (Al,Ga)As layer. This current $(I^s_{ds})_3$ may be found using the model developed in Reference [10] as:

$$(I^s_{ds})_{3D} = \frac{2\varepsilon v_{s3}Z(V_g - V_{t3})^2}{t_d\left[(V_{po})_{3D} + \frac{3v_{s3}L}{\mu_3}\right]} \tag{24}$$

where μ_3 and v_{s3} are the mobility and saturation velocity in (Al,Ga)As, t_d is given by Eq. (9).

Implications of finite n_{so} and, as a result, the finite gate voltage swing are illustrated by Fig. 5. The transfer characteristic shown in Fig. 5a does not show any transconductance degradation up to current levels as high as 40 mA. On the other hand, the characteristic shown in Fig. 5b shows transconductance degradation when the drain current exceeds 15 mA, indicating a very small value of $n_{so} \approx 4 \times 10^{11}$ cm^{-2}, obtained from the best fit to the experimental data. Both devices are normally-on and the gate current is negligible at all gate voltages. Therefore the decrease in the transconductance at large gate bias cannot be attributed to the gate leakage current.

IV. CHARACTERIZATION TECHNIQUE

To illustrate the characterization technique, high current normally-on MODFETs were selected and measured. Fabrication data for three transistors is given in Table I. I-V characteristics were measured for half of a device ($Z = 145\ \mu m$) under a probe station. The crystals were grown by molecular beam epitaxy. Details of the crystal growth and device fabrication have been reported elsewhere [11]. From the measured I-V characteristics we need to extract the series source and drain resistances R_S and R_D, V_{t2}, V_{t3}, μ_2, μ_3, d_d and V_{s2}. The value of V_{s3} is not important because the "MESFET" is governed by Shockley's model due to the very low mobility, μ_3. The following discussion demonstrates a method of determining all parameters necessary to the model with the exception of N_d and d_i which are known from the crystal growth.

The first objective is to find gate voltages <u>within</u> the range $V_{t2} < V_g < V_{t3}$ without attempting an accurate determination of V_{t2} and V_{t3}. We start by plotting the drain-to-source saturation current, I_{ds}^s and the saturation transconductance g_m^s as a function of V_g. Following the two piece charge control model [6] we find the drain-to-source saturation current at large values of V_g:

$$I_{ds}^s \simeq I_2 \frac{V_g - V_{t2} + V_0}{(V_{po})_{2D}} \tag{25}$$

where V_0 is given by Eq. (21). Hence $V_{t2} - V_0$ may be found from the linear asymptote of the saturation current at large gate biases (but below $(V_g)_{max}$), corresponding to the maximum transconductance. We take $V_{g1} = V_{t2} + V_0$ and

$$V_{g2} = V_t - V_0 + \tfrac{3}{4}((V_g)_{max} - V_{t2} - V_0) \tag{26}$$

as lower and upper bounds, respectively, for the gate voltage in the MODFET regime of operation.

Plots of I_{ds}^{s} vs. V_g and g_m^s vs. V_g measured in our laboratory are shown in Figs. 6 and 7 for devices 1 and 2. The points marked by Δ in Fig. 6a and 7a are shifted due to the use of two sweep ranges for the gate in our measurements: −1 V to 0 V for points marked by open circles and from 0 V to 0.8 V for points marked Δ. This offset voltage is due to the shift of the threshold voltage caused by traps [9,11] and is taken into account by a parallel transfer of the measured data in the gate voltage range $0 < V_g < 1$ V (points marked in Fig. 6a and Fig. 7a by solid circles). By drawing asymptotes to the curves at large gate voltages we find that $V_{g1} \simeq V_{t2} + V_0 \approx -1.4$ V, $V_{g2} \simeq 0$ V for device 1 and $V_{g1} \simeq -1.6$ V and $V_{g2} \simeq 0$ V for device 2.

In the voltage range $V_{g1} < V_g < V_{g2}$ the simple charge control model for the two-dimensional gas [4] applied. Accordingly, at very low drain-to-source voltages

$$R_{ds} = \frac{L}{\mu_2 C_{2d}(V_g - V_{t2})Z} + R_{S+D} \qquad (27)$$

where $R_{S+D} = R_S + R_D$. Hence, R_{S+D} and V_{t2} may be determined from the best least square linear fit to the measured R_{ds} vs. $\frac{1}{V_g - V_{t2}}$ in the gate voltage range $V_{g1} < V_g < V_{g2}$ (see Figs. 6b and 7b). In most cases the device structure is symmetrical; therefore, $R_S = R_D = R_{S+D}/2$. The value of R_S determined in this manner is in good agreement with those determined independently [7].

The thickness of doped (Al,Ga) can be derived from Eq. (6) and the value of V_{t2} derived above. The values $\phi_b = 1$V, $\Delta E_c/q = 0.34$ V and $\varepsilon = 10^{-10}$F/m were used in the calculation. For devices 1 and 2 an optimum value of $V_{t2} \simeq$

-1.9 V is deduced for which $d_d = 570$ Å, in agreement with that expected from the fabrication procedure. Knowing d_d, the values of C_{2D} and C_{3D} can be found from Eqs. (5) and (8). The undoped layer thickness, d_i, is known and Δd = 80 Å [4]. For device 1 $C_{2D} = 0.22$ pF and $C_{3D} = 0.35$ pF. For device 2 C_{2D} = 0.20 pF and $C_{3D} = 0.31$ pF (see Table II).

Once the values of $R_s = R_D$ and V_{t2} are measured and the values of d_d, C_{2D} and C_{3D} are deduced as described above, the threshold voltage of the (Al,Ga)As "MESFET" V_{t3}, the pinch-off voltage of the two dimensional electron gas, $(V_{po})_{2D}$, the low field electron mobility of the two dimensional electron gas, μ_2, and of the doped (Al,Ga)As layer, μ_3, and the equilibrium electron concentration, n_{so}, can be determined using the plot of the intrinsic drain-to-source conductance at a low drain-to-source voltage

$$g_{d_i} = \frac{1}{(1/g_{ds})_{measured} - R_{S+D}} \tag{28}$$

vs. the gate voltage, V_g.

A qualitative sketch of g_{d_i} vs. V_g is shown in Fig. 8. When $V_{t2} < V_g < V_{t3}$, we have drain conductance due only to two dimensional electron gas

$$g_{d_i} = \frac{\mu_2 C_{2D}(V_g - V_{t2})}{L} \tag{29}$$

which is shown with dashed lines in Fig. 8. The slope of this curve at large gate biases changes due to the transition from "MODFET" to (Al,Ga)As "MESFET" operation because $\mu_3 \ll \mu_2$. When $V_g > V_{t3}$

$$g_{d_i} = \frac{\mu_2 C_{2D}(V_{po})_{2D}}{L} + \frac{\mu_3 C_{3D}(V_g - V_{t3})}{L} \tag{30}$$

which applies for $V_g - V_{t3} \ll (V_{po})_{3D}$.

The values of μ_2 and μ_3 can be determined from the slopes shown by dashed line in Fig. 8 using Eq. (29) and Eq. (30), respectively. Furthermore, V_{t3} can be determined as shown in the same figure. The pinch-off voltage of the two dimensional gas $(V_{po})_{2D}$ is then found as

$$(V_{po})_{2D} = V_{t3} - V_{t2} \tag{31}$$

The value of n_{so} is now found as

$$n_{so} = \frac{C_{2D}(V_{po})_{2D}}{q} \tag{32}$$

The saturation velocity, v_{s2}, can be deduced from the drain-to-source current $(I_{ds})^s_m$ at $V_g = (V_g)_{max}$, where the transconductance reaches its maximum value. This value of the current is given by [4,6]

$$(I_{ds})^s_m = I_2 \frac{[(V_g)^2_{max} + V_0^2]^{1/2} - V_0}{(V_{po})_{2D}} \tag{33}$$

Eq. (33) is derived using the two piece model similar to Eq. (18). Here

$$(V_g)_{max} = (V_{gs})_{max} - (I_{ds})^s_m R_s \tag{34}$$

and $(V_{gs})_{max}$ is the measured value of the gate-to-source voltage at $g_m = (g_m)_{max}$. From Eq. (33) we find

$$v_{s2} = \frac{\mu_2}{L} \cdot \frac{(I^s_{ds})_m}{[K^2(V_g)^2_{max} - 2K(I^s_{ds})_m]^{1/2}} \tag{35}$$

where $K = \mu_{2D}C_{2D}/L$. The value of v_{s3} is not important because the "MESFET" operation is governed by Shockley's model due to the very low mobility (see Eq. (24)).

The value of the parameters determined in this fashion for three of our devices are given in Table II. The I-V characteristics calculated for devices 1 and 2 using these values of the parameters are shown in Fig. 6 and Fig. 7. As can be seen from the figures, the agreement with experimental data is quite good. We should note that device #1 has a thin undoped layer, $d_i = 20$ Å, and, hence, a large n_{so}. In this case the role of the parallel conduction in the (Al,Ga)As layer is quite small. Device #2 has $d_i = 100$ Å, which is quite large. As a result, n_{so} is relatively low and the role of the conduction in the doped (Al,Ga)As layer is quite considerable. The values of n_{so} determined from the drain conductance measurements (as discussed above) are in good agreement with our theoretical calculation [8]. All other parameters can be predicted with reasonable accuracy.

One exception is the low field mobility, μ_2, of the two dimensional electron gas at 300 K. As indicated in Table II, these values are consistently low. For devices 1 and 2 they seem to be close to the Hall mobilities we measured on the same wafers. However, we show in Reference [12] that the low values of the Hall mobility are caused by the contribution to the conduction from the doped (Al,Ga)As layer. When this contribution is taken into account, the deduced values of the Hall mobility for the two dimensional gas at room temperature are consistently close to 8000-9000 cm^2/Vs. Similarly, an apparent decrease of the low field mobility was observed in short gate GaAs devices [13,14]. Further studies are necessary to find out the causes for this anomalously small low field mobility in short gate devices.

V. CONCLUSION

We have developed an extended charge control model which includes the contribution to the drain-to-source current from the doped (Al,Ga)As layer. This contribution becomes important at large gate voltages and leads to a substantial decrease in device transconductance at large gate voltages. Our model is in good agreement with the device characteristics of high current normally-on MODFETs. The maximum current swing up to 300 mA/mm gate is demonstrated. This is the largest current swing yet reported for a MODFET and is very promising for high speed device applications.

A unified and complete characterization technique for deducing the parameters used in our model is introduced and utilized for device characterization. The deduced parameters such as the saturation velocity, equilibrium two dimensional gas concentration, thickness of the doped (Al,Ga)As layer, etc. are in good agreement with the independent calculations and measurements. However, the deduced values of the room temperature low field mobility of the two dimensional electron gas are considerably smaller than those obtained from the Hall measurements and studied of long gate MODFETs. A similar anomalous mobility reduction has been observed in short gate GaAs MESFETs.

The characterization and modelling technique presented here is well suited for computer-aided design of MODFET based digital IC's.

ACKNOWLEDGEMENT

The work at the University of Minnesota is partially funded by the Army Research Office and MEIS Center at the University of Minnesota. The work at the University of Illinois is funded by the Air Force Office of Scientific Research. The authors would like to express their appreciation of W. Kopp, R. Fischer, J. Klem and D. Arnold for contributions to this project.

REFERENCES

[1] C. P. Lee, D. L. Miller, D. Hou and R. J. Anderson, "Ultra High Speed Integrated Circuits Using GaAs/AlGaAs High Electron Mobility Transistors," Presented at the Device Res. Conf., Burlington, Vt., June 20-22, 1983.

[2] T. J. Drummond, S. L. Su, W. Kopp, R. Fischer, R. E. Thorne, H. Morkoç, K. Lee and M. S. Shur, "High Velocity N-on and N-off Modulation Doped GaAs/$Al_xGa_{1-x}As$ FETs," IEDM Tech. Digest 25, 586-589 (1982).

[3] K. Lee, M. S. Shur, T. J. Drummond, S. L. Su, W. G. Lyons, R. Fischer and H. Morkoç, "Design and Fabrication of High Transconductance Modulation Doped (Al,Ga)As/GaAs FETs," J. Vac. Sci. Technol. B, 1, 186-189 (1983).

[4] T. J. Drummond, H. Morkoç, K. Lee and M. Shur, "Model for Modulation Doped Field Effect Transistor," IEEE, Electron Dev. Lett. EDL-3, 338-341 (1982).

[5] D. Delagebeaudeuf and N. T. Linh, "Metal-(n)AlGaAs-GaAs Two Dimensional Electron Gas FET," IEEE Trans. Electron Dev. ED-29, 955-960 (1982).

[6] K. Lee, M. S. Shur, T. J. Drummond and H. Morkoç, "Current-Voltage and Capacitance-Voltage Characteristics of Modulation Doped Field Effect Transistors," IEEE Trans. Electron Dev. ED-30, 207-212 (1983).

[7] Michael Shur, Kwyro Lee, Kang Lee, Tho Vu, Peter Roberts and Max Helix, "Source Series Resistance of GaAs MESFET's," unpublished.

[8] K. Lee, M. S. Shur, T. J. Drummond and H. Morkoç, "Electron Density of the Two-Dimensional Gas in Modulation Doped Layers," J. Appl. Phys., 54, 2093-2096 (1983).

[9] A. J. Valois, G. Y. Robinson, K. Lee and M. S. Shur, "Temperature Dependence of the I-V Characteristics of Modulation-Doped FETs," J. Vac. Sci. Technol. B 1, 190-195 (1983).

[10] Michael Shur, "Low Field Mobility, Effective Saturation Velocity and Performance of Submicron GaAs MESFETs," Electronics Letters 18(21), 909-910, (1982).

[11] T. J. Drummond, W. G. Lyons, S. L. Su, W. Kopp, H. Morkoç, K. Lee and M. S. Shur, "Bias Dependence and Light Sensitivity of (Al,Ga)As/GaAs MODFETs at 77K," IEEE Trans. Electron Dev., in press.

[12] K. Lee, M. Shur, T. J. Drummond and H. Morkoç, "Room Temperature Mobility of the Two Dimensional Electron Gas in (Al,Ga)As-GaAs Modulation Doped Layers," unpublished.

[13] Kang Lee, M. S. Shur, Kwyro Lee, T. T. Vu, P. C. T. Roberts and M. J. Helix, "Low Field Mobility Profile in GaAs Ion-implanted FETs," unpublished.

[14] H. Fukui, "Determination of the Basic Device Parameters of a GaAs MESFET," Bell Sys. Tech. Jour., 58(3), 771-797 (1979).

FIGURE CAPTIONS

Fig. 1 A schematic cross-section of a MODFET illustrating the layer structure of the device.

Fig. 2 A qualitative sketch showing carrier dependence on gate bias. The interface carrier density of the two dimensional electron gas (2d-gas) and the sheet carrier density in the (Al,Ga)As MESFET region (3d-gas) are plotted vs. gate voltage. For illustration of parameters, refer to text.

Fig. 3 A pictorial diagram showing charge distribution in MODFETs. (a) At low gate bias in the range $V_{t2} < V_g < V_{t3}$, 2DEG density (n_{s2}) along the 2D channel is always less than n_{so}. (b) At high gate bias when $V_g > V_{t3}$, we have parallel conduction in (Al,Ga)As (n_{s3}) and $n_{s2} = n_{so}$ for $0 < x < L_1$, and $n_{s2} < n_{so}(n_{s3} = 0)$ for $L_1 < x < L$.

Fig. 4 The effect of finite n_{so} on transfer characteristic due to 2DEG. Maximum transconductance is obtained at $V_g = V_{t3} = V_{t2} + (V_{po})_{2D}$. The maximum drain current carried by 2DEG (I_2) is obtained when $V_g > V_{t3} + V_0$. (a) Transfer characteristic, (b) transconductance. The values of parameters used for calculation are $V_{t2} = -0.06$ V, $\mu_2 = 4{,}000$ cm^2/Vs, $v_{s2} = 1.5 \times 10^7$ cm/sec, L = 1 μm and Z = 290 μm.

Fig. 5 Comparison of transfer characteristics of two devices. Solid dots are experimental values. The lines are best theoretical fitting.

(a) shows no transconductance degradation, modelled well assuming $n_{so} = \infty$ (solid line).

(b) shows apparent transconductance degradation, modelled well assuming $n_{so} = 4 \times 10^{11}$ cm^{-2} (dotted line).

Fig. 6 (a) Transfer characteristic of device #1. Open circles and triangles are experimental points. Solid circles are the triangles ($V_g \geq 0$) shifted by −0.16 V (see text). The dotted line is a theoretical curve using data in Table II assuming two dimensional electron conduction only.

(b) Drain conductance data for device #1. Circles are experimental points and triangles are intrinsic drain conductance data assuming $R_S + R_D = 12\ \Omega$.

Fig. 7 (a) Transfer characteristic of device #2. Open circles and triangles are experimental points. Solid circles are the triangles ($V_g \geq 0$) shifted by −0.36 V (see text). The dotted line is a theoretical curve using data in Table II assuming two dimensional electron conduction only. The solid line is obtained assuming conduction by the two dimensional electrons and undepleted carriers in the (Al,Ga)As MESFET section.

(b) Drain conductance data for device #2. Circles are experimental points and triangles are intrinsic drain conductance data assuming $R_S + R_D = 20\ \Omega$. The data for $V_g > -0.8$ V are moved by −0.17 V to compensate for the threshold voltage shift.

Fig. 8 A qualitative sketch for drain conductance vs. gate voltage at low drain voltage (solid line). The two dashed lines correspond to Eqs. (29) and (30) respectively.

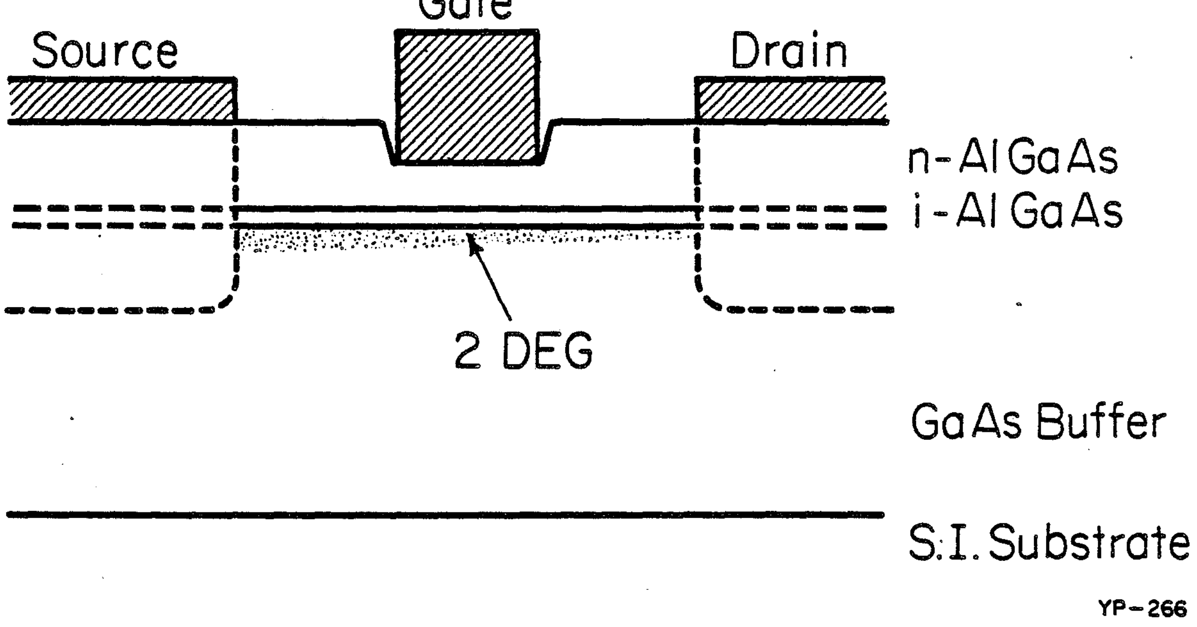

Gate
Source
Drain
n-AlGaAs
i-AlGaAs
2 DEG
GaAs Buffer
S.I. Substrate
YP-266

Fig. 1

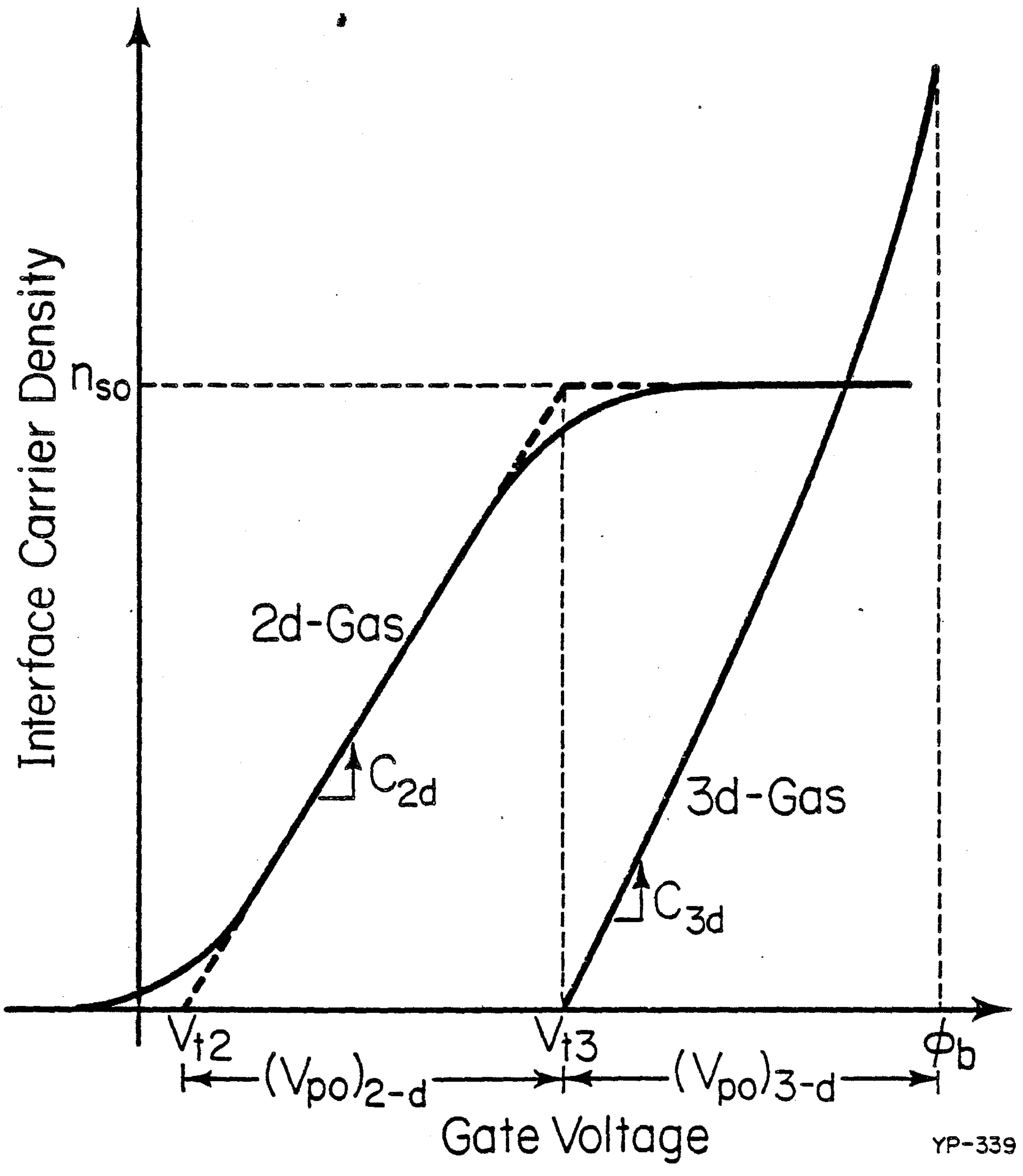
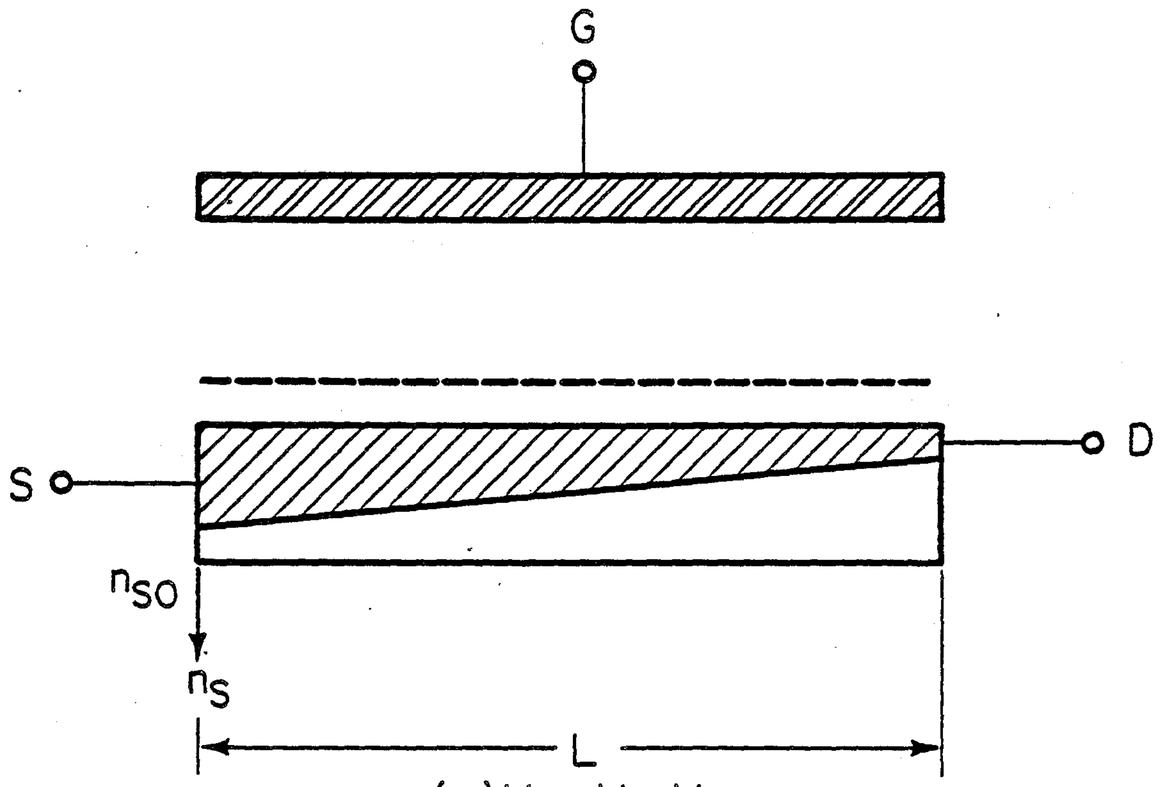
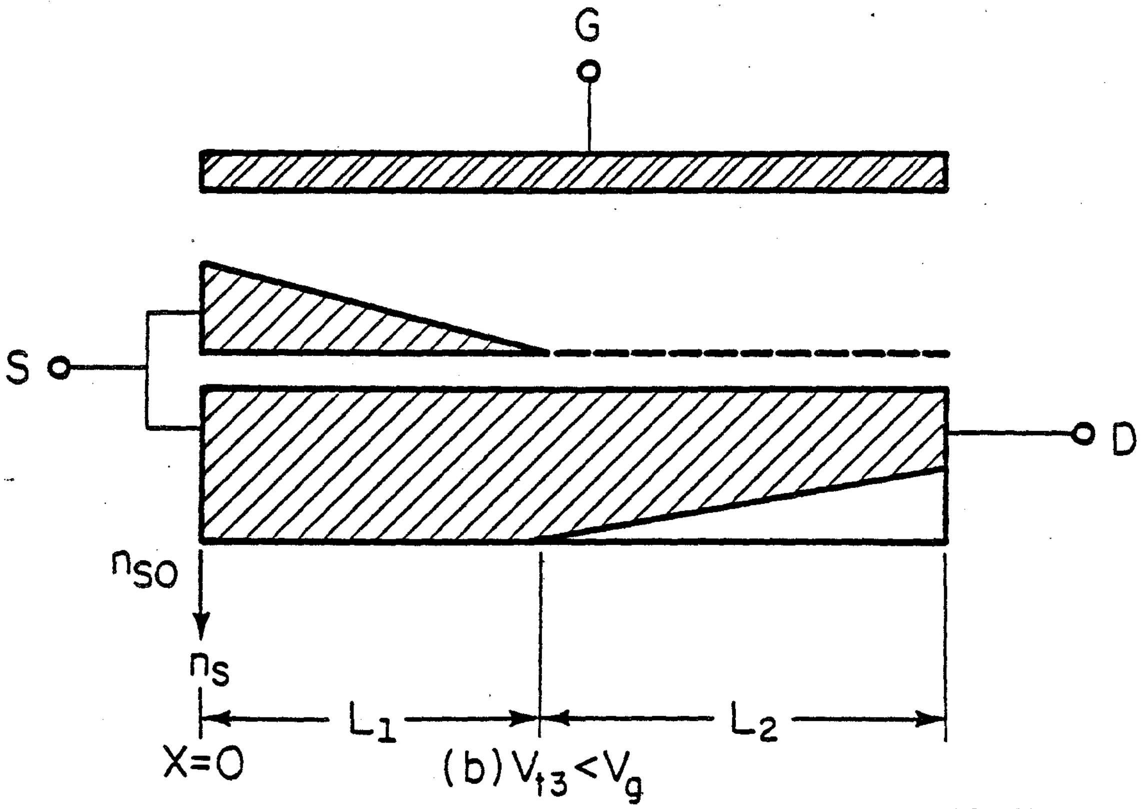

Interface Carrier Density
n_{so}
2d-Gas
C_{2d}
3d-Gas
C_{3d}
V_{t2}
V_{t3}
ϕ_b
$(V_{po})_{2-d}$
$(V_{po})_{3-d}$
Gate Voltage
YP-339

Fig. 2



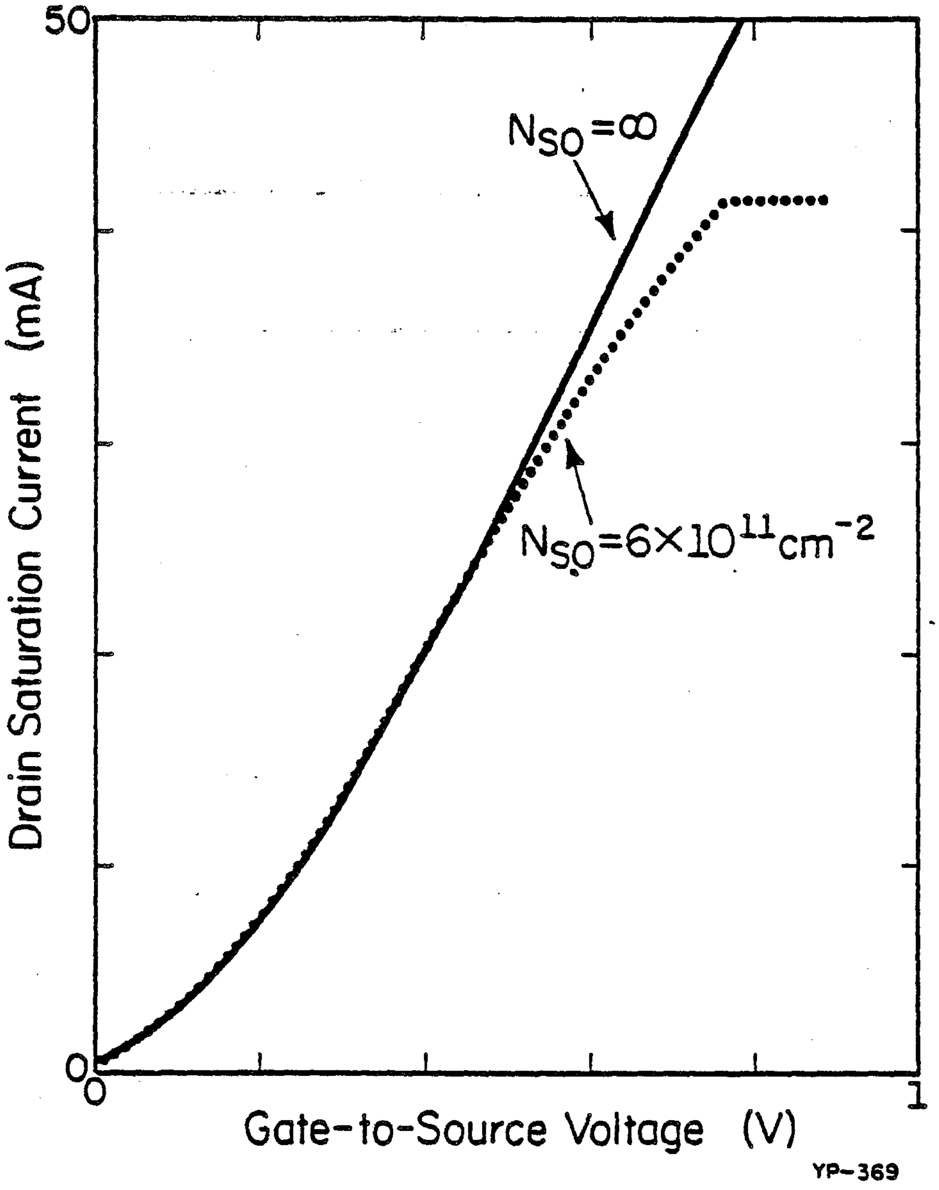
(a) $V_{t2} < V_g < V_{t3}$



(b) $V_{t3} < V_g$

YP–370

Fig. 3



50
0
0
1
Drain Saturation Current (mA)
Gate-to-Source Voltage (V)
$N_{SO}=\infty$
$N_{SO}=6\times10^{11}cm^{-2}$
YP-369

Fig 4a

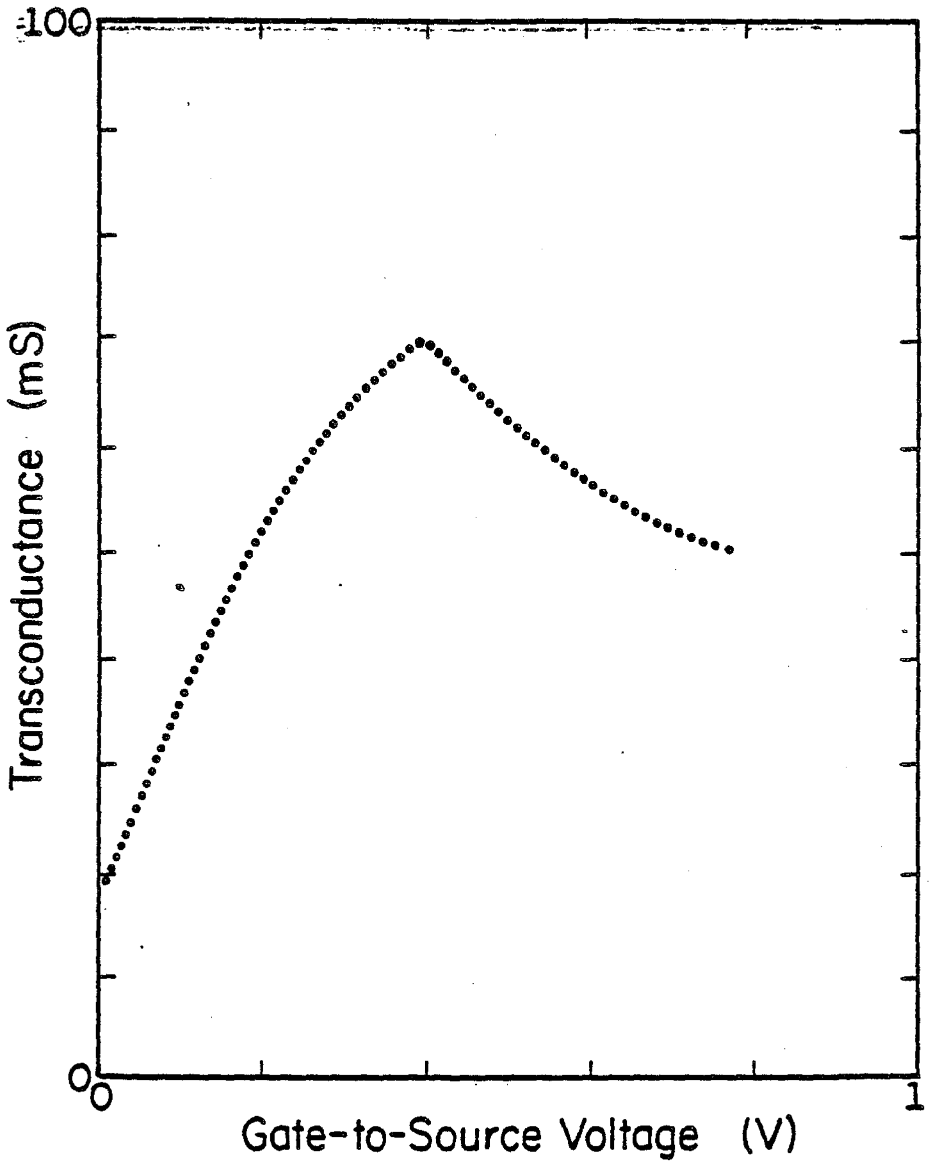
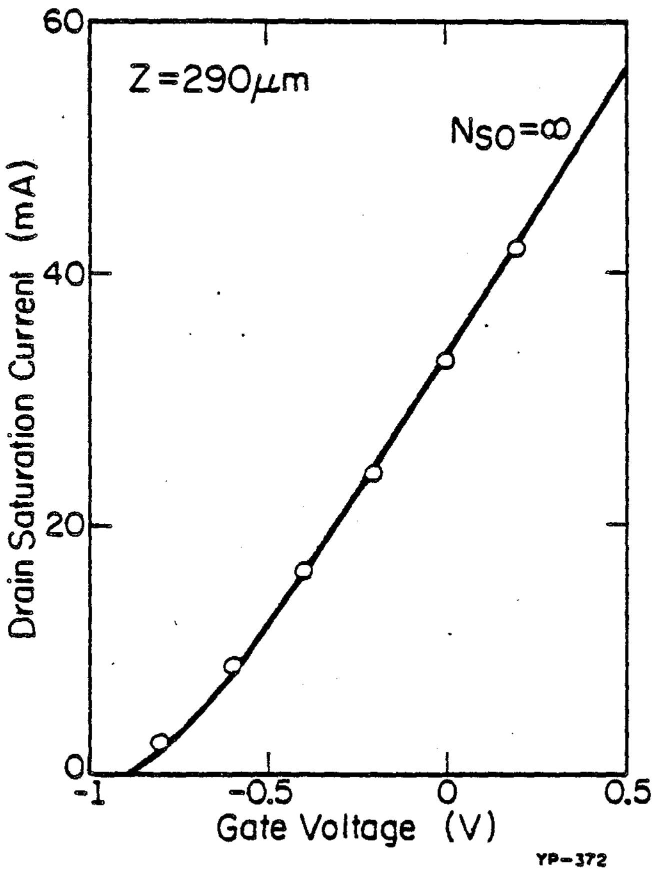
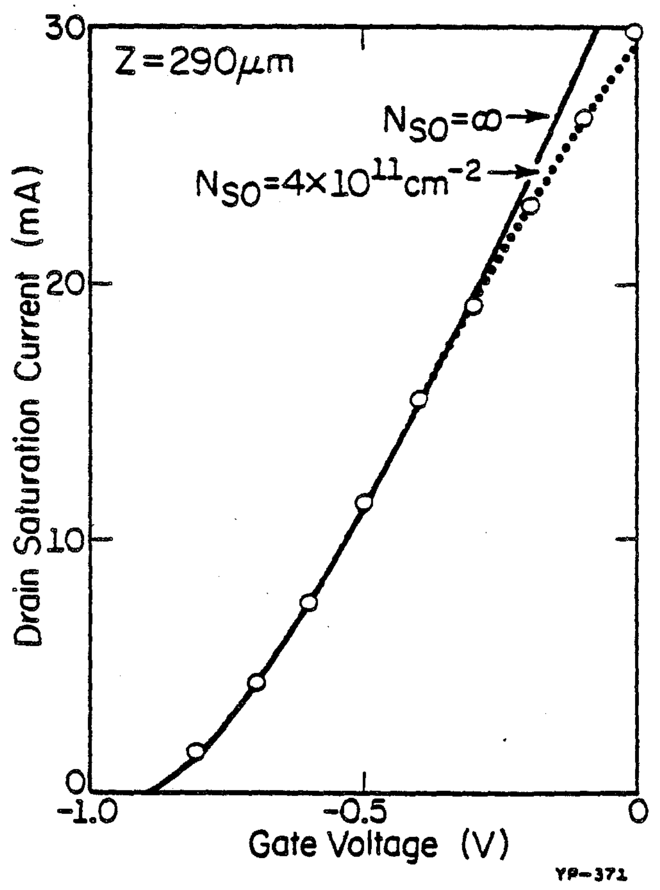


Fig 4b



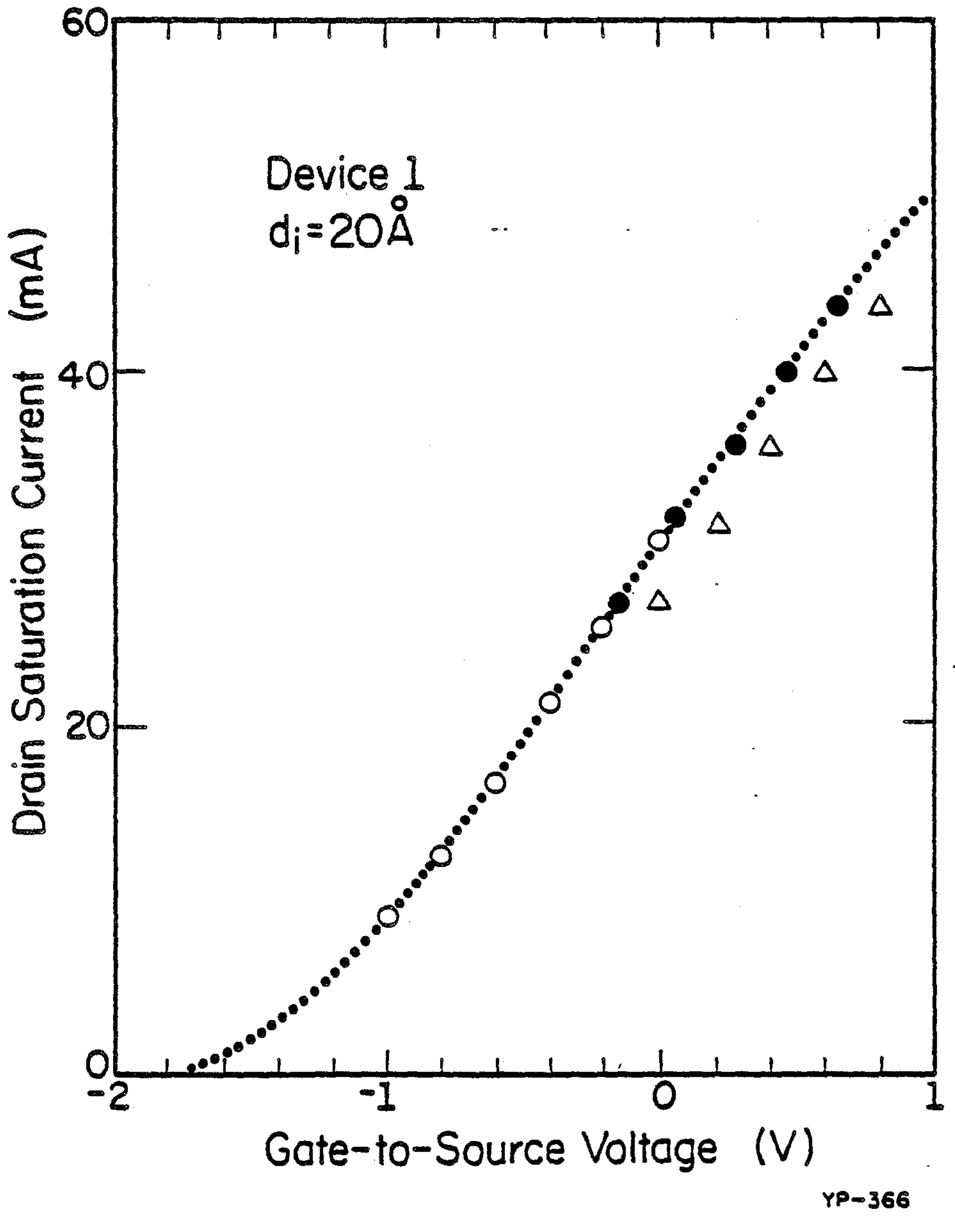
Z=290μm
Nso=∞
Drain Saturation Current (mA)
60
40
20
0
-1
-0.5
0
0.5
Gate Voltage (V)
YP-372

Fig 5a



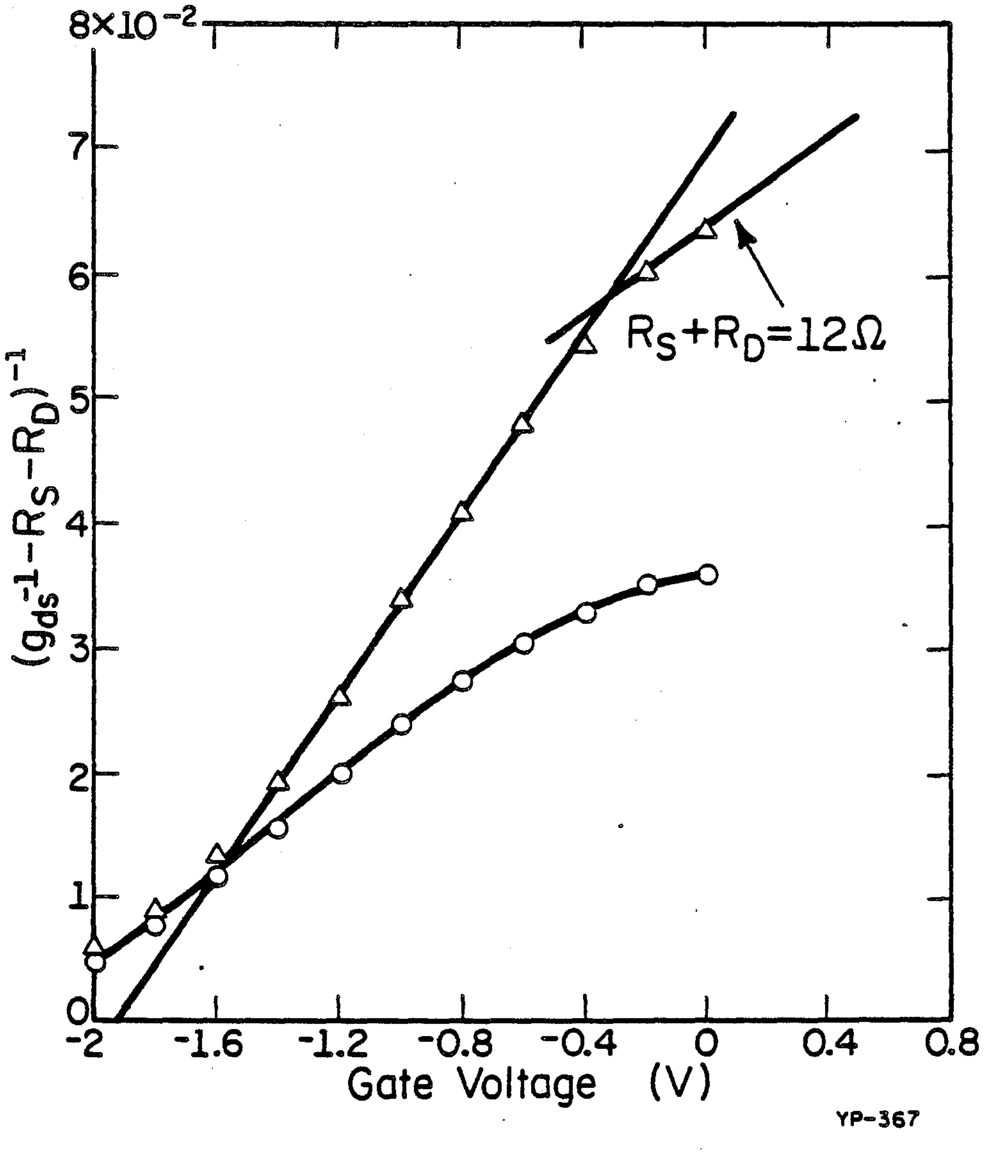
30
20
10
0
-1.0
-0.5
0
Z=290μm
$N_{SO}=\infty$
$N_{SO}=4\times10^{11}cm^{-2}$
Drain Saturation Current (mA)
Gate Voltage (V)
YP-371

Fig 5b



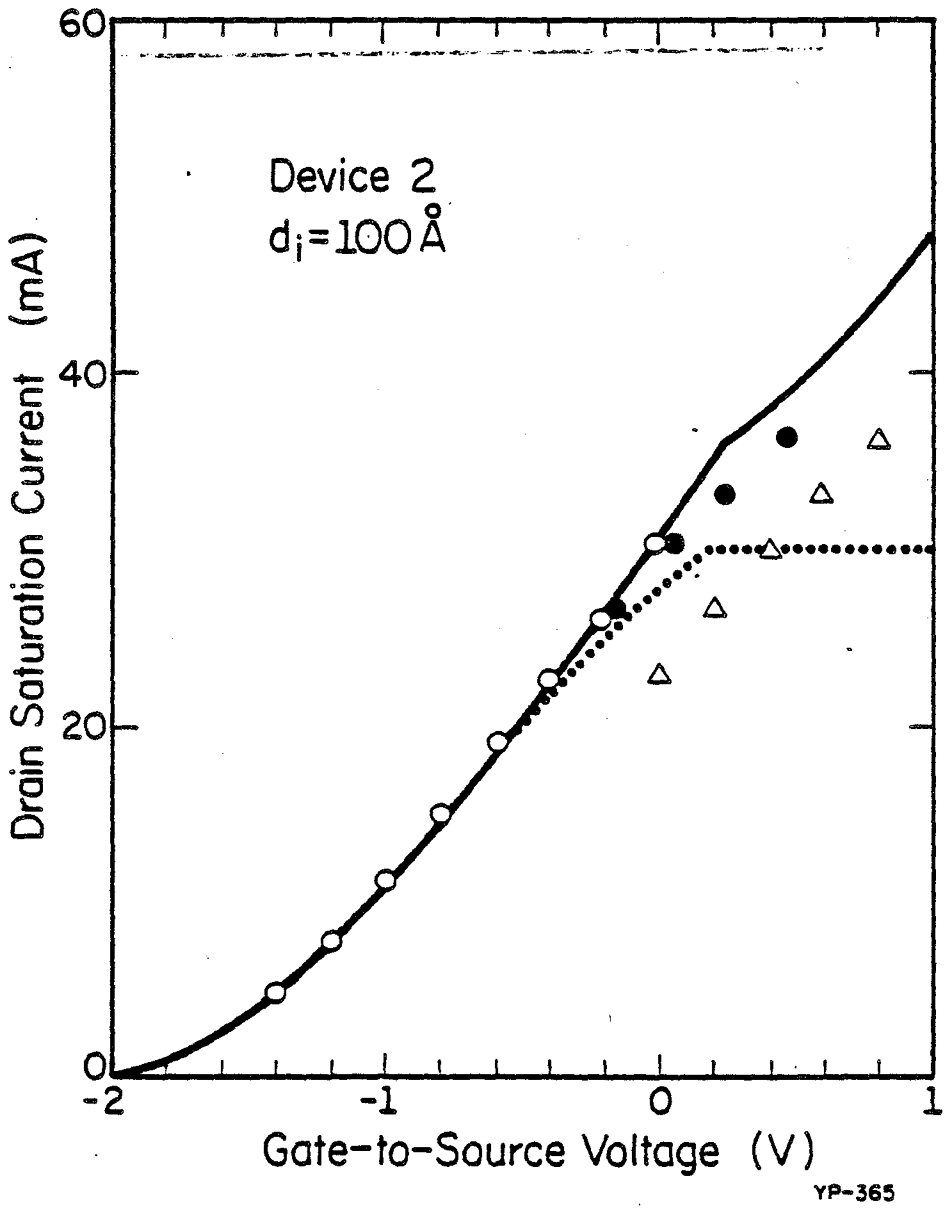
60
40
20
0
-2
-1
0
1
Device 1
d_i=20Å
Drain Saturation Current (mA)
Gate-to-Source Voltage (V)
YP-366

Fig 6a



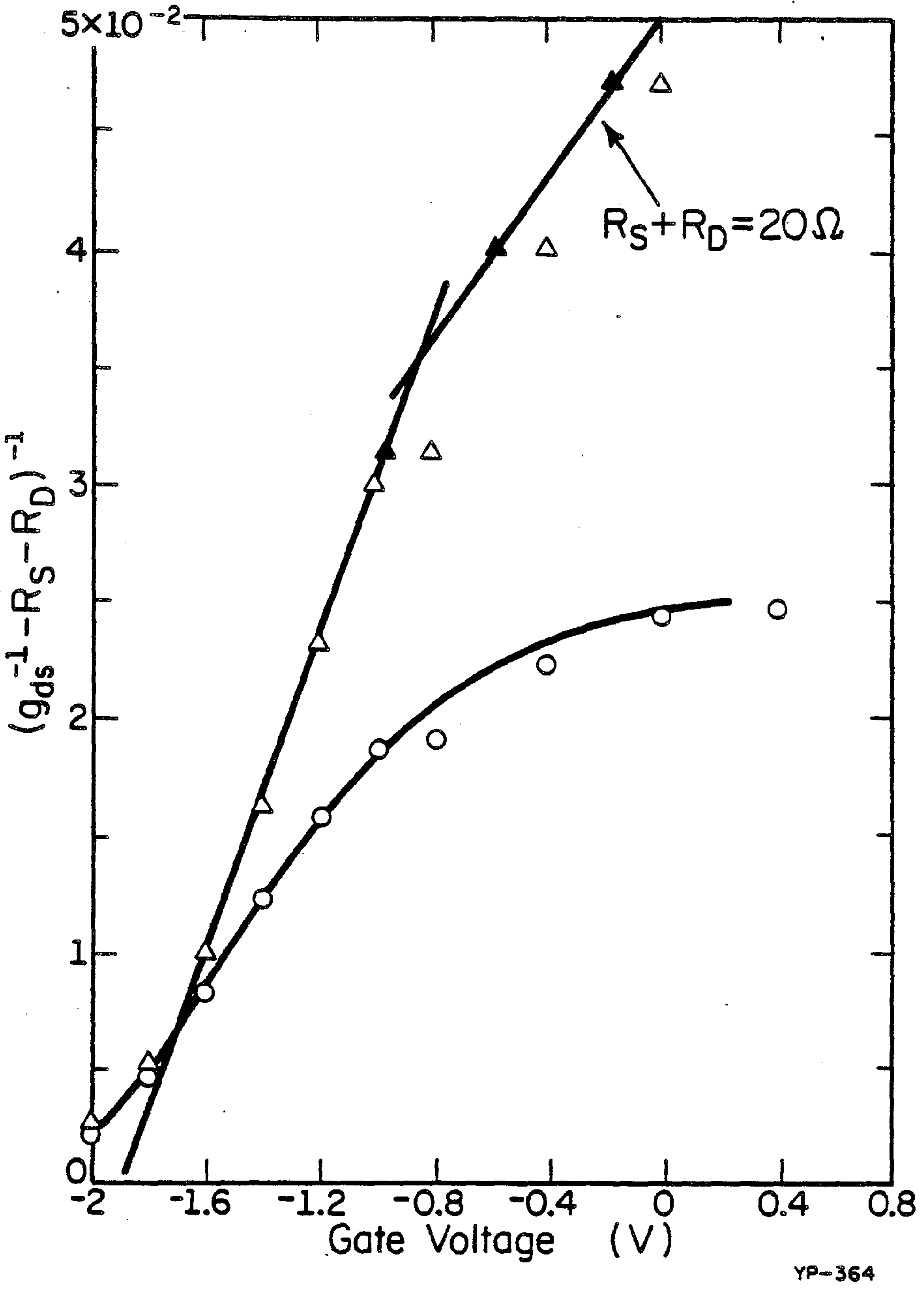
8×10^-2
7
6
5
4
3
2
1
0
-2
-1.6
-1.2
-0.8
-0.4
0
0.4
0.8
Gate Voltage (V)
(g_ds^-1 - R_S - R_D)^-1
R_S+R_D=12Ω
YP-367

Fig 6b



Device 2
$d_i = 100\,\text{Å}$
Drain Saturation Current (mA)
Gate-to-Source Voltage (V)
60
40
20
0
-2
-1
0
1
YP-365

Fig 7a



5×10⁻²
4
3
2
1
0
$(g_{ds}^{-1}-R_S-R_D)^{-1}$
$R_S+R_D=20\Omega$
-2
-1.6
-1.2
-0.8
-0.4
0
0.4
0.8
Gate Voltage (V)
YP-364

Fig 7 b

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

ANALYSIS AND DESIGN IN MSG: FORMALIZING FUNCTIONAL SPECIFICATIONS

Microelectronic and Information Sciences Center

Technical Report #09

V. A. Berzins
Computer Science Department
University of Minnesota

M. Gray
Computer Science Department
University of Minnesota

ANALYSIS AND DESIGN IN MSG:
FORMALIZING FUNCTIONAL SPECIFICATIONS

MEIS Technical Report #09

V. A. Berzins
M. Gray

This research reported in this paper was supported in part by the Microelectronic and Information Sciences Center.

C 1984?

ABSTRACT

Model building is identified as the most important part of the analysis and design process for software systems. A set of primitives to support this process are presented, along with a formal language for recording the results of analysis and design. The semantics of the notation is defined in terms of the actor formalism, which is based on a MeSsaGe passing paradigm. The automatic derivation of a graphical form of the specification for user review is discussed. Potentials for computer aided design based on MSG are indicated.

1. Introduction

The software development process is most commonly represented by the software life cycle model. There is a fair amount of agreement that the early stages of the software life cycle consist of requirements definition, functional specification, and design. There is a lack of agreement on the meaning of these terms, and no precise definition of the activities that are performed in each stage. The result is often a chaotic process, where all aspects of design are mixed together, resulting in software products that have cumbersome and complicated user interfaces and that do not satisfy user needs.

We believe the early part of the life cycle consists of requirements analysis, functional specification, architectural design, and module design. Requirements describe the problem. They constrain both the product (eg. functionality, performance, reliability) and the development process (eg. cost, schedule, methodology), and can generally be met by a number of different systems. The functional specification describes the solution: the user interface of the system to be implemented. The functional specification gives only an external view of the system, while a design contains the design concepts and software structures to be used in implementing the system, which are needed by the implementors but not by the users. The goal of architectural design is to identify the software modules to be used in the implementation and to give precise black box descriptions of their behavior. Module design identifies data structures and algorithms for implementing each module.

In this paper we develop the concepts needed to support a disciplined approach to functional specifications. These concepts are embodied in MSG, a specification language supporting the functional specification and design stages of the software development process. The aspects of MSG relevant to functional specifications are described here, and the aspects relevant to design will be discussed in a later paper.

We have limited our discussion to the life cycle approach since it is widely used and better understood than alternative models of software development such as rapid prototyping[1].

1.1. Definition of Functional Specification

A functional specification is a conceptual model of the proposed software system, which presents only those aspects of the system relevant to the users of the system.

1.2. Requirements for a Functional Specification Language

There are two distinct audiences for functional specifications: the analysts/designers and the customers/users. The analysts/designers need a formal and precisely defined language for constructing and recording models of software systems. The customers/users need to review the specification without learning languages or concepts foreign to the problem domain. It is unlikely that the needs of both audiences can be met by the same notation. We recommend that the documents to be reviewed by the customers be derived from a formal specification, with as much mechanical assistance as possible. Computed diagrams can serve to present an overview of the system, while details can be presented in a prototype user's manual derived from the formal specification by hand; extra human effort is not implied, because the user's manual will be needed anyway.

The formal notation must support the model building process in a number of ways.

(1) The notation must have a precise and unambiguous semantics.

(2) The notation must be based on a clean and coherent model of computation.

(3) The notation must be sufficiently powerful to express all relevant aspects of system behavior.

(4) The notation must support a convenient set of abstractions for building models of software systems. Abstractions are vital for suppressing detail without sacrificing precision. A well chosen set of building blocks can considerably ease the burden of the analyst.

(5) The notation must be irredundant, to avoid update inconsistencies.

(6) The notation must be modular and support effective searching.

(7) The notation must support partial descriptions, so that it may aid in developing a model and completeness checking.

(8) The notation must be easy to understand and modify. Small conceptual changes should correspond to small changes in the text.

(9) The notation must be relatively easy to learn. Since it will be used by professionals, some specialized training can be required.

(10) The notation must apply to a broad range of applications, so that many different notations are not needed.

(11) The notation must apply to a large part of the life cycle, so that translation from one form to another is avoided. We believe a closely related set of notations can span the functional specification, architectural design, and module design stages.

(12) The notation must support mechanical processing. This goal is sufficiently important to justify some compromises with respect to the previous goals.

1.3. Previous Work

The PSL/PSA system [2] has been used to support natural language functional specification. It formalizes some aspects of the natural language as the objects, relations, and attributes of a relational database. The regular structure of the database allows the automatic generation of indexes, summary reports, and graphics, as well as some consistency and completeness checking. Natural language descriptions are associated with the objects and structures of the database. The meanings of the objects and relationships in the database are left loosely defined, and cannot be made precise without sacrificing usefulness, because the database has a fixed schema. Mechanical processing can apply only to those aspects of the specification that have been captured in the database structures. Such a system can never *guarantee* that a functional specification is complete, consistent, or irredundant in the usual sense of those words.

The SREM system [3] introduces an explicit timing model, which uses R-nets (response networks) to identify events subject to timing constraints, and provides a simulation facility that helps show the feasibility of meeting real time constraints. The resulting increase in precision is its main advantage over PSL/PSA. The system has a relatively narrow but important class of applications (embedded systems). It has very little support for abstractions and modularity, and has no clear underlying model of computation.

Another approach is the structured analysis method of DeMarco [4]. This technique portrays a software system as a network of data transforming processes linked by data flow paths. Descriptions of the processes and flows are usually written in pseudocode or natural language. The technique is easily understandable by end users because it focuses attention on essential issues. Networks of data transformations are not well suited for describing systems dominated by chains of state transitions. This technique also does not make use of data abstractions. As in the PSL/PSA system, mechanical processing can only apply to the network structure, with no guarantees of consistency, completeness, or irredundancy.

Ross[5] describes another method, known under the copyrighted name SADT, which is similar to structured analysis. SADT includes arcs representing control flow and sequencing information in their networks, while retaining much of the understandability of data flow diagrams. Although this approach removes some of the difficulty describing systems of state machines, the problems of handling abstract data types and limited mechanical computability remain.

Functional specification based on data structures is advocated by Orr [6]. This technique lays out the data structures at the implementation level and attempts to deduce data transformations that will map input structures into output structures. Such a technique does not support abstractions, thereby fixing implementation details too early in the analysis phase.

We believe that no significant progress beyond the existing approaches will be made until a mathematically rigorous and semantically precise specification language based on the concepts of abstraction and iterative refinement is available. Such a language must provide a variety of review representations including diagrams. The language must also be general enough to specify state transformation functions as well as data transformation functions. Work at SRI on the Hierarchical Development Methodology (HDM) has produced such a language known as SPECIAL[7], which is used in architectural design to give precise black box specifications for the individual modules, and in module design for (mechanically) verifying that a particular implementation satisfies those specifications. The work reported here was motivated by the difficulty of applying the SPECIAL concepts and approach to the functional specification stage.

1.4. Overview

Section 2 discusses the concepts and abstractions needed for constructing models of software systems and Section 3 shows how MSG supports these concepts. Section 4 describes how diagrams suitable for user review can be constructed from an MSG specification. Section 5 indicates what kinds of mechanical processing are desirable, while Section 6 presents our conclusions.

2. Models of Software Systems

Models of software systems are used to express and to analyze the design at various levels. A model of the existing system is often built to aid in determining and analyzing the requirements. A model of the proposed system is built during the functional specification phase, and models of the internal software modules are built during the architectural design phase. Model building absorbs most of the effort in the early stages of software development, and a language for expressing models of software systems is the basic tool of the analyst and designer.

Many of the existing techniques for recording functional specifications, such as bubble charts [4], R-nets [3], and PAISLey [8] model software systems as collections of communicating processes. The first two techniques do not have a precise semantics, while the last constructs an applicative prototype implementation, which sometimes forces specifying irrelevant detail, and does not provide for documenting restrictions and assumptions. The communicating process paradigm seems to be appropriate, but the domain must be characterized carefully and a notation with a precise and appropriate semantics must be developed.

We believe that software models should be expressed in terms of a strongly typed version of the actor model of computation[9]. Our contribution is a set of primitives suitable for constructing functional specifications, and an actor based definition of the semantics of those primitives. The rest of this section reviews the actor formalism and shows how the specification primitives can be defined.

2.1. Actors, Messages, and Events

In Hewitt's model, everything is an actor, including procedures, data, tasks, and messages. Actors are black boxes that can send and receive messages. Messages are guaranteed to arrive some finite time after they were sent. The behavior of a system of actors can be captured by recording the messages received by the actors in the system. This approach focuses on interfaces and on externally observable behavior rather than on the internal mechanisms for realizing that behavior. Formally, the *behavior* of an actor system is a partially ordered set of events.

An *event* occurs when an actor (the target of the event) receives another actor (the message of the event), and the two interact. Events are instantaneous and occur at discrete points in time. Each event *activates* finitely many (zero or more) other events, by sending messages to the target actors of those events. If *E1 activates E2* then the event *E2* occurs after and was caused by the event *E1*. *Activates* is a strict partial ordering relation on events with at most one immediate predecessor for each event, so that the history of a computation can be represented by an activation tree (or forest). Each event corresponds to the arrival of a single message, and the immediate predecessor is the event in which that message was sent, if the message originated inside the system. The number of direct and indirect predecessors of any event is finite. The root of the tree is the event that caused or triggered the rest of the computation, corresponding to the receipt of a request message originating outside the system. The activation ordering is under the designer's control, and can be used in either a descriptive (recording what happened) or prescriptive (specifying what is supposed to happen) way. The behavior of an actor can be specified by describing the set of events to be activated by the arrival of a message, as a function of the contents of that message, and possibly also of the sequence of prior messages received by the actor.

The messages arriving at an actor are assumed to be totally ordered by the *arrival* ordering, so that the sequence of messages received by an actor is a well defined concept. This assumption implies that messages are always received one at a time, even though they may originate from several independent processes, and in the case of a parallel implementation requires reliable arbitration hardware. An actor can accept the next message only after the state changes due to the previous message have taken effect and the messages activated by the previous message have been sent. The arrival ordering is needed to define actors with internal states. The behavior of such actors depends on previous messages, and is sensitive to the order in which those messages arrived. The arrival order can be observed but cannot be precisely predicted, and can be controlled by the designer only by introducing activation orderings among the arrival events. Unpredictable transmission delays are the sole source of nondeterminism in the model.

Actors are locally deterministic and independent, in the sense that the set of events activated by the arrival of a message *M* at an actor *A* is a function of the sequence of messages received by *A* up to and including *M* [10]. Actors are independent in the sense that the messages sent by an actor are completely determined by the messages received by that actor. This implies actors have no hidden interactions, because the behavior of an actor cannot be affected by the messages received by other actors. We believe that the components of a functional specification must have this independence property for the sake of clarity and ease of analysis.

Some care is required in aggregation, because the independence property is not automatically preserved when a system of actors is treated as a single actor. A sufficient condition for independence is that all messages crossing the boundaries of the subsystem must be included in the sequence of messages received by the actor representing the entire subsystem, and that those messages do not contain objects that are capable of changing state. This condition is stronger than necessary, because it implies independent subsystems do not share subcomponents. Actors without internal states may serve as components of two distinct independent subsystems, because the behavior of such subsystems is the same as if each subsystem had its own copy of the shared actor.

2.2. Abstractions for Functional Specifications

A specification language should support the kinds of abstractions commonly used in describing software systems. The abstractions needed for functional specification are transforms, state machines, and data types. This section shows how these abstractions can be modeled as actors. The examples in the next section show that these building blocks are sufficient to describe most current software systems.

2.2.1. Transforms

A *transform* is an actor that computes a mathematical function, in the sense that the events activated by a message depend only on that message, and not on the contents of earlier messages. A transform has no memory or internal state. There may be more than one response to a single request, in which case the responses are not causally ordered, and the potential for a parallel implementation exists. Responses can either be sent to the actor that originated the request, as happens when a subroutine returns a value, or they can be sent to other actors, as happens in a system with a pipeline structure.

2.2.2. State Machines

A *state machine* is an actor that exhibits internal memory, in the sense that the events activated by a message depend on earlier messages. Many software systems contain databases or provide commands that allow users to modify internal states in ways that affect the behavior of the system. Examples are inventory control systems, operating systems, and flight simulators. Such systems are naturally modeled as state machines, where user commands or transactions correspond to messages received by the state machine, and the answers produced by the system correspond to the reply messages sent by the state machine.

2.2.3. Types

A *data type* or data abstraction consists of a set of data objects and a set of operations for creating and manipulating those objects. Data types are used in functional specifications in modeling messages and states of state machines. The data types common in mathematics and in programming languages should be defined once and kept in a specification library, while the data types specific to the problem area will have to be defined for each application. The ability to define new types is important in cases where the standard types do not naturally describe the meaning of the data, or where the meaning of the data will be gradually extended. Using the data types natural to a problem can greatly improve the clarity of a specification. For example, a natural data type to use in modeling an airline reservation system is a *flight*. Defining the meaning of a new data type in terms of the operations provided rather than in terms of a data structure has the advantage of decoupling different kinds of interactions with the data. New operations can be added or existing ones modified without affecting the meaning of the other operations, or the conceptual model of the data.

In an actor model for a data type, the operations of the type correspond to messages recognized by the actors representing the type. The actor model can be organized in at least two ways.

The first way [10] is to model the entire type as an actor, and the instances of the type as a distinct set of actors. The actor representing the type as a whole recognizes and responds to messages that create new objects of the type. The actors representing the instances of the type recognize messages that manipulate existing objects of the type. This kind of model is natural for an object oriented style of programming, which is supported in languages such as SMALLTALK[11,12] and versions of LISP with *flavors* [13].

The second way is to model the type as an actor that recognizes messages corresponding to all of the operations of the data abstraction. In this case the instances of the type have a more passive role, and can be represented either as actors or as elements of appropriately chosen abstract sets. This kind of model corresponds to the style of programming natural in most other languages supporting data abstractions, such as CLU[14], ALPHARD[15], ADA[16], and so on.

We believe that the concepts and paradigms of the specification language should compatibly extend those supported by the programming language. Artificial encodings should be avoided, because they add to the intellectual burdens of the analysts and designers, whose capacity is typically strained to the limit as it is. Since the second class of programming languages is more widely used, we have developed the second approach to modeling data abstractions in this paper. Reworking the specification language along the lines of the first approach for applications where that is more natural does not present any difficulties.

We believe that only immutable data abstractions, whose objects cannot be modified, should be used in functional specifications, because the resulting specifications are easier to understand and to analyze, and because it is easier to establish the independence property for subsystems of actors. The subset of MSG described here does not allow the definition of mutable data types.

3. The Description Language MSG

This section presents the subset of of MSG needed for functional specifications. Full MSG also provides support for design, including facilities for describing mutable types, iterators[17], and algorithms. The use of MSG in the design stage will be discussed in a later paper.

MSG is a means for defining actors, supplying primitives for modeling data and machine states and supporting the concepts defined in the previous section. MSG provides a set of primitives for describing both the data and the control aspects of a machine state. The semantics of the language will be discussed informally, using examples, rather than giving a formal definition of the syntax and of the mapping from MSG module specs into classes of actors.

3.1. Modeling Data

An important part of functional specification is modeling the data in the system, to capture the aspects of that data that will be visible to the user of the system. Abstraction is essential at this stage, because it is important to pin down the information content of the data without getting bogged down in the design of storage structures. There are two well known ways to define data abstractly, based on axioms and abstract models.

Axiomatic definitions of data are well suited for doing proofs[18,19,20,21], but they are difficult to understand and to modify without introducing inconsistencies or unwanted effects, even for people with extensive mathematical training.

An abstract model is a conceptual representation for the data, intended to capture the information content of the data in the simplest possible manner. For example, an appropriate model for a symbol table is a mapping from identifiers to attribute values. Such a model pins down the meaning of the data without ruling out any of the large number of data structures that can be used to implement the type. The abstract model is chosen for simplicity and directness of description, and is usually much simpler than the data structures of the implementation, which are engineered for space and time efficiency. Conceptual representations are used in defining the operations of the data type, and in expressing assertions about the objects of the type. Unlike the storage representation of an abstract data type, which is private to its implementation, the conceptual representation must be public. The conceptual representation may be needed in assertions because the operations provided by the type may be sufficient to compute the desired results but may not be sufficient for describing the essential properties of those results. The axiomatic approaches have been forced to introduce "hidden functions" to deal with this problem[22].

Abstract models provide a means of visualizing and informally reasoning about the data objects of the type. We believe that people really understand an unfamiliar data type only when they can construct a conceptual representation. Formal reasoning based on abstract models has been studied[23,24,10,25], and there appears to be no essential difference in the difficulty of constructing proofs using the two techniques, so that ease of understanding and modification are the determining factors in the choice of a data modeling technique.

3.1.1. Building Blocks

MSG supports the abstract model approach for specifying data. The basic building blocks for conceptual representations in MSG are generated by a set of basic types and a set of domain constructors. The basic types include a number of well understood mathematical domains, such as truth values, integers, real numbers, strings, and the singleton domain null = {nil}, as well as any previously defined abstract types. The domain constructors are standard mathematical operations such as Cartesian products, disjoint unions, sequences, power sets, function spaces, and relation spaces. The domain constructors can be applied recursively as well as to previously constructed types. Recursive domain equations involving function spaces and power sets do not lead to subtleties involving cardinality as they do in denotational semantics[26,27] because all of our compound objects contain only finitely many subcomponents.

All of the built in types correspond to the standard mathematical notions - size and precision are unlimited and instances of the built in types cannot be modified. All restrictions and state changes are explicitly described by MSG assertions.

3.1.2. Notation

Fig 1 shows the domain constructors available in MSG. D1 and D2 are domains. **Tuple**(a: D1, b: D2) denotes the set of all labeled pairs, where the first element of each pair has the label "a" and belongs to D1, while the second element has the label "b" and belongs to D2. **Oneof**(a: D1, b: D2) denotes the disjoint union of D1 and D2, where elements of D1 carry the label "a" and elements of D2 carry the label "b". **Map**(from: D1, to: D2) denotes the set of all mappings from finite subsets of D1 to D2. Maps are single valued, while relations can be many to many. There can be any finite number of labeled components in a tuple, oneof, or

set (D1)	finite subsets of D1
sequence (D1)	finite sequences from D1
tuple (a: D1, b: D2)	labeled Cartesian product
oneof (a: D1, b: D2)	disjoint union
map (from: D1, to: D2)	finite mappings from D1 to D2
relation (a: D1, b: D2)	set(tuple(a: D1, b: D2))

Fig. 1 Domain Constructors

relation type.

Individual sets are written by listing the elements enclosed in "{}", while sequences are enclosed in "[]". Within such a context, an unadorned variable denotes an element, while a variable prefixed with a "!" denotes a (possibly empty) subset or subsequence. An element of a disjoint union is written as "tag: value", where the ":" is an infix pairing operator. Mappings are treated and written as sets of pairs, so that "{1: 2, 2: 4}" denotes the function that maps 1 into 2, 2 into 4, and is undefined for all other values. Tuples are treated as mappings from labels to components values, and the dot notation for record components common to many programming languages is supported, so that if x denotes the tuple {a: 3, b: 7} then x.a = x(a) = 3. The order of the pairs in a tuple is not significant.

3.2. Data Type Example

The basic features of the language are illustrated in Fig. 2, which shows a model of the text_block data abstraction, which is useful for describing output layouts. MSG keywords and built-in types and functions are shown in **boldface.** This example was inspired by the display description facilities in Descartes[28], greatly restricted and simplified. We assume that text and spaces consist of characters with a uniform height and width. The text_block type provides operations intended to make it easy to write specifications of two dimensional layouts, which can get awkward without geometrical abstractions. We feel the difficult concepts in functional specifications should be isolated in formally identified abstractions, which are easy to use although they may be hard to define.

A text_block is a rectangular region of text (comments in MSG extend from the first "%" on a line to the end of the line). The conceptual representation of a text_block has three components, the height and width of the rectangle and the text it contains. The assertion in the **where** clause attached to a **model** declaration is called a *data invariant* because it must be satisfied by all valid conceptual representations for the type. In the example the data invariant requires the rectangle to have non-negative dimensions and the text to be defined throughout the rectangle.

The text is represented as a map from coordinates to characters, resembling a two dimensional array. Maps are similar to arrays, except that the index set can be any data type and the associated values cannot be modified. Maps are very useful in modeling. For example, a symbol table is a map from character strings to attribute values.

The *create* operation creates a one line text_block from a given string. The *above* and *before* operations create composite blocks with a vertical or a horizontal alignment, respectively. The *window* operation extracts subregions of a larger block, and the *image* operation makes textfiles suitable for printers with a bounded width from text_blocks that can be arbitrarily wide.

The behavior of an actor is defined by describing all of the primitive actions it may perform. A primitive action consists of the arrival of a message and the sending of a finite number of responses. A message can have a name, which indicates the operation to be performed, and a finite number of components, which must be named if there is more than one. Descriptions of message layouts are introduced by the keywords **receive, reply,** and **send.** The tuple that follows gives the names and types of the components of the message, and specifies the interface information that would appear in the heading of a procedure definition in a typical programming language. The message produced by a **reply** goes back to the actor that sent the request message, while that produced by a **send** goes to an explicitly named actor. **Receive**, **reply**, and **send** can be followed by an assertion in a **where** clause, which restricts and describes the values that may be taken by the components of the message. The assertion attached to a **receive** is a precondition or a guard, while the assertion attached to a **reply** or **send** is a postcondition. Compound actions are variants of guarded commands[29] composed of other actions, either primitive or compound. The example shows the conditional form (**cases**) of a guarded command. There is also an iterative form that is used in defining state machines. As in [30], termination of an iteration must be explicitly indicated by a keyword (**exit**) attached to one of the alternatives. If all of the guards are false, nothing happens pending the arrival of a message, as in [31]. If several guards are true, the results are

```
type text_block                                              % rectangular blocks of text
model {height, width: integer,                               % row #1 top, col #1 left
       text: map(from: {row: 1..height, col: 1..width}, to: char) }
where height >= 0, width >= 0, total_function(text)
behavior
 cases
  receive create{line: string}
  reply b: text_block
  where b = {height: 1, width: length(line), text: t},
        t(row: 1, col: c) = line[c] for all c in indices(line)
 or  % extract a sub-block with given bounds
  receive window{b: text_block, rmin, rmax, cmin, cmax: integer}
  where b = {height: h, width: w, text: t}
  reply bb: text_block
  where bb = {height: hh, width: ww, text: tt},
        hh = max(1+rmax-rmin, 0), ww = max(1+cmax-cmin, 0),
        if r in 1..hh and c in 1..ww and r+rmin-1 in 1..h and c+cmin-1 in 1..w
        then tt(row: r, col: c) = t(row: r+rmin-1, col: c+cmin-1)
        else_if r in 1..hh and c in 1..ww then tt(row: r, col: c) = " " end_if
 or  % put one block above another with a given space in between
  receive above{top, bot: text_block, space: integer}
  where top = {height: h1, width: w1, text: t1},
        bot = {height: h2, width: w2, text: t2}
  reply b: text_block
  where b = {height: h, width: w, text: t}, s = max(space, 0),
        h = h1+s+h2, w = max(w1, w2), t(row: r, col: c) =
           if r in 1..h1 and c in 1..w1 then t1(row: r, col: c)
           else_if r-(h1+s) in 1..h2 and c in 1..w2
             then t2(row: r-(h1+s), col: c)
           else_if r in 1..h and c in 1..w then " " end_if
 or  % put one block left of another with a given space in between
  receive before{left, right: text_block, space: integer}
  reply b: text_block    % like above, rows and columns switched
 or  % print out a block in vertical strips of max width pw
  receive image{b: text_block, pw: integer}
  where b = {height: h, width: w, text: t}
  reply d: sequence(string)                                  % sequence of text lines
  where if h = 0 or w = 0 or pw <= 0 then d = []             % empty window
        else_if w <= pw                                      % block width <= page width
        then d = [line, !dd],
             line[c] = t(row: 1, col: c) for all c in indices(line),
             length(line) = w, dd = image(b: bb, pw: pw),
             bb = window(b: b, rmin: 2, rmax: h, cmin: 1, cmax: w)
        else d = [!d1, !d2],
             d1 = image(b: b1, pw: pw), d2 = image(b: b2, pw: pw),
             b1 = window(b: b, rmin: 1, rmax: h, cmin: 1, cmax: pw),
             b2 = window(b: b, rmin: 1, rmax: h, cmin: pw+1, cmax: w)
        end_if
 end_cases
end text_block
```

Fig. 2 Text_block module

nondeterministic. We expect most guarded commands to have disjoint guards.

Assertions can be used to introduce names for subcomponents. For example the guard for the *above* operation introduces *w1, h1, t1, w2, h2* and *t2* as names for the components of *top* and *bot*. The types of such local names are determined by context. In the *image* operation, "line" is a local name for the first line in *d*, while *dd* is a local name for the sequence containing all but the first line of *d*. Note how the "!" prefix affects the implied type of local variables.

Assertions have truth values and do not have side effects, so that the order in which they are written is immaterial. The results of sending a message to an actor can be used in an assertion, such as *window* in the example, provided that the request does not cause any state changes (the action in question is not part of a longer action sequence and has a **reply** but no **send** or **update** clauses). The meaning of a set of assertions is the conjunction of its elements.

We believe conditional assertions can make complicated conditions easier to understand. Conditional assertions are defined by the following equivalence:

if P then Q else R end_if = (P and Q) or (not P and R).

An example of a conditional assertion can be found in the definition of the *image* operation of text_ block.

3.3. Modeling States

For theoretical purposes, the state of an actor can be taken to be the equivalence class of message sequences that will drive the actor into the given state, while for purposes of analysis and design, conceptual representations are more convenient[10]. We have found it useful to distinguish between *data states* and *control states*.

A data state represents the long term memory of an actor, and has a conceptual representation of the kind described above. Data states are useful for describing mutable data types and state machines that perform single atomic actions.

A control state represents the short term memory of an actor, and can be modeled as a finite state automaton. The correspondence between the control structures of MSG and regular expressions is shown in Fig. 3, while the correspondence between regular expressions and finite state automata is well known[32]. Control states are useful for describing state machines that perform transactions involving sequences of actions, such as interactive terminal sessions or communications protocols. We have provided a special mechanism for describing control states because we believe encoding control states as data states complicates a specification.

Control states are like R-nets in SREM[3], except that irrelevant details are suppressed. At the functional specification stage, they should be used to define control sequences only to the level of detail that is visible from outside the system: ideally, each state transition of the model should correspond to the receipt of a message and the sending of the associated responses, where all of the messages mentioned cross the system boundary. We have allowed the **receive** clause of a guard to remain optional in MSG despite this ideal, because it is

a **and_then** b	(a ; b)
cases a **or** ... **or** z **end_cases**	(a | ... | z)
repeat a **or** ... **or** z **end_repeat**	(a | ... | z)*

Fig. 3 Regular Expressions for MSG Control States

sometimes convenient to factor out common parts of a set of guards in a nested **cases** statement (cf. the undo operation in the editor example).

3.4. Functional Specification Example

When using MSG to record a functional specification, each of the software subsystems to be built and each of the external systems with which they interact are modeled as actors. In this section we give a functional specification for a simplified text editor, which interacts with two external systems, the user and the file system. A partial functional specification for the text editor is shown in Fig. 4. This example illustrates the use of auxiliary definitions and of conceptual representations for states.

Auxiliary definitions are used to introduce symbolic names for abstractions useful in modeling the system at hand. In this example, we have used definitions to introduce a symbolic name for the type line_number, and for *last*, a derived component of the state representing the line number of the last line in the buffer. Definitions can also be used to introduce symbolic constants, predicates, and functions. Concepts needed to *describe* properties of the system should be introduced as definitions, while entities used to *represent* components of the system should be introduced as modules defining actors. Definitions can be imported from other modules, to avoid duplication and potential inconsistency.

The data state of the editor is modeled as a tuple with several named components. The components of a data state are supposed to be logically independent, although some constraints may be imposed by the data invariant (eg. the current line *here* must be within the bounds of the current buffer). It is sometimes convenient to define derived components of the state, such as *last*. Derived components are analogous to derived v-functions in SPECIAL and to computed views in databases. The values of derived components are completely determined by the primary components of the data state, and cannot have their new values constrained by **update** assertions.

The control states of a state machine are defined implicitly by the primitive actions in the **behavior** section. The alternatives of a guarded command are separated by the keyword **or** while the components of an action sequence are separated by the keyword **and_then.** These keywords act like the alternation and concatenation operations in regular expressions. A **repeating** guarded command terminates when an action containing the keyword **exit** occurs.

The editor has a transient initial state, where it is waiting for a file name, followed by a recurring state where it obtains and executes commands, until it receives a *quit* command. State transitions are described by sets of assertions following the keyword **update,** in which context unadorned variables denote their values in the old state, while variables prefixed with a "*" denote their values in the new state.

Variants in message layouts are implicit. The **receive** clause is part of the guard, which is **true** if and only if the message that was last received has a name and a layout matching the names and types specified by the **receive**, and the contents of the message satisfy the constraints imposed by the **where** clause, if there is one. Variants in incoming messages can be used to model exception handling, optional arguments, and operator overloading. We follow the convention that an anonymous reply message corresponds to a normal termination condition, while a named reply corresponds to an exception. There are no restrictions on the number of components a reply may contain, for any termination condition.

The editor example also illustrates the kind of decisions that must be made during the functional specification stage. Two of the requirements for this editor were facilities for inserting text and for reversing the effects of unintended commands. The first was provided by the *insert* command, which inserts a line of text after the current line. To demonstrate that the requirement was met, it is necessary to show that text can be inserted anywhere in the file. Given that there will be another command to set the current line, this is the case, because there is a line position before the first line of text that enables insertions at the beginning, as well as after each line in the buffer. Even though the requirement can be met, it is still necessary to check that the commands for doing so are acceptable to the user: moving to insert text before the current line may be too cumbersome, or a

```
machine editor
define line_number = integer                               % local definition
define last = length(buffer)                               % last line
import document, textline, file_name                       % nonlocal defs
  from filesystem
model {buffer: document,                                   % working copy
        here: line_number,                                 % current line
        current_file: file_name,
        history: sequence(document)}                       % history of buffer
  where 0 <= here, here <= last
behavior
  receive edit{f: file_name}
  update *buffer = filesystem.contents(name: f),
           *here = 0, *current_file = f, *history = []
and_then
  repeat
    send prompt{s: string} to user where s = "command: "
  and_then
    cases
      receive command{op: string, line: textline}
      where op = "insert"                                  % after current line
      update *buffer = [!prefix, line, !suffix],
             prefix = buffer[1 .. here],
             suffix = buffer[here+1 .. last],
             *here = here + 1,                             % new line current
             *history = [buffer, !history]
    or
      receive command{op: string} where op = "undo"
      cases
        history = [b, !h]
        update *buffer = b, *history = h
      or
        history = []
        send undo_failure{s: string} to user
        where s = "nothing to undo?!"
      end_cases
    or
      receive command{op: string} where op = "quit"
      send update{name: file_name, contents: document} to filesystem
      where name = current_file, contents = buffer
      send response{s: string} to user where s = [!current_file, "updated"]
      exit                                                 % from repeat
    or
      ...                                                  % other commands
    end_cases
  end_repeat
end editor
```

Fig. 4 A Simple Text Editor

fictitious line before the first may be too confusing. Similarly the second requirement is met by the *undo* command, and it is necessary to check whether it is acceptable for the undo command itself to be irreversible, and for text changes to be reversible while line motion is not.

As can be seen, many of the most difficult decisions involving design tradeoffs and interpretations of requirements are made in the functional specification stage. We feel that precise and formal functional specifications are essential at this stage, to force refinement of all vague concepts in the requirements. Otherwise, some of these refinements will not be done or reviewed until the coding phase, when revisions are much more expensive. While the effort of producing a formal functional specification may not be negligible compared to coding, it should still be appreciably smaller because many details are left out. For example, the specification of the *undo* command is simple to express given our conceptual representation of the editor state, whereas an implementation is likely to be orders of magnitude more complex, due to efficiency considerations: in the implementation, it will be necessary to save differential state information, rather than a snapshot of the entire buffer after each command.

A model of the user of the editor is shown in Fig. 5. This model is used to document assumptions about how the user initiates and terminates an editing session. A more thorough discussion of the importance of modeling the environment of the proposed system can be found in [8].

A model of the filesystem is shown in Fig. 6. Models of the external systems that will interact with the proposed system are important because they provide a place to document assumptions about external interfaces. It is often useful to simplify and abstract from the real interface to the external system, because this contributes to portability and the concurrent design of systems. For example, the *update* operation of the filesystem will either create a new file or modify an existing one, as appropriate, masking a distinction that is maintained by many real file systems. If the proposed system relies only on the virtual interface defined in the functional specification, then in order to port the system to a new environment, all that must be done is to re-implement routines with those interfaces in terms of the primitives provided by the new environment. The same applies to concurrent development, where software is designed before the details of the operating environment and the hardware configuration are fixed.

```
machine user
import file_name from filesystem
behavior
 send edit{f: file_name} to editor                          % invoke editor
and_then
 repeat
   receive prompt{s: string}
   send command to editor
 or
   receive response{s: string}
   where s ~= [!f, "updated"]
 or
   receive response{s: string}
   where s = [!f, "updated"]
   exit                                                     % done editing
 end_repeat
end user
```

Fig. 5 User of the Editor

```
machine filesystem
define file_name = string
define document = sequence(textline)
define textline = string
model map(from: file_name, to: document)
behavior
  cases
    receive contents{name: file_name}
    reply {text: document}
    where if filesystem = {name: c, !rest}
            then text = c else text = [] end_if
  or
    receive update{name: file_name, contents: document}
    update if filesystem = {name: c, !rest}
             then *filesystem = {name: contents, !rest}    % file exists already
             else *filesystem = {name: contents, !filesystem}    % new file
             end_if
  or
    ...   % other filesystem operations
  end_cases
end filesystem
```

Fig. 6 Filesystem Model for the Editor

3.5. Use of Decomposition

Abstraction rather than decomposition is the primary means for controlling complexity in MSG. We believe that decomposition is important for the graphical methods of functional specification because it provides a way to abstract from details. Decomposition is an imperfect vehicle for supporting abstractions, because composite objects cannot be understood independently of their parts. Decomposition is better suited for defining the structure of a particular mechanism for implementing a system, as is needed in a feasibility study, than it is for precisely defining a user interface, as is needed in a functional specification.

In MSG, each actor is viewed as a black box, so that the meaning of the actor is described by giving assertions about the inputs and outputs, rather than by specifying what is inside the box. There is no hierarchy of modules in the usual sense. Since calls on other actors are allowed in assertions, we can still have some actors defined in terms of other actors, but the lower level actors are used to *recognize* the correct outputs of the higher level actor, rather than to *construct* those outputs, as in a conventional decomposition.

The system to be built can be described as a collection of actors rather than as a single actor, where all of the actors are on the same level. This is useful for partitioning large systems into independent functions, and also for allowing the system behavior to be described at the most natural level of abstraction. In such a case, there will be an actor describing the (each) central function performed by the system, and a pair of actors describing the transformations from concrete inputs to abstract inputs and from abstract outputs to concrete outputs. The editor example above shows only the central machine, without giving the mapping from the strings typed by the user to abstract editor commands. Formats and responses to ill formed inputs must be included in a complete functional specification.

4. Diagrams and Walkthroughs

The diagrams associated with MSG were chosen to meet the following constraints - they must be consistent with and support the underlying computational model, be usable on video display terminals (VDT), and be mechanically derivable from the MSG specification. There are several different kinds of information in an MSG specification, and we provide a separate type of diagram summarizing each kind of information. The fundamental diagram provides an overview of the system of actors, while the action diagrams provide descriptions of individual actors. All of the diagrams are directed graphs, where the types of the components and interactions are indicated by different kinds of nodes and arcs.

The fundamental diagram shows the actors representing the system and its environment and the message paths connecting them, and is an extension of the DeMarco data flow diagram. Figure 7 shows the fundamental diagram for the editor example. Transform actors are shown as circles, machine actors as rectangles, type actors as rectangles with a double top, and message links as solid lines with arrowheads in the direction of information flow. There is one node for each actor, and one arc for each message sent from one actor to another. Labels for the nodes and arcs are derived from comments in the MSG text.

There are only three nodes in the diagram for this example, while a more realistic one would have five, with transforms mapping concrete inputs into abstract ones, and abstract outputs into concrete ones. We believe the central subsystems of a functional specification should be defined in terms of the most natural conceptual representation for the data, and that the input/output formats for the data should be defined by separate transforms. This has the merit of simplifying the specifications and of decoupling decisions that must be made early (central functions) from those that are likely to change (formats). If this strategy is followed, the fundamental diagram will have a small number of components for all but the most complex systems: roughly equal to the number of subsystems, external systems, input types, and output types.

For very large systems, the fundamental diagram may be too complicated for use in reviews and walkthroughs. We provide a mechanism to derive more suitable diagrams, in which groups of actors are represented by subsystem nodes and are connected by message trunks. Subsystems are groups of transforms, machines, and types which are shown as squares with inscribed circles. Trunks are groups of message lines shown as double solid arrows. Note that these are display abstractions, not behavior abstractions, and do not imply an intention by the designer to regard the grouping as significant. We envision an interactive display with the ability to expand a trunk into its component messages on request, which initially shows a trunk for each pair of actors that exchange more than one kind of message. Figure 7 shows the various filesystem messages grouped into a trunk for easy viewing.

The action diagrams are provided to display the descriptions of individual actors. These diagrams show a summary of the externally observable behavior of an actor, in a format similar to a traditional flowchart. The temptation to include details that are not externally observable can be countered by a guideline requiring every primitive action to either receive or transmit a message. Since the behavior of a type or machine can contain a very large number of cases, the diagrams are layered, where lower level diagrams show explosions of composite nodes in higher level diagrams. The structure of the MSG description is shown in terms of sequences, cases, repeats and primitive actions. An action which does not update the state of an actor is represented by a circle, while one that does update the state is represented by a rectangle. Causal chains are depicted by a dashed line connecting action, case and repeat symbols. Cases and repeats are shown as squares with inscribed circles, cases being differentiated from repeats by entry labels on the symbols (+ for cases, * for repeats). The explosion of the editor actor is shown in Figure 8, while a subsequent explosion of a repeat is shown in Figure 9.

We envision a work style where diagrams are used as an informal notation in the early stages of analysis, to support exploration and refinement of tentative ideas, supported by a graphics workstation. When the structure of the specification has stabilized, MSG text capturing the information in the diagrams is automatically generated from the graphics and serves as a template guiding further refinements, which are done based on the text form of the specification. Final reviews and walkthroughs are assisted by diagrams automatically derived from the final MSG specification.

5. Computer Aids

Formal specifications are cumbersome to use without automated support. The most basic facilities needed are syntax and type checks, cross references, pretty printing, and automatic generation of diagrams. A database system capable of accepting MSG modules and providing access, concurrency, and configuration control is important for projects of any size. Facilities for graphical input of diagrams and generation of MSG skeletons from the diagrams are useful, as are facilities to include imported definitions automatically, because the amount of data entry and page flipping is reduced.

More sophisticated aids are also made possible by the precise semantics of MSG. A tool for locating overlapping guards in **cases** or **repeat** statements would be useful for locating areas of potential nondeterminism. While such areas are not necessarily errors, they should be reviewed to make sure the nondeterminism was intended, rather than the result of ambiguous requirements. A tool for checking whether the data invariants of a data type or state machine are preserved by its primitive actions would also increase the confidence that a specification is well formed. Tools for simulating the behavior of a system from its functional specification would be useful for checking whether the specification captures the customer's intent. Test cases automatically derived from the functional specification would ease the burdens of implementation and quality assurance. Another intriguing possibility is the automatic translation of assertions from the abstract data models used in the functional specification to the implementation oriented data models of an architectural design, given a description of the correspondence between the data models. We are currently investigating tools of these types.

6. Conclusions

We have found that a precisely defined specification language such as MSG is very important for functional specifications, because the language determines what can be said. Our experience with an early version of MSG in a software engineering course indicates that the formal and precise nature of the notation is an aid rather than a barrier to human designers, once the underlying concepts are mastered. A mathematical semantics for the language is essential in order to support computer aided design at the early stages of development.

6.1. Contributions

We have developed a clear and unambiguous model for the meaning of a functional specification in MSG. This model supports concurrency which in turn allows parallel implementations. In contrast to the DREAM[33] modeling system, states are local to actors and the only interactions between actors are by message passing. This allows responses to messages to be modeled as atomic actions, eliminating complications due to synchronization, and making concurrent descriptions simple and natural. In contrast to SREM[3], we provide support for abstract conceptual models for states, eliminating encodings that expose complications due to implementation issues. The data flow and state machine paradigms have been combined in a single coherent framework. Data states have been distinguished from control states, and a convenient notation has been provided for defining each of the two. A graphical notation for providing summary information has been provided, such that the graphics can be related to the semantics of MSG, and can be mechanically generated from the text form of the MSG specification.

6.2. Evaluation of MSG

In this section we evaluate MSG with respect to the requirements for functional specifications set forth in Section 1.2.

MSG has a precise semantics, and a formal definition is in preparation. It is based on a coherent model of computation, namely the actor model. The examples in this paper and others we have developed for a software engineering course lead us to believe the notation is sufficiently powerful to (conveniently) express all functional aspects of system behavior. Performance considerations, such as real time constraints, have not yet been incorporated. The notation provides a useful set of built-in abstractions, and facilities for adding more. The notation is not redundant, except for the possibility of overlapping guards in a *cases* or *receive* statement.

The notation has been expressly designed to allow message types to be defined independently of assertions describing finer details, and to allow the existence of modules or interactions to be recorded even if details are not yet available, so that partial descriptions are supported. Modular descriptions are supported, and components in a module description have a fixed order, to make it easier to find things. Concepts defined in other modules must be explicitly imported, providing cross references and disambiguating nonlocal names. Our experience with students indicates that programmers can learn to read the notation in a few weeks, and can learn to write it in ten weeks. We have had limited experience with modifying MSG specifications but we believe the structure of the language helps to localize design decisions in the text. We have used MSG in functional specifications and in architectural design, and we are currently working out the sublanguage intended for module design. MSG can also be used in the subset of requirements definition involving defining interfaces to external systems. We believe that the notation will support mechanical processing, and we are currently working on implementing a number of the tools mentioned in the previous section.

6.3. Future Work

More work is needed to develop and evaluate the effectiveness of tools that go beyond the bookkeeping aspects of design. We intend to use MSG to support a number of meaningful consistency checks, computer aided refinement of designs, and the automated generation of test cases from the specifications.

The aspects of MSG addressing architectural and detailed design need to be refined, and will be the subject of a future paper. It is currently not clear whether MSG will need to support the description of actors that create other actors, or if the current type mechanism is sufficient to support descriptions of applications like a mail system where new mailboxes and new network nodes can be created as the system runs.

We believe that the framework developed here is promising for the specification of real time systems, but the details remain to be worked out. In addition to data driven processes, real time systems contain periodic processes, which are to be initiated at fixed time intervals. Our model must be extended to allow predicates involving time delays in assertions, so that actors behaving like clocks can be defined. Experience will show if such a modeling scheme is adequate.

Our assumption of reliable message transmission is necessary for a guaranteed upper bound on the response time. Our model makes no constraints on the amount of delay in a message transmission, and thus is compatible with any reasonable set of timing standards. We believe that as much of the functionality of a system as possible should be expressed independently of time, because that leads to a simpler description, and helps to expose opportunities for parallel implementation.

MSG should be exercised in an application such as data processing where the functionality of the system is dominated by its database component, to verify our conjecture that the data modeling primitives we have introduced are also applicable to the conceptual modeling part of database design.

Fig, 7 Fundamental Diagram of the Editor

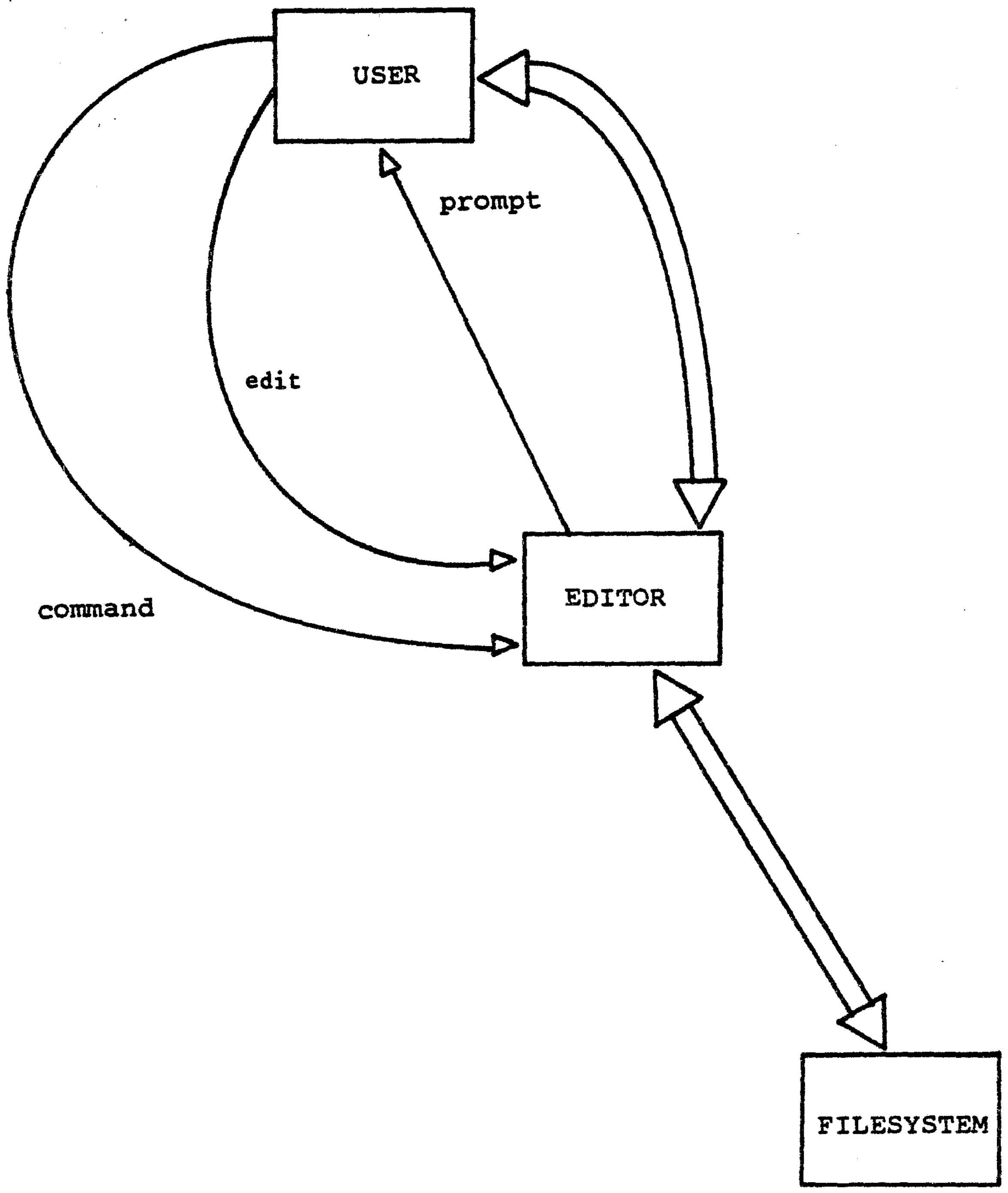

USER
prompt
edit
EDITOR
command
FILESYSTEM

Fig. 8 Editor Machine Description Diagram

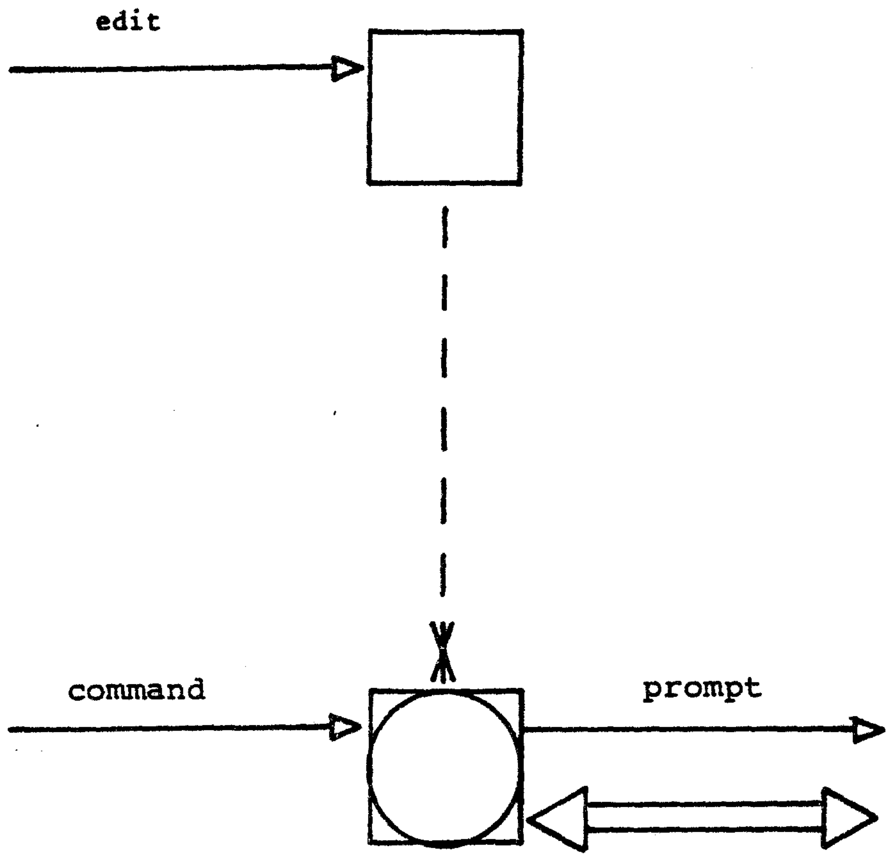
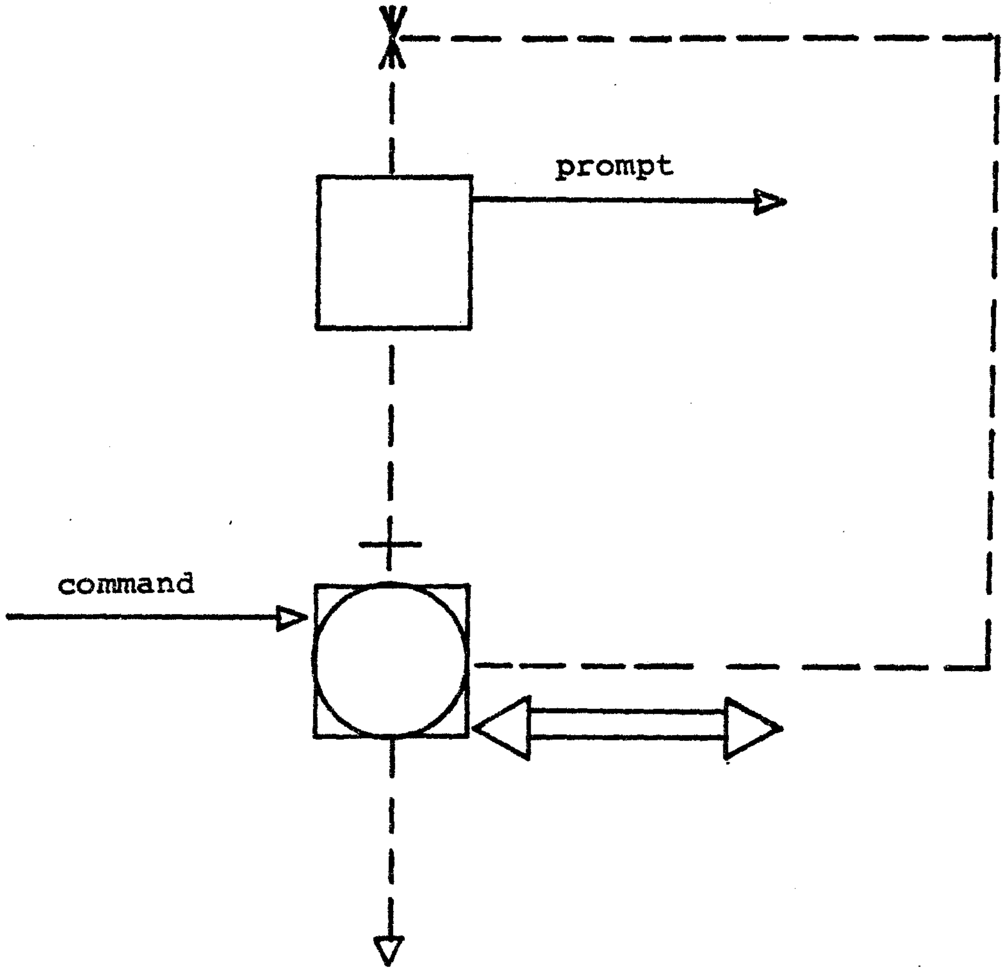

edit
command
prompt

Fig. 9 Repeat Action Explosion Diagram


prompt
command

BIBLIOGRAPHY

[1] P. G. Neumann, ed., "Special Issue on Rapid Prototyping", SIGSOFT, Dec 82.

[2] D. Teichrow and E. A. Hershey III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", *IEEE Transactions on Software Engineering, vol. SE-3,* no. 1, 41-48, Jan 1977.

[3] M. W. Alford, "A Requirements Engineering Methodology for Real-Time Processing Requirements", *IEEE Transactions on Software Engineering, vol. SE-3,* no. 1, 60-68, jan 1977.

[4] T. DeMarco, *Structured Analysis and System Specification,* Yourdon Press, 1978.

[5] D. T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Transactions on Software Engineering, vol. SE-3,* no. 1, (jan 1977), 16-33.

[6] K. T. Orr, *Structures System Development,* Yourdon Press, New York 1977.

[7] L. Robinson, *The HDM Handbook,* SRI International, Menlo Park, CA, 1979.

[8] P. Zave, "An Operational Approach to Requirements Specifications for Embedded Systems", *IEEE Transactions on Software Engineering, Vol. SE-8,* No. 3, 250-269, 1982.

[9] C. E. Hewitt and H. Baker, "Actors and Continuous Functionals", *Formal Description of Programming Concepts,* North-Holland, New York, 1978, 367-387.

[10] A. Yonezawa, *Specification and Verification Techniques for Parallel Programs based on Message Passing Semantics,* Ph. D. Thesis, MIT, 1977.

[11] D. H. H. Ingalls, "The Smalltalk-76 Programming System: Design and Implementation", *Proc. Principles of Programming Languages Symposium,* 1976.

[12] A. Goldberg and D. Robinson, *Smalltalk-80: The Language and its Implementation,* Addison Wesley, 1983.

[13] D. L. Weinreb and D. Moon, *Lisp Machine Manual,* MIT Artificial Intelligence Laboratory, Cambridge, MA, 1981.

[14] B. Liskov, R. Atkinson, T. Bloom, E. Moss, J. Schaffert, R. Scheifler, *CLU Reference Manual,* Springer-Verlag, New York, 1981

[15] M. Shaw, *Alphard: Form and Content,* Springer-Verlag, New York, 1981.

[16] *Ada Programming Language,* DoD, ANSI/MIL-STD-1815A, 1983.

[17] B. H. Liskov, A. Snyder, R. Atkinson, J. C. Schaffert, "Abstraction Mechanisms in CLU", *CACM Vol. 20, No. 8,* 564-576, Aug. 1977.

[18] J. V. Guttag, E. Horowitz, and D. R. Musser, "Abstract Data Types and Software Validation", *CACM,* v. 21, n. 12, 1048-1064, Dec. 1978.

[19] J. A. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright, "Abstract Data Types as Initial Algebras and the Correctness of Data Representations", *Proc. Conf. on Computer Graphics, Pattern Recognition, and Data Structures,* 89-93, 1975.

[20] J. A. Goguen, J. W. Thatcher, E. G. Wright, "An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types", *Current Trends in Programming Methodology Vol. 4,* Prentice Hall, Englewood Cliffs, NJ, 1978.

[21] R. Nakajima, M. Honda, and H. Nakahara, "Describing and Verifying Programs with Abstract Data Types", *Formal Description of Programming Concepts,* North-Holland, New York, 1978, 527-556.

[22] M. E. Majster, "Limits of the 'Algebraic' Specification of Abstract Data Types", *SIGPLAN Notices Vol. 12, No. 10,* 37-42, 1977.

[23] C. A. R. Hoare, "Proof of Correctness of Data Representations", *Acta Informatica,* v. 1, n. 4, 271-281, 1972.

[24] W. A. Wulf, R. L. London, and M. Shaw, "An Introduction to the Construction and Verification of Alphard Programs", *IEEE TSE Vol. SE-2, No. 4,* 253-265, Dec. 1976.

[25] V. Berzins, *Abstract Model Specifications for Data Abstractions,* Ph. D. thesis, MIT, 1979.

[26] D. Scott, "Data Types as Lattices", *SIAM Journal of Computing Vol. 5, No. 3,* 522-587, Sep. 1976.

[27] G. D. Plotkin, "A Powerdomain Construction", *SIAM Journal of Computing Vol. 5, No. 3,* 452-487, Sep. 1976.

[28] M. Shaw, E. Borison, M. Horowitz, T. Lane, D. Nichols, and R. Pausch, "Descartes: A Programming-Language Approach to Interactive Display Interfaces", *Proc. SIGPLAN '83 Symposium on Programming Languages Issues in Software Systems,* 100-111.

[29] E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, NJ, 1976.

[30] D. L. Parnas, "A Generalized Control Structure and Its Formal Definition", *CACM Vol. 26, No. 8,* 572-581, Aug. 1983.

[31] C. A. R. Hoare, "Communicating Sequential Processes", *CACM Vol. 21, No. 8,* 666-677, Aug. 1978.

[32] J. E. Hopcroft and J. D. Ullman, *Formal Languages and Their Relation to Automata,* Addison-Wesley, Reading MA, 1969.

[33] W. E. Riddle, J. C. Wileden, J. H. Sayler, A. R. Segal, and A. M. Stavely, "Behavior Modeling During Software Design", *IEEE Transactions on Software Engineering, Vol. SE-4,* No. 4, 282-292, 1978.

MEIS

MICROELECTRONIC & INFORMATION SCIENCES CENTER

INSTITUTE OF TECHNOLOGY
UNIVERSITY OF MINNESOTA

227 Lind Hall / 207 Church Street S.E.
Minneapolis, Minnesota 55455
612/376-9122

AN APPROACH TO CUSTOM DESIGN OF VLSI CIRCUIT DESIGN AUTOMATION

Microelectronic and Information Sciences Center

Technical Report #10

E. Bruce Lee
Department of Electrical Engineering
University of Minnesota

Choonkyung Kim
Department of Electrical Engineering
University of Minnesota

Marek Perkowski
Institute of Automatics
Warsaw Technical University

AN APPROACH TO CUSTOM DESIGN OF VLSI CIRCUIT DESIGN AUTOMATION

MEIS Technical Report #10

E. Bruce Lee
Choonkyung Kim
Marek Perkowski

The research reported in this paper was supported in part by the Microelectronic & Information Sciences Center and the Honeywell Solid State Electronics Division.

CONTENTS

1. Introduction
   - 1.1. Tasks of the report
   - 1.2. Custom design circuits
   - 1.3. Basic assumptions of the approach

2. Design automation system
   - 2.1. Requirements for systems of automatic design of VLSI digital systems
   - 2.2. Place of hardware desciption languages in the design automation systems
   - 2.3. General assumptions and characteristics of the design automation system DIADES
     - 2.3.1. Man-machine interaction
     - 2.3.2. Description of the requirements for the device The role of transformations
     - 2.3.3. Hierarchical and iterative design
     - 2.3.4. Application of Artificial Intelligence methods in the search of quasioptimal design solutions
     - 2.3.5. Simulation
     - 2.3.6. Documentation of the design data and process
     - 2.3.7. Openess and flexibility of the system
     - 2.3.8. Languages of the system
   - 2.4. Process of design
   - 2.5. Structure of DIADES

3. Source language to describe hardware
   - 3.1. General information about language ADL
   - 3.2. General structure of ADL
   - 3.3. Declarations
   - 3.4. Basic statements
   - 3.5. Description of parallel actions
   - 3.6. Description of automata and logic functions
   - 3.7. Descripton of structure of system.

4. Abstract implementation of operational unit
   - 4.1. Abstract models of digital blocks
   - 4.2. Algorithm of abstract implementation of operational unit
   - 4.3. Factorization of common substructures of abstract structure
   - 4.4. Examples of abstract implementation

5. Simulation
   - 5.1. Simulation control language INADL
     - 5.1.1. Syntax
     - 5.1.2. Semantics
   - 5.2. The SIMADL simulator

6. Conclusion

The authors are grateful to the graduate students of Warsaw Technical University and of the University of Minnesota who have put much effort in writing, testing and maintaining of many programs of DIADES system.

These students include Jan Baranowski, Jerzy Mityk, Jerzy Piotrowski, Glenn Wadden, Antonio Pascoal, James McGee, Richard Myers, Marek Urbainski, Michael Shelton, Audrzej Cieslak, Dave Van DeVaa, Branislav Vajdic, Greg Larson, Yeung Huh, Steve Gunderson, Aloke Guha, Chengchih Chu, Jacek Wojutynski, Pawet Dynko, Tomasz Strawinski and Robert Irani.

1. INTRODUCTION

1.1. Tasks of the report.

1. The very quick development of semiconductor technology is a characteristic of the last ten years. One of the most commonly met technologies is the MOS (NMOS, PMOS, CMOS) technology, used in LSI and VLSI circuits of calculators, microprocessors, wrist-watches, memories, games, military, telecommunication and industry circuits. Some modern design methodologies for such circuits are currently under investigation both in industry and at the universities [Dire81], [Newt81], [Alle81], [Hewl81].

   However, to our knowledge none of the modern methodologies is widely accepted and practically used for circuit design.

   The problems of design of VLSI MOS circuits are complex and multiaspect. A number of the design criteria should be formulated in a way different from what was done in the classical synthesis of logic circuits and their design methodology - a lot of new research problems arise. They result both from increasing the level of design as well as from the very specific properties of the technology, which to a higher extent than the previous ones combine the stages of system, architectural, block, logic, and technological design. At the same time some well known and already forgotten problems like for instance minimization and state assignment of automata or Boolean function minimization once more appear in somewhat different form.

2. Because of the large complexity of VLSI MOS circuits it is a current view (Hartenstein and others) that in the design of these circuits occur or will quickly occur the "hardware gap" similar to the existing already "software gap". One of the main possibilities of decreasing this gap is to look among the systematic design methodologies which make use of integrated design automation systems, whose input is a description of the circuit in a high level hardware and design description language. Most of the currently used systems are of the interactive type like CALMA or COMPUTER-VISION which lack a higher level language description and processing capabilities.

3. There are different approches for CAD of VLSI circuits and different methodologies are proposed. What will be proposed in this and the following reports is an approach which attempts to construct systematic design procedures integrated in a complex design automation system. We will treat some aspects which to our knowledge were not previously discussed.

   The report presents some concepts and ideas which underlie a Minnesota version of DIADES system (DIgital Automata DESigner) and also gives a detailed description of part of the programs currently implemented or under implementaion.

1.2. Custom design circuits

1. The amount of difficulty arising in VLSI design is so large that it can be hardly included into one unique design methodology. The problems can be seen from the management and market point of view, as well as from the technology, process orgainization, system architecture of different types of circuits design methodology points of view. The present approach details only some specific selected problems related to MOS VLSI design and its automatization, not attempting to fulfill all the requirements.

One of our goals was to take a complex view of the entire collection of problems of VLSI design coming from the conviction that automatic design is most useful when it joins successive design stages into one process and all design documentaion is stored in the computerized data base. Design using a complex design automation system gives special requirements to particular programs and methods of their interaction with the human designer.

To have the real possibility of completing the work we have to limit the class of designed circuits and their complexity which leads us to concentrate on custom design circuits. Currently many companies design such circuits for the demands of clients in the series from 1000 to 100,000 chips, and the most important problem encountered here is the quick turnaround rather than optimality of the results. Because of the big predictions about the number of types of custom design circuits in the near future as well as clients demands and limited personnel possibilities one can assume that a big market will soon exist for the software packages for complete and quick design of such circuits.

The custom design circuits structures strongly depend on different requirements and conditions under which they work: plenty of digital structures, signal types, codes and design solutions are applied here. The designer must consider a number of criteria such as chip cost, semiconductor area, number of pins, standard packages, speed, and power requirements. On the other hand, because of short series, the quality of circuit optimization is less important than in the for instance microprocessor or calculator chips produced in very large numbers. Because completely automatic optimization is very hard or even impossible, these circuits are optimized to a large extent by humans which prolongs substantially the design process. Contrary to custom design the most important factor is the low cost of design and speed of the entire process of transition from description of design requirements of a circuit to the set of mask composite pictures or pattern generator controlling tapes. Therefore complex, complete automation of such circuits is easier and in some sense more needed.

1.3. Basic assumptions of the approach

1. One of the goals is to create a design methodology which can be accepted by industrial engineers who are currently very reluctant to use the modern design tools and do not accept methods which they do not completely understand and cannot readily verify. We want to reach this goal on the analysis of different currently used design methodologies and both univeristy and industries approaches to CAD of VLSI.

2. The problem of VLSI can be approached with design based on homogeneous structures [Menn68], [Odno73], [Mead80]. However such structures seem to be useful only in selected types of circuits and are seldom used in custom design. They will not be treated in this report at all.

3. Another assumption applied is that the methodology proposed be easily adaptable to rapidly changing technologies. As we will see in the sequel some of the methods presented fulfill these criteria and can be applied to a wider class of circuits. A part of the methods apply abstract descriptions and is dependent on the changing technologies only through the replaceable cost functions. The other part transforms descriptons on so high a level it is not dependent on the technology and as such it also fulfills the assumption.

4. The next assumption is connected with the desire of investigating new, original design methods and algorithms, however these may be more risky from the point of view of practical efficiency (for instance then the algorithms of joint minimization and state assignment of automata).

5. Much attention is devoted to the problems of optimization (minimization) of circuits. This is achieved through system, logic and topologico-geometrical methods. This caused elaboration of new algorithms for Boolean minimization of different types of circuits, state assignment of automata, Manhattan layout minimization and other.

On the other hand in designing the system as a whole we consider the project of optimization on the highest design level as the most important. With respect to the requirements of efficiency of algorthims the system practically applies approximative (suboptimal) methods. However by its design attention was given to having the potential possibility of generating different equivalent variants of the circuit from the very highest levels of design. It is then not reasonable, in our opinion, to apply time consuming minimization to partial problems while the entire design problem is solved without even an attempt for minimization (for instance when the excitation functions of a counter are minimized which can be totally rejected from the circuit as a result of flow diagram optimizing algorithm which deletes redundant variables). Let us also point out that often it is difficult to define properly a specified cost function: we do not know if the function assumed really specifies our assumptions with respect to the evaluation of different variants of the designed circuit - it is then not reasonable to find solutions which are absolutely minimal with respect to such "artificial" method of variants evaluation. The authors are convinced that optimization of the circuit on the higher design levels lead to better results than only at the mask design level. On the other hand note that the design process must be iterated, because often we cannot express the idea of area minimization through minimization of some cost functions on the higher level of description. In MOS VLSI circuits it is sometimes, for instance, profitable to copy twice a certain counter with the goal of getting smaller amount of wires - the system design is then here strictly bound to the technological level.

6. In an MOS VLSI design automation system we often meet very complex combinatorial optimization problems. For solutions of the problems of this type we can take one of the following approaches:
   - simplify the problem and next look for its strictly optimal solution,
   - look for the quasioptimal solution to the nonsimplified problem.

Selection of one of the above approaches should always depend on the specifics of the problem.

One of our approaches was also to systematize the optimization methods applied in different design automation algorithms and making a common base for them. Different authors investigate mainly either approximative algorithms (mainly industrial) or strict algorithms which are very inefficient (mainly from universities). Our goal was to match (reconsile) the requirements of practice and the goals of theory. The heuristic method gives no full satisfaction (even to practically oriented system creators) because it never says how far is the solution generated from the real optimum. On the other hand (as it results both from the theoretical research in program complexity [Karp72] as well as from the common experience) only the approximative method is able to generate results for practical problems of higher dimensions in the interesting for us class of problems.

Authors look then for certain compromises: everywhere where it was possible to strictly formulate the optimization problems we apply the methods (and respectively - algorithms) of tree search - mainly based on the branch-and-bound principle. We will call these algorithms the tree algorithms. These algorithms are full, i.e., they have potential possibility of searching the complete space of solutions and are convergent (or minimal), i.e., they guarantee finding of the minimal solutions (i.e., solutions with minimal value of the cost function). This property results from their fulness, halting property and that they generate even better solutions. Application of branch-and-bound principle prevents from extensive complete search, which increases their efficiency with respect to the algorithms which search the complete solution space. However, their real efficiency results mainly from application of heuristics, and we practically remain nearly always with the quasiminimal solutions.

In our opinion the advantage of one method of automatic design over another one lies in equal extent in theoretically proven properties which limit the (always occuring) combinatorial search, as in respecting several heuristic, algorithmic and software aspects which are rarely discussed. These aspects can be presented by discussing algorithms and solution processes. The detailed discussion of programs is difficult so we will present only general algorithms.

7. For practical reasons (or sometimes also theoretical) in many problems the tree algorithms cannot be applied. In these problems, related mainly to the system or technological synthesis, the typically heuristic approach was applied, which utilizes different rules based on experience of modelling the behavior of a designer. We utilize here different programming techniques known from Artificial Intelligence as well as human-computer conversation, where the designer undertakes alternative decisions, selects design statements and procedures. For some of the problems discussed in the sequel specification of strict optimal algorithms is impossible for three reasons:
   - mathematical unsolvability of some problems (for instance in solvability of some problems arising by optimization of flow-diagrams arises from the unsolvability of Post word problem),
   - combinatorial exposion which occur in practically all problems and makes application of minimizing algorithms impossible even for some very simple problems for which such algorithms theoretically exist(*),
   - fuzziness of criteria and assumptions of synthesis, which are partially formulated and precized by the designer in the entire synthesis process itself.

8. The next assumption relates to the system organization. One tries to select such subroutines and data structures which can be used for many reasons. We have tried to organize the programs as a hierarchy of modules with possible wide scope of application at each of its levels.

(*) According to the Karp hypothesis [Karp72] and other research in program complexity the combinatorial explosion is in such problems unavoidable because polynomially convergent algorithms do not exist.

2. DESIGN AUTOMATION SYSTEMS

The traditional digital design procedures are inadequate for modern VLSI systems:

- they cannot be used on large systems,
- they do not consider standard blocks,
- they do not correspond to the typical design methods practically used by hardware VLSI designers,
- the description languages used in the procedures are too poor to describe the systems.

As a result of these drawbacks these methods can be applied only for synthesis of digital blocks such as: counters, adders, simple control units; while for the design of entire systems on a chip composed of such blocks the elaboration of new design methods is required.

2.1. Requirements for systems of automatic design of VLSI digital sytems.

We will now discuss the requirements for the systems of automatic design. The different authors and systems design teams formulate various requirements including tasks of synthesis, analysis, simulation, verification, and documentation. According to current knowledge no one of the systems fulfills all the requirements specified below:

S1. System should have many description levels corresponding to the different design levels (system, block, logical, circuit, geometrical levels) and the design process should be multi-stage. Synthesis consists in transition from general abstract descriptions to the more detailed descriptions and those more connected with particular technologies.

S2. System should have simulation programs, and shall ensure timing and concurency analysis on many levels of description. The simulation should give tools for detection of bottlenecks, time-logic inconsistences, hazards and races.

S3. For the simulation tasks the description of various input signals should be provided.

S4. System should provide tools for generation of diagnostic tests.

S5. Because of economical, technical, reliability and other design considerations the optimization of certain performance criteria such as total area of chip, area of certain layer, speed, power consumption and others is very important. The system should then provide the tools for automatic optimization of the selected criteria. It is also desired that it consider various cost functions, which can be easily interchanged for different technologies. The optimization procedures must be time and memory effective.

S6. The description of the device which is produced by the system on all the design stages, and especially the source description, should confirm the formal document describing uniquely and completely the system under design. This document is applied by system designers, logic designers, system programmers, customers and eventually also microcode programmers. The existence of the system serves then also to make cooperation easier among different groups of people involved in the entire design process.

S7. System should have tools for automatic, as well as human aided generation of different equivalent structures from standard subsystems, and for selection of the structures quasioptimal with respect to the certain criteria. This generation can be the result of the procedures transforming the descriptions. It can also result from selection of one from the several design procedures, devoted for the same aims but differing in quality of the results and the speed of obtaining them. Selection of the adequate procedure is related to the type and the complexity of the problem as well as the computational complexity of the algorithms applied (for instance selection of one of the two-level Boolean minimization algorithms can result from number of minterms in the function description).

S8. The "on-line" interactive human-computer dialog should be provided, at best with use of light-pen or graphic tablet: the designer selects menus, gives commands, questions, answers, uses graphic capabilities.

S9. Programs and languages of the system should take into account all technical and technological constraints such as the application of standard blocks, limited fan-in and fan-out, number and placement of pins, standard masks for buffers, design rules, etc.

S10. Together with the diagnostic capabilities of the programs which input the data (for instance error diagnostic of source language compiler) all programs of the system should provide checking of the uniqueness and consistency of the solutions on many levels of description as well as the extended error diagnostics. This is related to the possibility of multiple intervention of the human designer in the system run and of the different "semantic depth" of errors in input descriptions which causes that some errors are detected on further design stages.

S11. System should be flexible enough to permit changes and additions in the case of technology replacement - without destroying the system's structure as a whole.

S12. System should be modular, easily expandable, reliable, and easily portable with the possibility of cooperation with other programs and systems.

S13. The implementation language of the system should be a language with powerful algorithms creating capability. It should provide rich data structures, problem-solving mechanisms, associative memory, data base, graphic input/output, data structures and elementary functions capable for efficient implementation of combinatorial logic and geometrical algorithms.

S14. The source language should have tools for description of devices related to the discussed above properties of the entire design automation system.

S15. The system should include a language of easy modification for programs and data in the design process.

2.2. Place of hardware description languages in the design automation systems.

The following groups can be distinguished among the hardware description languages:
- register-transfer languages (sequential languages),
- structure-description languages (nonsequential languages),
- problem-oriented languages.

The first group includes such languages as: LOTIS, RTL, DDL, SFD-ALGOL, CDL, APDL, ALGORITM, AHPL; the second one: STRUKTURA or Stabler language, the third JARVS or ciclogrammes language.

The present authors are acquainted with about 40 hardware description languages, while according to some references more than a hundred languages of this type have been proposed. Characterizing however most of the languages known to us we can conclude that:
- description of the devices' behavior is detailed in the very first type,
- structure of the device is practically defined on the register-transfer level and not generated by the computer. This will not leave to the system the possibilities of generation, selection and optimization of the variants,
- the problems of devices optimization and creation of the essentially different variants preserving their equivalence have been not treated in the languages,
- there are not complex statements and parallel statements,
- none of the languages includes all the properties which we consider important to obtain the mentioned above assumptions,
- in many languages the external languages are nonreadable and the internal languages are not suitable to execute the equivalent tranformations on them.

On the bas of system requirements given in section 2.1 and the critics of current languages, as well as from our experiences with the previous versions of the DIADES system, the following requirements can be formulated with respect to the hardware description language being the source data to the integrated digital design automation system:

L1. There should exist the possibility of describing the device on many levels of description: algorithmic (behavior or system description), microprogram, functional, structural (logic and circuit), geometrical. The two first levels enable register-transfer description, the third-description of abstract automata. At each of the levels the possibility of the more or less detailed descriptions should be provided, as well as the exchanges among the behavioral and the structural descriptions.

L2. Notation of the source language should ensure sufficient amount of information for simulation of the device, i.e., must provide the tools to describe its behavior.

L3. The source language should provide sufficient information for the synthesis of the device. Sometimes this requires description of the device's structure, not only the behavior.

L4. The source language should be clear, readable, with general structure similar to the known programming languages, as simple as possible, making easy structural programming.

L5. Language should be exact, it should enable expressing each property of the device. It can however permit for the general description as well, and not to decribe the repeating or not important details. It should be similar to the informal descriptions, flowgraphs and notes used by the designers in the early design stages. It should be easy for programmer. It should not be far from the intuitive designers' concepts of the digital devices.

L6. The user of the system should not necessarily know the internal structure of the automation system.

L7. Language should enable description of arbitrarily complex parallel processes.

L8. Language should be easily expandable and possibly provided with the automatic syntactic extendability.

L9. Language should enable description of wide class of digital systems, not only computers but also asynchronous automata, iterative systems with dissipated control, etc.

L10. For the users' convenience the complex arithmetic, logical, bit, mixed and other statements should be provided.

L11. Language should include properties related to the possibilities of executing optimizing transformations, as well as the mechanisms for preserving description segments from such transformations (we will call this the fixing possibility).

L12. It should give tools for modular programming by the several programmers.

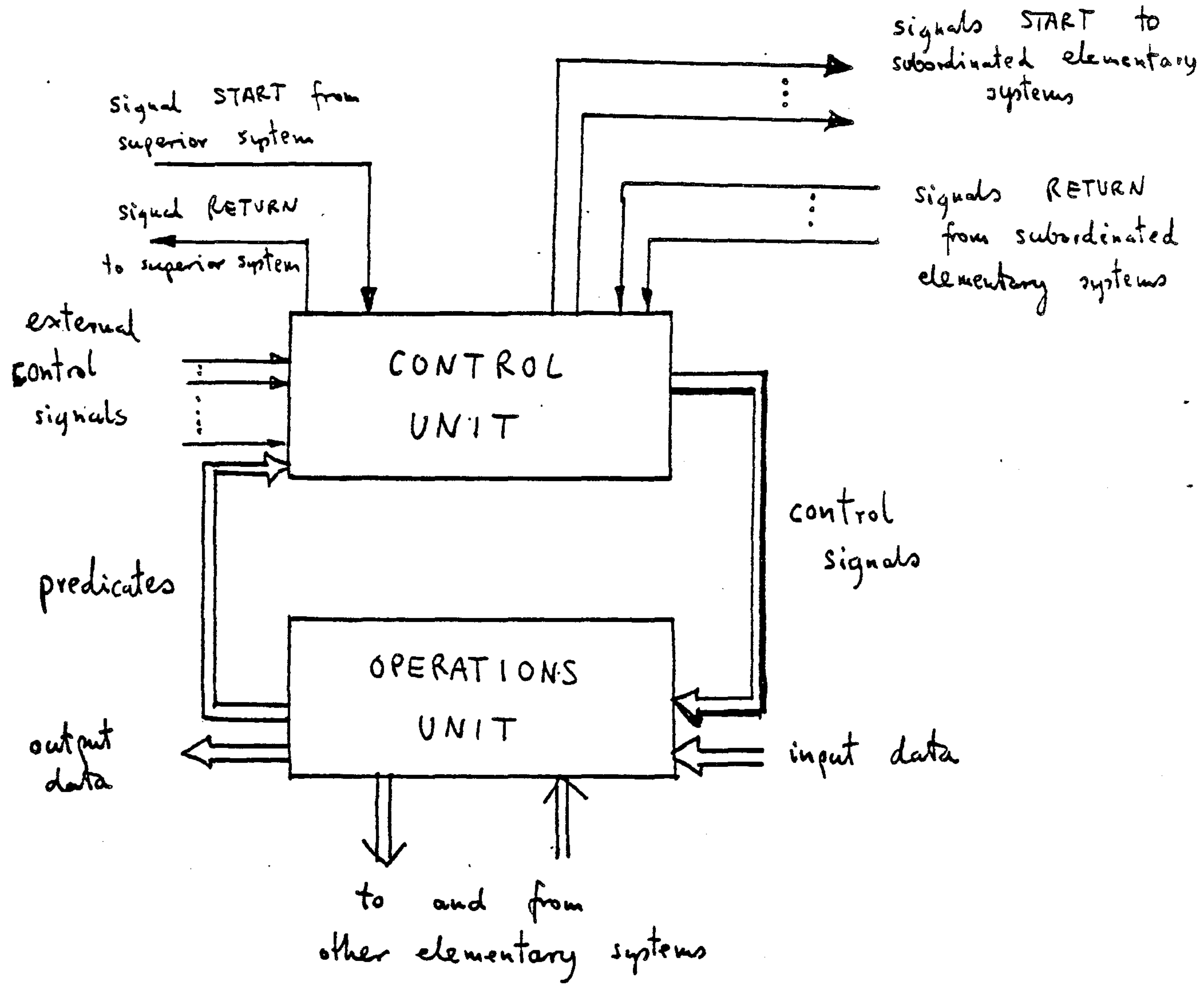

signals START to subordinated elementary systems
Signal START from superior system
Signal RETURN to superior system
signals RETURN from subordinated elementary systems
external control signals
CONTROL UNIT
control signals
predicates
OPERATIONS UNIT
output data
input data
to and from other elementary systems

Fig. 2.1

L13. The digital systems are composed from the system under design (SD) and some cooperating with it external system (ES). Both of these systems are transducers of digital data, i.e., D/A and A/D converters are included to ES. By the external system we will then understand some device which is not under design (it is specified as a "black box") and whose set of outputs gives the nonempty product with the set of SD inputs <u>or</u> whose set of inputs has the nonempty product with the set of SD outputs (usually both these conditions are satisfied). The external system <u>can be simulated</u>, but always the described signals must be interpreted as natural numbers or binary words.

L14. Both behavioral and structural description should be <u>hierarchical</u>, which is the requirement of both the synthesis and the simulation. The digital system is the hierarchical composition of the elementary systems. Each elementary system is the pair composed of CS and OU (see Fig. 2.1), the single CS or the single OU. Ways of interaction of these blocks will be described later on. Next, each CS and OU is the composition of the elementary units, executing the elementary operations. Blocks of the system should correspond to procedures and macros of language, which are called through the names and the parameters. This permits avoiding the detailed structure description and simplifies the behavior description.

L15. Language should provide <u>powerful error diagnostics</u>.

L16. Language should be <u>easily realizable</u>.

L17. Language should be <u>conversational</u>.

2.3. <u>General assumptions and characteristics of the design automation system DIADES</u>

By the elaboration of DIADES system it is attempted, to include, if possible, the above given requirements for the system, especially its source language. Before we present the specific languages and algorithms of DIADES, we will present some basic assumptions and we will characterize the proposed "philosophy" of the design, because they have meaningful importance for the system solutions applied further.

2.3.1. <u>Man-machine interaction</u>

Early stages of the design (by design we understand transition from requirements description to device description) are based mainly on experience, knowledge and problem-solving capabilities of the designer. He uses a certain set of mental rules of type:

<conditions, action, goals>

to obtain vertical and horizontal transformations of the design data. He defines, experiments and learns new, both heuristic and methodic: design axioms, design rules and design processes.

It is well known that the experienced designer is able to synthesize complex digital systems without using algorithmic methods of automata theory or only by their partial application, however by wide application of standard solutions, transformations, design tricks and "know how", application of informal decomposition of the problem and selective analysis of variants based on some heuristic rules, which he partially learns in the very process of design. By these activities he disposes a certain "mental language" of system description, in which he transforms data on all design levels, as well as sometimes the knowledge of algorithmic transformation methods.

He selects transformations in order to optimize the chosen criteria within the given constraints. While selecting, he considers the goals of the design, the characteristic features of the designed system and takes into account elements from a given catalogue of primitive devices. Man formulates plans for the operation of the device and next realizes them, transiting to the more detailed descriptions. By formulating plans he is however not able to predict all hardware and operational consequences of the decisions undertaken, he often maked mistakes in simple transforming actions, or in the coarse of the verification.

Based on an analysis of various designers' behavior it is concluded that:

1) The human shall be the essential element of the design system, undertaking the basic decisions and realizing the feedback. This requirement is connected with the difficulty of formulating adequate cost functions for MOS circuits, especially at the higher stages of design. A number of research publications on CAD confirm the opinion that the human is still not replaceable in actions requiring pattern recognition abilities and a goal-oriented behavior.
2) Abilities of the human and the computer are disjoint and mutually complementary.
3) The design automation process should be made to conform to the described above human design mode.
4) The system should include heuristic programs, modelling human design behavior.
5) Except for the heuristic procedures and tools for human-computer interaction the system should provide cooperation with standard design automation tools and algorithms, the requirements for which have been specified i section 2.1.

The human-computer interaction is indispensable especially in satisfy requirements S4, S7, S15. The best mode available is that of requirement S8.

2.3.2. Description of the requirements for the device. The role of transformations.

One of the possibilities of describing device requirement is application of the hardware description languages. Until now these languages were used for detailed and complete device descriptions. It can be however observed that some of these languages, used at the top level of abstraction can define the requirement rather than describe the device. The resulting methodology treats the design process as a sequence of heuristically selected vertical transformations (from abstract to more detailed descriptions) and horizontal transformations ("find other (better) solution on the same level on detailment of description"). These transformations should lead to the detailed logical and geometrical description of quasioptimal variants of the system under design. The higher the level of the description - the more readable are the description and the more substantial are the results of the transformations to create the differing final descriptions obtained after vertical transformations.

The horizontal transformations were introduced for fulfillment of requirement S7. These transformations should be selected in such a way that the variants generated are behaviorally equivalent, but they considerably differ respectively to other requirements, such as their speeds or semiconductor areas.

It results then that the meaning of the source language in DIADES is other than of such languages in other design automation systems, where the source program in detail specifies the structures of both CU and OU, leaving to the synthesis programs only the possibility of detailing, not changing, this structure. A program in ADL has an option of not specifying in fixed form the time relations or the system's structure, but only describing its behavior, i.e., logic and to some extent time relations among the input and the output variables.

2.3.3. Hierarchial and iterative design

Design in DIADES is executed on many levels of the design procedures hierarchy (satisfaction of requirement S1) and with possibility of iterative repeating of design stages (satisfaction of requirements S4 and S7). In the subsequent stages are created the device descriptions on different levels of abstraction. In certain stages the descriptions are decomposed into the partial descriptions. The design can be done independently for the decomposed segments. There exist at this point the possibility of returning to the previous description variants on the same level, as well returning to the higher levels of synthesis. The system generates sequences of different descriptions on the given abstraction level. The designer evaluates them; he has fascilities of deleting the descriptions which do not fulfill certain conditions, sending them to further stages of synthesis or analysis, introducing changes and modifications. He should aim at investigating many variants of the initial description and internal descriptions, together with additional rules, criteria an parameters, taking also into account the partial results from different levels of the previous design processes. For instance, as it results from the following text, as well as from the subject's references, for the synthesis of MOS LSI circuits the following is important: ensuring the possibility of different source descriptions, application of the mentioned above transformations, and comparison of many variants of implementation. The designer has often different ideas on how to describe the device on the highest abstraction level, however he is too lazy or has not enough time to make them more detailed and to optimize them, as well as to make their accurate comparison on one of the lower levels of the description. Such possibilities should be then supported by the system automatically, for instance by existance of different possibilities of some design stage execution (requirement S7). The selection of the method, done by the designer, should regard, together with other criteria resulting from the specific properties of the problem and the complexity of the data, also the restrictions of a practical nature such as time CPA, space of memory CM, input/output time, etc. The system should satisfy all of the above requirements.

2.3.4. Application of Artificial Intelligence methods in the search for quasioptimal design solutions

The heuristic methods, including the methods of Artificial Intelligence are currently increasingly applied in the various domains of computer aided design [Diet71], [Davi69], [Fri63], [Schi77], [Zakr75], [Stei73], [Dreu72], [Breu76], [Meli74], [Abra78], [Lato76], [Lato78], [Suss75]. They often give approximate solutions, but generate them comparatively quickly, and are memory-efficient. These methods are also commonly applied in many related problems of graph theory and operations research and so-called "ill-structured problems" [Newe69] which promotes the idea of transferring them to the design automation systems. These methods have been already applied to solve many problems also in this domain, like: optimization of placement, Boolean function decomposition, decomposition of logic networks, wiring, layout minimization, multilevel logic networks synthesis, etc. These methods apply usually the collection of detailly considered heuristics, which essentially limit the number of variants searched by the computer to find the quasiminimal solution.

With respect to the above, as well as to some other requirements mentioned previously, DIADES system makes use of the different heuristic methods, especially tree-search methods.

2.3.5. Simulation

There exist two simulators in DIADES: the register-transfer simulator and the logic simulator. This results from the various requirements for simulation at the different design stages (requirement S2). The first of the simulators has to verify the consistency of the description with the design requirements and to analyze the weak points of the device. This program can simulate the device both from its behavior and structure description(*).

(*) The behavior description shall be not linked to simulation, and the structure description with synthesis. Both types of description can be used as well to synthesize and simulate the device.

The role of the second simulator is the detailed analysis of timing phenomena resulting from the gates' delays. Simulation of the entire system on the gate level would be usually technically nonrealizable, and if possible, it will produce a lot of printouts which are difficult to comprehend for analysis of the entire system correctness.

2.3.6. Documentation of the design data and process

The advantage of the design automation system in comparison with the methods of "hand design" is that the devices synthesized automatically are characterized in a systematic fashion and with uniformity, are created from standard, repeating segments and have uniform documentation. These properties help with the design process and with complete documentation preparation, which can be of more substantial value in the design of custom design circuits than their speed or their semiconductor area (requirement S5).

2.3.7. Openess and flexibility of the system.

An attempt was made to implement an open system, i.e., so that new statements of the source language can be easily added or removed, as well as the elementary units, abstract blocks, integrated blocks and symbolic layouts, transformations and programs, without substantial changes of the entire system or the methodology of design (requirements S11, S12).

For instance in the tree-search programs various search strategies can be declared, operators added, weights of parameters, cost and quality functions or restricting predicates interchanged.

2.3.8. Languages of the system

Considering generally the requirements formulated in section 2.1 and especially S13 we can conclude that the basic language of the system should be powerful and flexible, with properties of ALGOL68, ADA and modern languages of Artificial Intelligence like PROLOG or CONNIVER or that the system should apply several basic languages (current systems apply FORTRAN, ALGOL, PL/I, PASCAL, SIMULA and assemblers). DIADES uses a mixture of LISP, FORTRAN and PASCAL which was the best solution within the current possibilities.

Basic assumptions for the source language are discussed in section 3.1.

2.4. Process of design

The simplified schematics of the design process in DIADES system is presented in Fig. 2.2. It shows the design stages and points of human intervention in the process. The computer's part

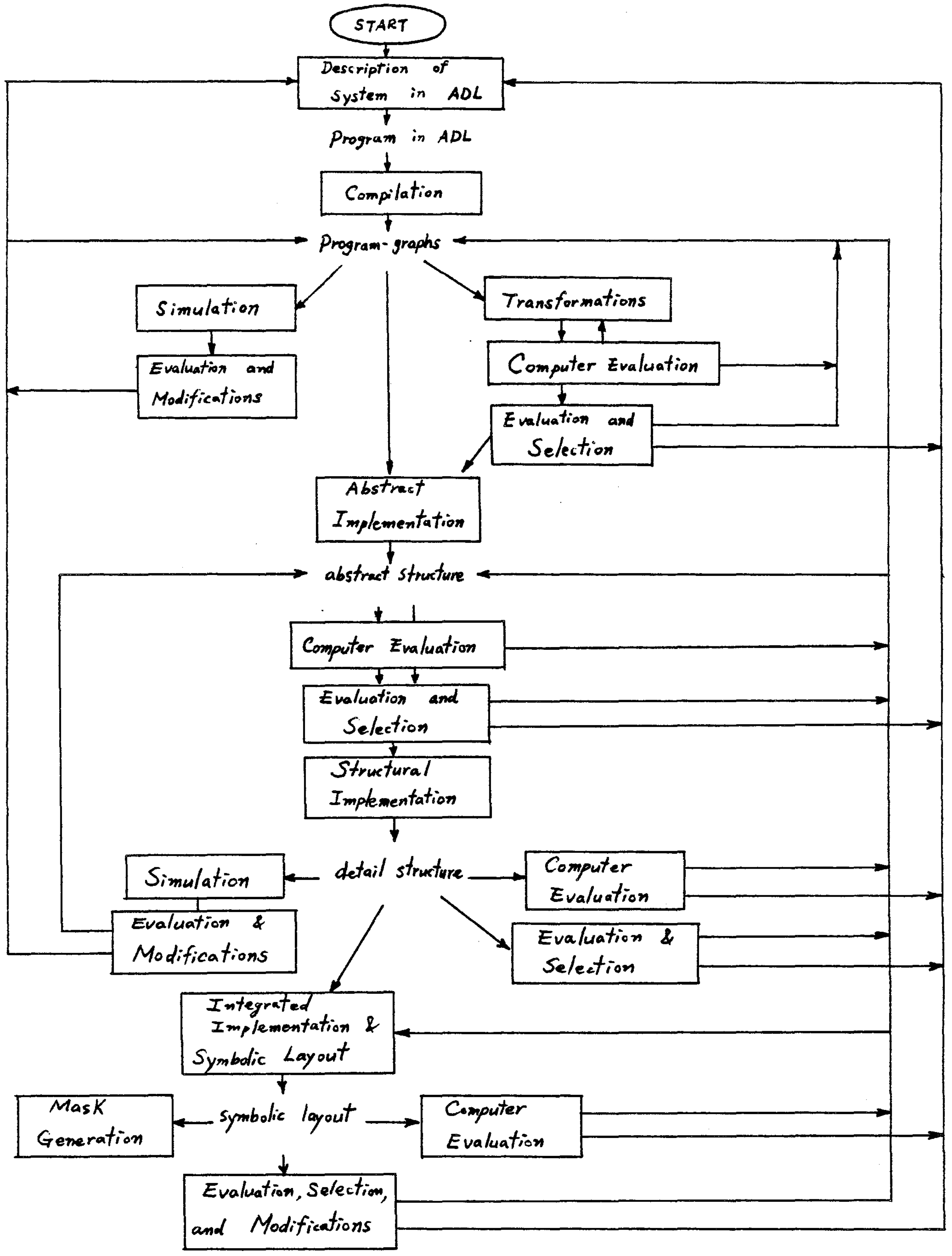

START
Description of System in ADL
Program in ADL
Compilation
Program-graphs
Simulation
Evaluation and Modifications
Transformations
Computer Evaluation
Evaluation and Selection
Abstract Implementation
abstract structure
Computer Evaluation
Evaluation and Selection
Structural Implementation
detail structure
Simulation
Evaluation & Modifications
Computer Evaluation
Evaluation & Selection
Integrated Implementation & Symbolic Layout
symbolic layout
Mask Generation
Computer Evaluation
Evaluation, Selection, and Modifications

Fig 2.2

of the process is shown in rectangles and the designer's - in ovales. The description in small letters specify the basic data of the system.

The input data is the program in the language ADL82 (Automata Describing Language-version 1982 Minnesota) especially created for the system's requirements. The standard output data is the mask description. The parts of the system can also be used: some transient data can be given as inputs and some other treated as results. This is not shown in the figure, as well as the control commands and languages.

The design process presented below results from the assumptions previously undertaken. The designed systems are described on many levels of abstraction (detailment): source programs, program-graphs, abstract structures, detailed logic structures, circuits, symbolic layouts, mask descriptions (with respect to requirement S1). These descriptions are decomposed in the design process into partial descriptions. In the design process the different variants of the descriptions are created (requirement S7) - part of them is stored for further transformations and part remembered in transitory form only. The programs are supplied with error diagnostics (requirement S10).

1. Source program in ADL82 is translated into program-graph in internal language GRAF. This graph corresponds to the flow diagram. Compiler of ADL verifies syntactical correctness of the description and to some extent the semantics as well.

   Program-graph is the basic, input data to main programs of DIADES system:
   - programs of optimizing transformations,
   - programs of abstract implementation,
   - program of register-transfer simulation.

   Execution of these programs is partially done in the interactive and partially in the autonomous mode.

2. Implementation includes several stages. Generally, it can be divided into stage of abstract implementation (more connected with mechanisms of ADL82 and block synthesis), stage of structural implementation (synthesis on level of gates and flip-flops) and stage of technological synthesis which first stage is the integrated implementation (transition to target technology) and next is the geometrical design.

   Abtract implementation of operational unit consists in transition from the statements of the program-graph to abstract structure, i.e., structure composed of abstract blocks. By abstract blocks we understand here such blocks as: counters, adders, logic gates with not specified numbers of bits, inputs and outputs. All statements of ADL82 have their counterparts in segments of control unit, and some (e.g. WAIT WHILE, +) in subunits of the operations unit. Implementation of statements applies the ready, standard patterns of units.

In the program-graph created from ADL program there exist two types of nodes: synchronous and asynchronous. Execution of synchronous nodes requires at least one pulse of the basic clock. Some statements are implemented as synchronous, other as asynchronous. The designer can declare with use of ASY operator asynchronous implementation of statements standardly implemented as synchronous. The direct, initial implementation of a program with complex statements leads to the quick device (because each such statement is executed in one pulse) but with large amount of hardware (because a separate, complex segment of operational unit corresponds to it).

3. The transformation programs cooperate closely with the abstract implementation programs. The program-graph is subject to several transformations and following them implementations which lead to new program-graphs and new abstract structures. These transformations lead usually to decrease in the number of statement types by simultaneous increase of total number of these statements, which will lead to obtaining variants even slower, but also even cheaper. The transformations split for instance complex, one-pulse statements into sequences of simple one-pulse statements, modifying respectively OU and CS until the quasioptimal variant with respect to the selected cost function is reached. The cost function being a weighted sum of the complexity and speed of the device is used.

4. As the result of simulation on the register-transfer level the nonproper behavior of the device can be detected. The user can do modifications both in ADL82 and in GRAF language (requirement S15).

5. The result of the abstract implementation programs is the creation of the detailed description of digital structures corresponding to the abstract structures. This stage, among others, includes such standard automata design procedures as: minimization of Boolean functions, etc. The designer evaluates the variants and selects the design procedures.

6. When the structure description is detailed with accuracy to basic logic gates and flip-flops the gate level simulation can be applied which can detect hazards, races, timing and logic inconsistences.

7. The detailed structure description is the starting point for integrated implementation procedures and next for the symbolic layout design.

The cost functions which evaluate the variants on the subsequent design stages are supposed to be selected in such a way that the cost function at the higher level roughly evaluates costs of different variants. The more detailed cost function for the counterparts of these variants on lower level of abstraction can be calculated not before the next design stage is executed, which process cannot be performed for too many variants. And so:

for the evaluation of program-graphs the cost function is applied which calculates number of nodes and complexity of their descriptons, which roughly evaluates the complexity of abstract structures being implementations of these graphs. The abstract structures are evaluated by the number and complexity of nodes and connections, and the detailed structures by the number and complexity of elements, blocks and wires and the estimates of the semiconductor's area occupied. The symbolic layouts only can be evauated with respect to the measure which is close to the real area. The minimization of area being one of basic criteria of the entire system - it is a criterium which cannot be calculated on the previous design stages. All the optimization methods applied by the programs of the system have however an auxiliary character only. The superior decisions with respect to the variants evaluation and the selection of the alternative design ways are undertaken by the designer himself. The selection decisions are undertaken on many levels of the design, both by humans and the computer. The data for these decisions, like the values of the cost functions are calculated by the computer and the essentially worse variants are deleted automatically. Schematics from Fig. 2.2 can be treated as a generator of equivalent solutions with two types of feedback, which specify selection of the next variants in correspondence to the evaluation of the variants already obtained. The internal feedbacks are realized by the selection of new variants with the aid of mechanisms existing in the optimizing and transforming programs. The external feedback, of the more essential meaning is realized by the human designer through his decisions communicated to the computer on the base of analysis of the sytem's generated solutions.

The external feedback loops from Fig. 2.2 show that on the base of the done by himself analysis of the variant the designer can come back to the previous stages of synthesis and make use of the stored variant descriptions or execute interactively corrections in some of the descriptions (usually in the description created as the last).

One of the most important designer's tasks is writing of the source program. According to the requirements of the customer and some other assumptions he must prepare the description of the system - usually more than one description is needed. The designer shall at this point consider only the information processing aspect of the system designed, not going into the details of its structure or the timing analysis.

Thanks to the decomposition of the designed system the designer can create many variants of the source description by replacement of some control lines of the program. This involves calling of the another design programs and data files and leads to joining of the descriptions from the different subcomponents.

The othe variants of the source description can be created successively as the results of printout observation from the design processes of the previous variants. Descriptions of new variants can be created with use of UPDATE or XEDIT programs (requirement S15).

Concluding, the creative behavior of human designer in the system consists in:

- creating source program describing behavior, structure or both of them of the designed system,
- specifying a decomposition by adequate source description,
- introducing corrections, control statements, auxiliary parameters, cost functions, restrictions, etc.,
- specifying ways of implementation of blocks and selection of transformations in the structure implementation,
- selection of operators and transforming strategies to optimize descriptions,
- organization of data files and ways of multivariant design,
- interrupting the design process and return to previous stages.

2.5. Structure of DIADES

At this point the difference between the set of CAD programs and the integrated CAD system must be pointed out. It is required from the system to have the unified data structures, the means for data-base handling, for cooperation with the operating system, the methods of the design process description an the total automation possibility in which only the source description is provided. Each program must have a certain place in the structure of programs, and must be able to cooperate with other programs of this structure. The important element of system are the unified data structures corresponding to the designed systems on the successive design stages. The execution of each design process can be organized in many ways by the designer. This can be done for instance through selection of the sequence of the programs called, their eventual repetitions and error escape ways organized on the operating system's level. He can use the disc and tape files. The system's diagram is presented in Fig. 2.3. The files are shown as well as the programs and languages.

The description languages are:

ADL82 -	source language for describing system under design,
INADL -	language with simulation conditions,
GRAF -	basic internal language of the system,
STRUCT1 -	language to describe digital structures on the abstract level,
STRUCT -	basic language of detailed description of digital structures,
DDL -	symbolic layout description (design description language)
SPICE -	circuit simulation,
DL -	Honeywell structure language (for interface),
GDL -	CALMA language (for interface).

The permanent files are:

INPU -	source programs in ADL,
KIMFOR -	description for simulation (GRAF),
INADL -	description of simulation conditions (INADL),
KIMOUT -	file for optimization, abstract implementation and control unit implementation (GRAF),
STRUCT1 -	results of abstract implementation (STRUCT1),
STRUCT2 -	abstract structure description after removal of subscripted variables and optimization (GRAF1),
STRUCT3 -	detailed structure including gates, flip-flops, ROMs, MOS, FET and groups of elements (STRUCT),
TARF -	target technology, elements as in STRUCT3 (STRUCT),
GAST -	as in TARF, without flip-flops (STRUCT),
FTABLE -	flow-tables of finite automata (arrays),
BOOLFUN -	Boolean functions (cubes arrays),
STRUCT -	detailed structure (FORTRAN form),
FLOOR -	floor plan description,
GBLDCK -	description of G-blocks, gates placed symbolically inside blocks,
MOSFET -	description of G-blocks, fets placed symbolically inside blocks,
PLSGB -	standard G-blocks for standard blocks,
ALOKE -	placed gates, placed fets, in placed blocks,
CONN -	routed connections,
TFILE -	symbolic layout text file (DDL),
GFILE -	graphic file corresponding to TFILE,
SPICE -	data for circuit simulation,
MOLOF -	results of logic gate simulation,
DL -	results in DL language,
GRAFIL -	pictures of logic diagrams,
MASF -	mask file (GDL),

The programs are:

-TAG -	compiler from ADL82 to GRAF language.
-IMPLEM1 -	program performing abstract implementation of program in GRAF. IMPLEM1 creates rough description of structures as a graph with abstract digital blocks as nodes and information paths as arrows.
-IMPLEM2 -	program of structural implementation of abstract structures in STRUCT1. It removes subscripted variables replacing them with ROM's, RAM's multiplexers, and demultiplexers. It executes also optimizing transormations on level of abstract blocks.
-IMPLEM3 -	program replaces the abstract blocks with detailed descriptions of structure from gates and removes descriptions of automata and Boolean functions realizing them with use of structural synthesis programs.
-IMPLEM4 -	program realizes optimizing transforms on the level of logic gates.
-COIM -	program of control unit implementation as finite automaton.

-OPTIMIZATION OF PROGRAM-GRAPHS - collection of programs for optimization of program-graphs.
-SIMADL - register-transfer simulator.
-MINIMA - minimization of automata.
-ASSIGN - state-assignment of automaton and calculation of excitation function.
-MNSA - joint minimization and state-assignment of automaton.
-LOGIC MINIMIZATION - collection of programs for minimization of Boolean functions.
-LIFO - translator from STRUCT4 to STRUCT.
-FOLI - translator from STRUCT to STRUCT4.
-FODL - translator from STRUCT to DL.
-MOLO - logic gate-level simulator.
-GOAL - graphic output program.
-IMPLEM5 - program realizes transition to the gates of the target technology.
LAYOUT - designs plans of gates' placement inside G-blocks,
FETPL - designs plans of fets' placement inside G-blocks,
FLOOP - designs floor plans for layouts,
COMBINE - combines results of LAYOUT, FETPL and FLOOP - prepares detailed plan of placement,
ROUTE - routes connections,
GUHA - creates symbolic layout,
SVRESH - interactive graphic editor of symbolic layout,
COMP - compaction of symbolic layout,
MASGEN - mask generation for CALMA's input,
DRAW - draws symbolic layout on monitor,
CIRCR - recognizes circuit description from symbolic layout description,
LOGR - recognizes logic network description from circuit description.

3. SOURCE DESCRIPTION LANGUAGE

3.1. General information of language ADL82

The source language of DIADES is called ADL82. Translator of this language is realized in University of Texas Lisp 4.0 (Minnesota version). ADL82 language was especially elaborated for the requirements of system DIADES. It is hardware description language for description of digital automata, blocks and systems. The language has a number of properties occuring in languages like ALGOL, LISP, CDL, APL, APDL, AHPL, PROTEKT. It has also various new mechanisms being the result of its specific application. In comparison with the known languages of this type ADL82 is a language of relatively extended grammar, flexibility and diversity of description means. Such form of the language is a result of an attempt to regard as far as possible all language requirements formulated in p.2.3.

The task of the present chapter is:
- discussion of general properties of language in view of the presented in p.2.3 requirements,

- presentation of basic new means of language and examples of digital systems descriptiion.

Full, formal description of ADL82 (abbrev. ADL) is given in Appendix A and description of the compiler in Appendix D. Appendix B includes description of GRAF language which describes results of ADL compilation.

3.2. General structure of language

1. For satisfaction of requirement J1 ADL is a high level language enabling description of the designed system on many levels of abstraction and in many ways. We can distinguish the following basic levels of description:
   - algorithmic,
   - microprogram,
   - functional,
   - structural.

   No other of the (known to us) HD languages includes as many possibilities of descripton on as many levels.

2. ADL satisfies both requirements L2 and L3, i.e., joins properties of behavior and structure description language. An ADL program is a formalized list description of flow-graph with parallel branches (describing behavior), description of automaton or logic function, or declarative description of system's structure in which elements and their mutual connections are described. All these descriptions can be on various levels of detailment. Behavior description in the form of flow-graph describes to some extent the structure as well (because certain subunits of the operations unit correspond to the statements). This structure is however not important in the simulation on the register-transfer level. The detalization of the description can be changed by the designer. He can for instance describe the system on the level of MOS transistors, ROM's and PLA's as well as single dynamic or static negative gates, specifying each wire, logic gate or register's bit. He can describe arithmetic and other dependencies on bunches of wires (variables). Finally, he can describe a system as a "black box" specifying only its input and output variables. ADL includes powerful possibilities of hierarchical joining of different types of descriptions where different types of description stand on different hierarchical levels.

3. For satisfaction of requirement L4 some mechanisms are similar to ALGOL. For the designer's convenience (and to enable the system to make transformations) ADL has complex arithmetic, logic, and mixed statements as well as parallel and waiting statements (requirements L7, L10). The notation of ADL is infix which makes it easier to use than for instance the Reverse Polish notation of APL. The complexity of the ADL syntax is of course reciprocal to the simplicity assumed in L4. We have tried, however, to obtain simplicity through application of identical or similar syntactic constructs for different aims (for instance the lF statement applied to the structure description has identical syntax but different meaning).

4. With respect to requirement L5 it was assumed that the designer knowing ALGOL and the requirements for the designed system should be able to write an ADL program after only a short acquaintance with it (the data obtained after translation of the program can be however far from optimum in such a case).

5. Mechanisms of ADL enable structural programming (requirement L4). It is suggested that the ADL programs be written with the "top-down" technique, i.e., first the behavior of the blocks is described and their interaction and next the lacking details are specified in subroutines of decreasing level.

6. Some properties of ADL, for instance its lexicography result from enbedding this language and its compiler in LISP. In particular, all names of ADL must be the literal atoms (strings of characters starting with letters). Atoms must be separated with spaces, comas or parentheses. This property of the language permits for satisfaction of requirement L16 - its result is the simplicity of the translation, in which the syntactic analysis consists in finding the "key words" in the list structure. A large amount of parentheses resulting from this lexics may appear inconsistant with the requirement of language clarity. It is however only a problem of the programmers' habits. Large amount of parentheses in light of their automatic enumeration by the LISP interpreter and other diagnostics of LISP can help the designer in detecting errors in syntactic constructs, especially the deeply parenthesized ones.

7. For satisfaction of requirement L6 it was assumed that the ADL programs are written by the designer not knowing the technology or specific design methods of the system. He should only know the syntax and the semantics of ADL and can express in it the properties of the designed system. On the other hand the designer knowing the digital design, standard circuits and systems as well as the implementation methods and the particular properties of ADL compiler and other programs of the system can write the program so that it is better implemented by the system. Understanding the principles of the translation, the implementation and the syntesis gives the designer better possibilities of the description (selection of statements and program's structure, decomposition, segments "fixing", selection of blocks by structure description, etc.).

8. Requirement L8 is partially satisfied: through possibility of creating new operators defined as functions. No one of the HD languages includes the full expandibility. In DIADES there exist however a unique advantage: the designer who knows LISP can quite easily extend ADL by the operation on ADL programs as on the list data structures of LISP.

9. Description of asynchronous and interative circuits is possible an relatively easy. For this the various means are discussed below, among others the ASY statement (requirement S9).

10. To fulfill requirement S9 by construction of ADL nearly 30 examples from different applications like automatic control, computers, measurement, automatic telephony were considered. It was concluded that the language permits the description of a wide class of devices. The serial, pulse, frequency, time, phase and other signals can be tried to be described by use of ADL means. Although there exist devices (with phase and time signals) which cannot be described in ADL or are difficult to describe in it - they belong to the analogue rather than the digital domain (and in our present opinion further extension of the language is not reasonable).

11. To make the documentation and the error diagnostics easier (requirements S5, L15) two types of comments are introduced: stored ones and those occuring only in input data.

12. For the requirements L4, L5 and L10 ADL has a more powerful syntax than the other HD languages (for instance since the subscripts or the parameters can stand for expressions and not only variables). Also the rules of automatic completing of bits in arguments of expressions without error signalization.

13. Similarly to LISP, ADL is a free format language which simplifies writing and modifying programs. The only limitation is that atoms cannot be split.

14. The satisfaction of requirement L17 was impossible from the practical reasons.

3.3. Declarations

1. Each program consists of a declaration and a program's body. Declaration specifies the type of variables: input (INPUT), internal (INTERN), output (OUTPUT), and indexing or subscripting (INDEX). Declaring of type assigns different meaning to variables and is also very useful for the designer.

   For each variable its sort is also specified. Variables can have one of the two sorts: P-parallel and D-logical. For sort P variables the code, number of bits and eventually maximal value is specified (for counters), and also if this variable is a bus. Logical variables results from an attempt of assigning different semantics to the operations on single-bit parallel signals and operations on logical signals, and also for the simplicity of the description. Logical variables are always the single-bit ones.

   Buses and related to them three-level gates give the possibility of easy implementation of many MOS circuits, for instance multiplexers.

2. The input variables have at any moment of time certain values, created outside the considered digital system. The program variables correspond to abstract blocks of the designed system, such as: registers, counters, memories, flip-flops, keys, input signals from the environment, generators, control lights. The variables are identified with outputs from blocks. The variables are divided into stored (memory) variables and functional (combinational) variables. Stored variables prolong their state for longer than one pulse of the basic clock. These are the variables from the left side of the assignment statements. These variables correspond to outputs from the memory blocks (counters, registers, flip-flops). Functional variables correspond to outputs from gates and combinational blocks.

3. Language ADL includes also constants and parameters. The constants can be integer numbers, octal numbers and binary vectors (for fulfillment of requirement L4).

   The parameters and their values are declared in the list of parameters. In the bodies of programs and subroutines the names of parameters are used. This gives the designer the possibility of easily changing values of parameters and investigating their influence on the generated detailed structures, described in ADL on the general level (requirements S1, S7).

4. In ADL there exist also the multiply subscript variables and parameters as well as arrays of variables and arrays of parameters. This gives, among others, the possibility of creating the two-dimensional tables, memories, ROMs, etc.

5. The designer can declare the different clocks (generators) and the various codes of variables.

3.4. Basic statements

1. The register-transfer statements in ADL are very general. On the right sides complex paranthesized expressions can occur, with variables, constants, parameters, arithmetical, logical, relational, concatenation and other operations. On the left sides stand the simple and complex variables as well as concatenations of variables, for instance:

```
        ((R ' X '(B [ 0 : 4 ])) :=

              (AND (((A - B)*(C + D)) < (X / Y)) X1 ( W)))
```

   The advantage of such expressions is, along with the user's convenience (requirement L4) also the possibility of splitting these statements into simple statements not before the stage of optimizing transformations (requirement S7). A hierarchy of the operators is always uniquely specified by use of the parentheses (requirement L4). ADL has comparatively many operators: for instance the arithmetical operators are +, -, /, * and logical AND, OR, NOT, NAND, NOR, EXOR.

2. For satisfaction of requirement S8, L2 and L5 the ADL language includes the <u>four assignment operators</u>: :=, =:, == and ::

   In the assignment to the stored variable A:

   1) Statement A := B means transfer of contents of B to A in which the new value is assigned at the back slope of the clock pulse (or with phase ϕ_2 - depending of the implementation),

   2) Symbol =: corresponds to writing the new value with the leading slope of the clock pulse (or with phase ϕ_1 - depending on the implementation),.

   3) Symbol :: means asynchronous (static) writing of new value.

   By transfer of B to the output variable A being not an internal variable - the value of A becomes equal to the value of B only for part of the transfer time:

   1) Symbol := means transfer gated with the product of control signal from CU and the clock of CU.

   2) Symbol =: means transfer gated with the product of a control signal from CU and the negation of clock from CU.

   3) Symbol :: means transfer gated with the controlling signal from CU.

   Symbol == means constant (galvanic) connection of variables. It is used mainly in the structural description (p.3.7) where it can mean also the conditional connection (depending on the context).

3. Together with the statement IF, known from other languages, (having here however different meaning - depending on context - see p. 3.7) we have included, for satisfaction of requirements L7, L4, L10 <u>the statement COND</u>, similar syntactically to LISP but with non necessarily disjoint or giving logic one as a sum, predicates. Such statement can then be used for the description of parallel processes - see p. 3.5.

   Statement IF has the form:

```
     (IF <predicate> THEN <string of statements>)
or
     (IF <predicate> THEN <string of statements>
                     ELSE <string of statements>)
```

The statement COND has the form:

```
(COND (<predicate> <string of statements>) ...
      (<predicate> <string of statements>)) .
```

Implementations of IF and COND statements are different. Statement COND serves for description of flow-graphs and automata with the nodes of complex branchings specified by the designer; the designer specifies here to some extent the implementation of the control unit. Together with the possibility of fixing (preserving from the transformations), the COND statement serves as a tool for satisfaction of requirement L11.

By the translation of sequence of IF statements the characteristic structure of conditional nodes is generated, which is next the base for subsequent transformations.

4. In ADL, as opposed to other languages (for instance LISP) logic variables have no special representations. The particular cases of predicates are then: sort D variables, single bits or sort P variables, Boolean functions and expressions on these data, as well as relations. This property gives the essential extension of the concept of predicate which extends the language's power.

5. With respect to different codes used in circuits ADL language has the means for the description of codes, and the change of codes.

6. For the description of some often occuring (for instance in the digital control) fragments of program-graphs we have introduced the waiting statements WAIT and CONTIN. The waiting statement can have one of the two forms:

```
(WAIT WHILE <predicate>),
(WAIT <time scale> <natural number>).
```

First of the statements means checking the predicate's value and if the condition is not satisfied - transition to the next statement occurs. If it is satisfied one waits till the next pulse and the condition is once more checked. The second statement means waiting as many time units as is the natural number, in the declared time scale (respective CLOCK). Statement CONTIN means waiting for a single pulse. This is useful in the description of synchronization processes and has also some syntactical meaning similar to the CONTINUE statement of FORTRAN.

In ADL one type of the iterative loop statement was adopted in the form

```
(WHILE <predicate> DO <string of statements>).
```

Arbitrary statements can occur in the string, including the waiting and parallel statements. Simplification of the description of simultaneous setting and resetting many signals (used in digital control systems) is done in the "set-reset" statement.

7. With respect to the requirements L1, L2 and L3 several languages (such as for instance APL) introduce a rich set of operators which process particular bits of variables or select some groups of bits. In ADL there exist elementary tools of this type and the possibility of constructing from them the more complex operators. In many languages there exist shift operators. These operators are intentionally not introduced in ADL: to encourage the designer to describe the respective operations as multiplying and dividing by some power of two. This permits to use for such program the optimizing transformations with arithmetical semantics. The user can also desclare the shift operators as some subroutines of respective hardware semantics.

8. ADL subroutines can be divided into macros and blocks. Macros can correspond to the description of behavior (MACRO) and the description of structure (LOGMACRO). Macros serve only to ease the life of the designer, by enabling single description of the multiple repeating segments, and also permit to fix some segments. The graphs of macros are included in the graph of the main program or the graphs of their superior subroutines and are not distinguished later on. Macro can be parametrical and nonparametrical. They cannot be recursive, so that creating of the macro loops is not permitted.

   In type MACRO and LOGMACRO subroutines the widths and the codes of variables can be declared as formal parameters. This gives several useful possibilities of applying the same subroutines, but calling them in different places of the program with different actual parameters.

   Blocks (subroutines of BLOCK and LOGBLOCK type) are treated as separate digital systems, interacting among themselves through connections. These connections correspond to variables declared in the block as input and output variables. For type BLOCK subroutines they are also often the formal parameters. Type BLOCK blocks are called by name with actual parameters from the superior program to the CU of the block. The end of work of the block is notified with wire RETURN to the superior's program CU's implementation.

   The type LOGBLOCK blocks can be either called in such a way (but the designer must describe the START and the RETURN signals inside these blocks) or they are only declared, and the designer must describe the interaction with them completely by himself with the aid of the corresponding statements in the descriptions of the superior and the subordinated systems.

   In the structural desciption the type LOGBLOCK blocks are called analogically to the LOGMACRO blocks, but separate systems will correspond to them in the implementation. Differently as in the case of type BLOCK blocks - each name of LOGBLOCK should be used only once and called with one set of parameters.

9. LOGBLOCK and LOGMACRO serve for the <u>structural description</u> (see p. 3.7), while BLOCK and MACRO for the <u>behavioral description</u> (requirements L1, L3).

In ADL the following methods of systems' description are possible:

1) <u>Behavioral descriptions</u> (procedural, sequential): these apply in the main program, the type BLOCK and the type MACRO subroutines. Operator STR is not used. This type of description corresponds to the sequential register-transfer languages but ADL additionally includes BLOCKS and MACROs as well as the BLOCKs can be parametrized.

2) <u>Structural descriptions</u>: these utilize the scope of STR operator in the main program and in the type LOGMACRO or type LOGBLOCK subroutines. This type of description corresponds to the structural description languages, but in ADL occur the LOGMACROs and the LOGBLOCKs, as well as the LOGMACROs are parameterized (the inconvenience of some structural description languages is the neccessity of using separate names for macros and variables describing devices of the same internal structure).

3) Mixed descriptions. In the main program, the type BLOCK and MACRO programs inside the scope of operator STR and type LOGBLOCK, LOGMACRO and BLOCK subroutines are used. Outside the scope of operator STR the main program and type BLOCK and MACRO subroutines can call type BLOCK and MACRO subroutines an type LOGMACRO functions. In the LOGMACRO functions, which are defined as LOGMACRO subroutines and called as functions from behavioral description - the output value is transmitted through a function's name (as for instance in FORTRAN). The mixed descriptions of ADL extend the possibilities of expressing different properties of complex systems. The particular cases that are possible here are:
- structural description in which blocks have procedural description,
- structural description from mixed blocks,
- hierarchial structural description from mixed blocks,
- procedural description calling structures.

The calls can constitute an arbitrary graph without loops while in most of the HD languages the description is two-level.

10. Blocks permit also for the stage after stage <u>synthesis of the entire system</u>: at the first stage the block can be declared as the "black box" in which only its external connections are specified, and after describing other interacting with it blocks the body of the block is created and the entire description transmitted to the further stages of the design process.

11. To enable the user to fix programs and subroutines from transformations executed by the system the fixing parameter FIX is introduced. The designer declares for instance para-

meter FIX in declaration section if he want the given MACRO already optimized by him not to be modified by the optimizing programs after writting it into the body of the main program. By adequate decomposition of the descripton into blocks and fixing some of them the designer can preserve the desired segments from changes. He can of course fix also everything.

12. The possibility of application of four types of subroutines and their fixing gives the designer wide possibilities of decomposing the source description and gives him opportunities for invention. He should, taking into account his knowledge of further design stages and various technological requirements, select and verify different possibilities of decomposition which he considers the best. Here is a field for heuristics and "try and error" variants investigation.

    Decomposition of the system can deal both with the operational unit and the control unit. BLOCK realizes separate control unit and separate operational unit. MACRO - common control unit and common operational unit. LOGMACRO describes a system which can describe both common and separate, control or operational unit, depending on the type of the calling program. LOGBLOCK permits the description of separate control units or separate operational units. The systems with one control unit and many operational units can be then described; systems with one operational unit but controlled by the hierarchy of control units; systems with dissipated control, etc.

    The blocks can be useful when we want to apply certain previously designed systems as a part of the new system under design. Such system's description is stored in the disc data-base. They are also useful when we want to transform and optimize some part of the description separately. This permits an arbitrary separation of big systems into subsystems to execute each time the implementation of the not too large systems.

13. The extended, with respect to other languages, decomposition possibilities (requirements L12, L14) result also from the parallel statements, as well as from replacing of the repeating expressions, fields of bits in variables and statements with the simple symbolic names. The respective declarations are placed only once inside the IDEN list or (in the case of the entire statements) in the SYMB list. During the program's compilation the symbolic names are replaced with the respective expressions or statements. Besides the decomposition this simplifies the team programming (requirement L12) and correction of the eventual errors (which is easy because of satisfaction of S15).

Example 3.1.

Resistors classifying device

Given is the unit from Fig. 3.1. At the output from the digital ohmmeter occurs the parallel variable X in binary natural code corresponding to the value of the register with number J (J=1,...,PAR), where PAR is a certain parameter. The resistor is selected by the analog multiplexer. Address on input to the multiplexer is in the natural code. The system shall be designed which calculates to which class the given resistor belongs and next sends the number of this class in natural code and in parallel to register Y (Y do not belong to the system under design).

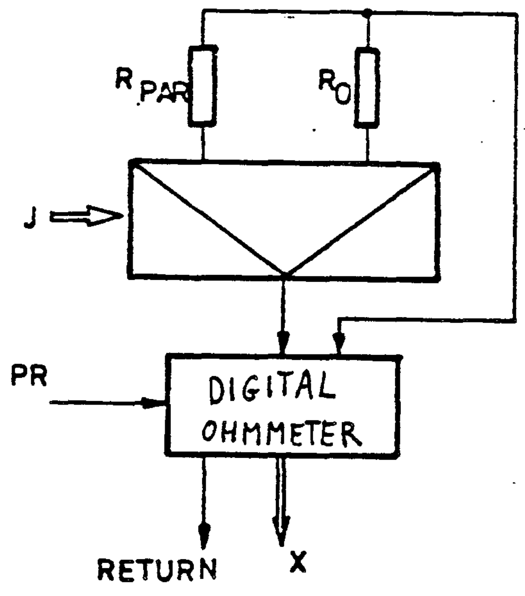


Fig. 3.1.

If X < 125 then Y = 3, if 125 ≤ X < 127 then Y = 2,
if 127 ≤ X < 129 then Y = 1, if 129 ≤ X < 131 then Y = 2,
if X ≥ 131 then Y = 3.

After the classification of all the resistors number 0 shall be transferred to the register Y. Processing in the digital ohmeter is initialized with signal PR. After processing the signal RETURN is created on the output of the ohmmeter. The system is ready after introducing the pulse signal START1 (given from outside after positioning of the resistors). The output variable is Y. This signal can be applied for printout of respective number or for creating signals for resistors sorting electromagnets (the repective units are not described).

According to the above formulation one can write an ADL program from Fig. 3.2. (a simpler program can be also written, however this one presents different properties of the language). The numbers from the left side serve as comments - they serve only to make presentation of the program easier. Enumeration of parentheses (done by the LISP interpreter) should enable the reader to distinguish each statement's scope.

Line 1 can include arbitrary text, for instance the name of the program and the number of the variant, designer's name, set of keywords, etc. Next row includes printout controlling statements for ADL compiler, discussed in Appendix A.

Lines 3 - 27 include the main program and lines 28 - 34 its subroutine. Lines 3 - 11 include the declaration of the main program, and lines 12 - 27 it body. Line 3 includes declaration of ADL language, symbol of the created graph - A, and name of the program - CLASSIFICATION. In line 4 the basic clock of frequency 1000 Hz is declared. List with atom C as the first element is a comment of ADL. Line 5 declares input variables to the system: X and START1. Variable X has sort P, code K1 and width 8. This denotes the parallel variable in binary, natural code which is stored in 8-bit word form. Variable START1 is a logical or sort D variable.

Simlarly line 6 declares internal variable J and line 7 declares output variables Y and J.

Line 8 declares that the subroutine of the main program has name PR and type BLOCK.

The FIX parameter in line 9 is followed with a list of formal parameters of subroutine PR. These parameters correspond to variables (signals) J and X which are used by BLOCK PR to interact with the main program CLASSIFICATION. Line 10 declares parameter PAR specifying number of resistors equals 49.

Lines 28 - 34 describe behavior of BLOCK PR. This BLOCK is described as a fixed "black box". We are not interested in the internal structure of the digital ohmmeter and in what is the electrial realization of the subscripted variable (R[J]). Really, variable (R[J]) is the analogue resistor's selecting unit - here it is described as a subscripted variable, which in the inplementation takes the form of the memory, the multiplexer or the demultiplexer (depending on the context). The implementation of block PR is however not interesting for us - this system is treated as an external system, which is designed separately. Body of the block (lines 32 - 34) means that after giving of the block's starting signal (i.e., PR=1 in the main program) the ten pulses of clock CLOCK must be waited (which corresponds to the time of the processing) and next in the register X the value of resistor with number J is placed. X is also an output variable.

The meaning of the remaining lines should not lead to difficulties as they are explained in commentaries in the program text. As the result of the above program's compilation the graph in language GRAF is created. It corresponds to that of Fig. 3.3a. The full descriptions corresponding to the statements numbers are given in Fig. 3.3b. Enumeration of the nodes was prepared by the TAG compiler. Let us note that the symbolic name of parameter PAR was replaced during compilation with its numerical value. Description of arrows: X-synchronous transition, E-asynchronous transition, <number of predicate> - transition with the satisfied predicate, (NOT<number of predicate>) - transition with the non-satisfied predicate. Sometimes for more clarity we will illustrate the programs or program-graphs with the use of flow diagrams. The flow-diagram for our example is presented in Fig. 3.4.

The transitions with satisfied conditions are denoted by T, transitions with not satisfied conditions by F. Numbers of graph nodes are denoted by Ai. We will also use below the graphs in which full descriptions of nodes are specified. Descriptions of GRAF's nodes and arrows are presented in the LISP prefix notation. In the Figures for more clarity they are in mathematical-infix notation. By drawing flow diagrams we will assume a shortened description for groups of program-graph nodes corresponding to statements: (WAIT WHILE <predicate>) and subroutine calling.

As the result of the abstract implementation of this graph we obtain a structure of the system corresponding to the diagram from Fig. 3.5a (Block PR together with its control is shown schematically without distinguishing its elements, because we are not interested in its implementation. Its description in ADL is required for simulation purposes only). As a result of the structural simulation the counter J and the four comparators are designed from the components and the unit of output variable Y is replaced with the unit from Fig. 3.5b.

By L we denote the writing-in control input to registers and other blocks, by SU - the subtracting input to counters. To explain the semantics of the control unit simple implementation of this unit is shown in Fig. 3.6. The reader is asked to compare this figure with that of 3.3a.

3.5. Description of parallelism

1. For satisfaction of the L7 requirement the ADL language is equiped with powerful notions to describe parallelism on several levels of description. These notions are stronger than those of the other HD languages in which there exists only the possibility of declaring parallelism on the of statements' or subalgorithms' level. The notions applied in ADL include possibilities of describing parallel program-schemata of Karp. They can then theoretically extend the descripton's power of the Petri nets and are not weaker from

the various parallelism models theoretically equivalent to the parallel program schemata (ADL additionally includes statements DEXOR and DROP). In the parallel braches of the ADL description arbitrary statements can stand (including jumps, conditional statements and other parallel statements). Different operators of control joining permits a description of different types of synchronization. Parallelism can also be expressed in the structure description.

2. There are statements of parallel execution of sets of statements in the HD languages. ADL distinguishes two types of parallelism: on the pulse and on the program levels. Parallelism on the pulse level is described with the SIM statement. Arguments of SIM are the assignment statements which are to be executed simultaneously, in one pulse. Application of SIM speeds up the program's execution in comparison with the sequential program or with the program, in which SIM statement was replaced with another parallel statement - FORK. Implementation of program with FORK is more complex because of greater generality of this statement.

3. FORK is the description of program's graph node in which the control (the control dot) <u>disjoints into parallel branches</u>. Starting from the moment of the control coming to FORK - the separate parallel actions are executed in the branches of this statement. After execution of the branches or of one of them the two or more controls can be once more replaced with one control in the control's joining node.

4. Statement (DROP) on the branch's end means interrupting the program's execution <u>in this branch</u> (local removal of the control). The removal is done when the DROP node is reached. DROP ends control in one of the parallel branches changing not the controls in the others.

5. Node COND with nondisjoint predicates also <u>disjoints control into parallel branches</u>. Control transfers to these branches which corresponding predicates are fulfilled. In particular cases it can then transfer to all (as in FORK) branches or to none of them (as in DROP). FORKs and DROPs can then be theoretically replaced with CONDs but it could lead to the nonclarity of the descriptions and to the too complex implementations. It can be shown that COND with nondisjoint predicates can be replaced with a set of FORK and IF statements, but once more this would lead to similar inconveniences.

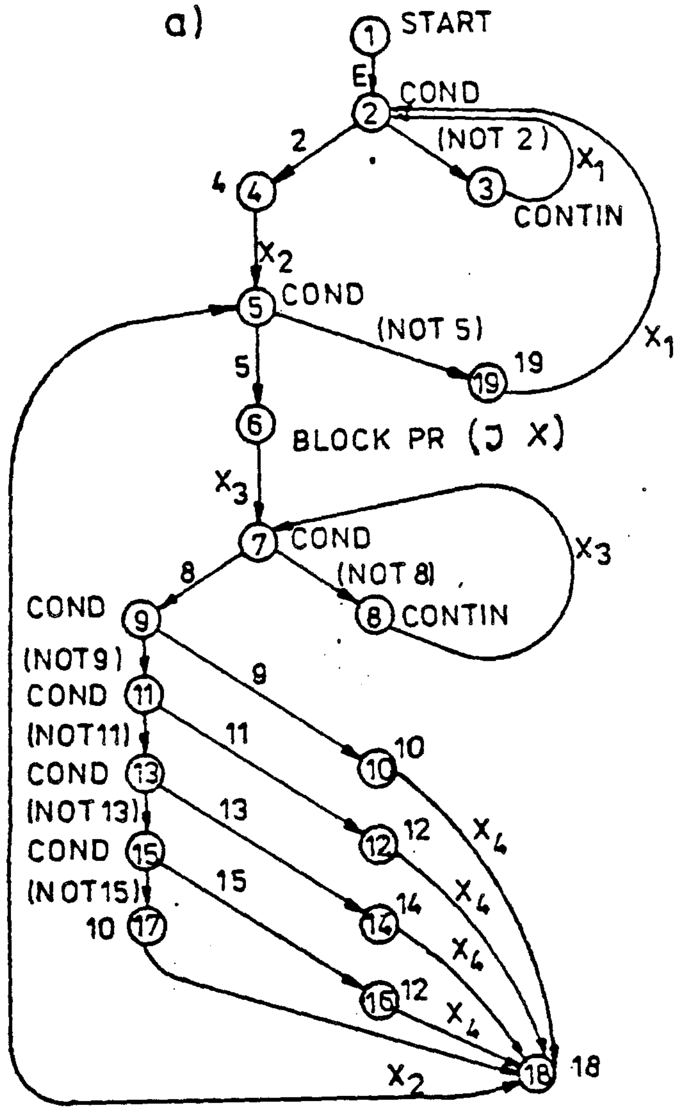
6. DAND and DEXOR describe nodes in which <u>control joins from parallel braches to a single branch</u>. Nodes DAND and DEXOR arise in the graph either from the last element of FORK statement or from separate, labeled statements: respectively (DAND) or (DEXOR), to which lead the jump statements.

   <u>DAND</u> describes a node which is passed by the control iff the controls of all the inpointing branches have yet arrived.

```
↓ 1 ↓   (BLASSIFICATION   VERSION 4   PERKOWSKI  (BLOCK CHANGED * ))
↓ 2 ↓   | |
↓ 3 ↓   ((( ADL   A   CLASSIFICATION
↓ 4 ↓         ((CLOCK (1000))) (C DECLARATION OF BASIC CLOCK 1000 HZ)
↓ 5 ↓         (INPUT (X (P K1 8)) (START1 (0))) (C INPUT VARIABLES)
↓ 6 ↓         (INTERN (J (P K1 6)))     (C INTERNAL VARIABLE)
↓ 7 ↓         (OUTPUT (Y (P K1 2)) (J (P K1 6)))
↓ 8 ↓         (SUBR ( BLOCK B  PR)
↓ 9 ↓                                      (FIX J X))) ( C SUBROUTINE)
↓ 10 ↓        (CONST (PAR 49)) (C DECLARATION OF PARAMETER PAR)
↓ 11 ↓                (C END OF THE DECLARATION SECTION ) )
↓ 12 ↓        ((START) A  (C BEGIN OF PROGRAM'S BODY)
↓ 13 ↓          |     (C LABEL)
↓ 14 ↓        (WAIT WHILE (NOT START1)) (C WAIT FOR SIGNAL START1)
↓ 15 ↓        ( J := PAR)  (C ASSIGNMENT  STATEMENT)
↓ 16 ↓        ( WHILE  (J > 0)  DO   (
↓ 17 ↓                (C  BEGIN OF DO LOOP)
↓ 18 ↓        ( PR  (J  X))   (C  CALL  OF  SUBROUTINE)
↓ 19 ↓        ( IF (X < 125)  THEN  (( Y :: 3) ( GO  11)))
↓ 20 ↓        ( IF (X < 127)  THEN  (( Y :: 2) ( GO  11)))
↓ 21 ↓        ( IF ( X < 129)  THEN  (( Y :: 1) ( GO  11)))
↓ 22 ↓        ( IF (X < 131)  THEN  (( Y :: 2) ELSE  (Y :: 3))
↓ 23 ↓    ||  (J := (J - 1))  (C  DECREASE  RESISTORS' COUNTER)
                         (C  END  OF  DO  LOOP)
↓ 24 ↓
↓ 25 ↓        ( Y :: 0 )   (C  CLEAR  OUTPUT  VARIABLE  Y )
↓ 26 ↓        ( GO  1 )    (C  JUMP  TO  THE  BEGINNING)
                    (C  END  OF  THE  MAIN  PROGRAM )  ))
↓ 27 ↓
↓ 28 ↓  ((ADL   B    BLOCK    PR  (J X)  (C SUBROUTINE)
↓ 29 ↓      (INPUT  (J  (P  K1  6)))
↓ 30 ↓      (INTERN  ((R [ J ] )(P  K1  8)) (X  (P  K1  8)))
↓ 31 ↓      (OUTPUT  (X  ( P   K1  8)) ))
↓ 32 ↓      ((START)   B            (WAIT  CLOCK  10)
↓ 33 ↓                              ( X  :=  (R [ J ] ))
↓ 34 ↓                              ( RETURN))
↓ 35 ↓  END
```

Fig. 3.2

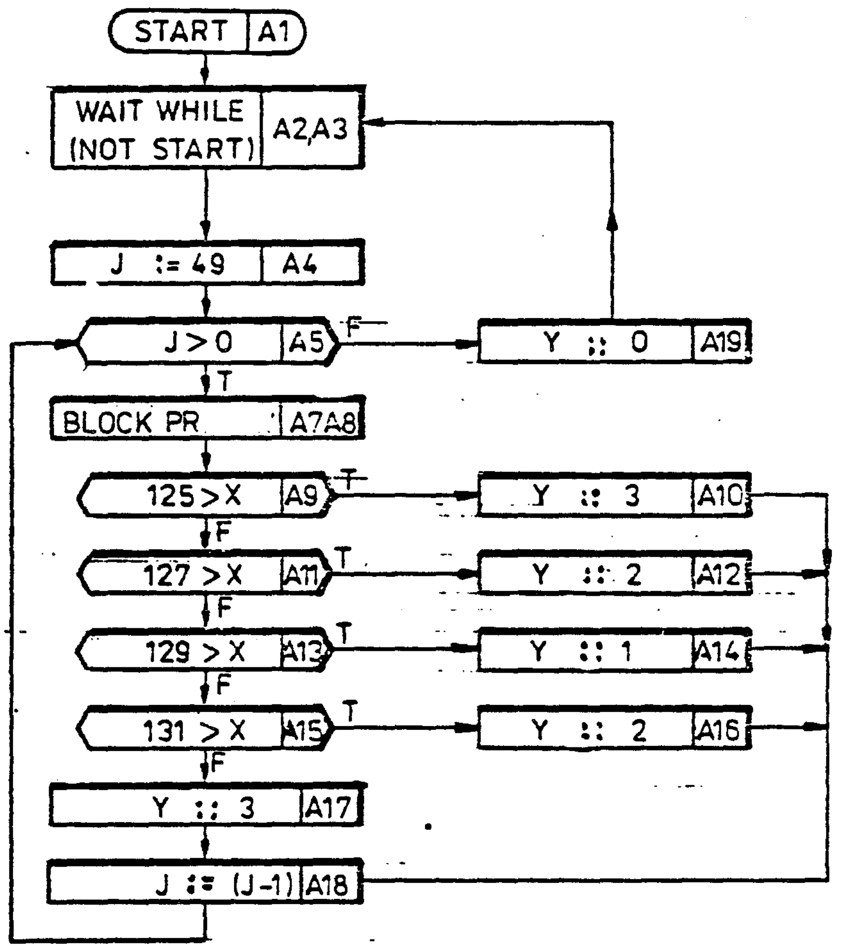
a)



b)

2 - START 1
4 - J # = 49
5 -(J > 0)
8 -(RETURN. PR)
9 -(125 > X)
10 - Y # # 3
11 -(127 > X)
12 - Y # # 2
13 -(129 > X)
14 - Y # # 1
15 -(131 > X)
18 - J # = (J - 1)
19 - Y = = 0

Fig .3.3



START A1
WAIT WHILE (NOT START) A2,A3
J := 49 A4
J > 0 A5
F
Y :: 0 A19
T
BLOCK PR A7A8
125 > X A9
T
Y :: 3 A10
F
127 > X A11
T
Y :: 2 A12
F
129 > X A13
T
Y :: 1 A14
F
131 > X A15
T
Y :: 2 A16
F
Y :: 3 A17
J := (J-1) A18

Fig .3.4

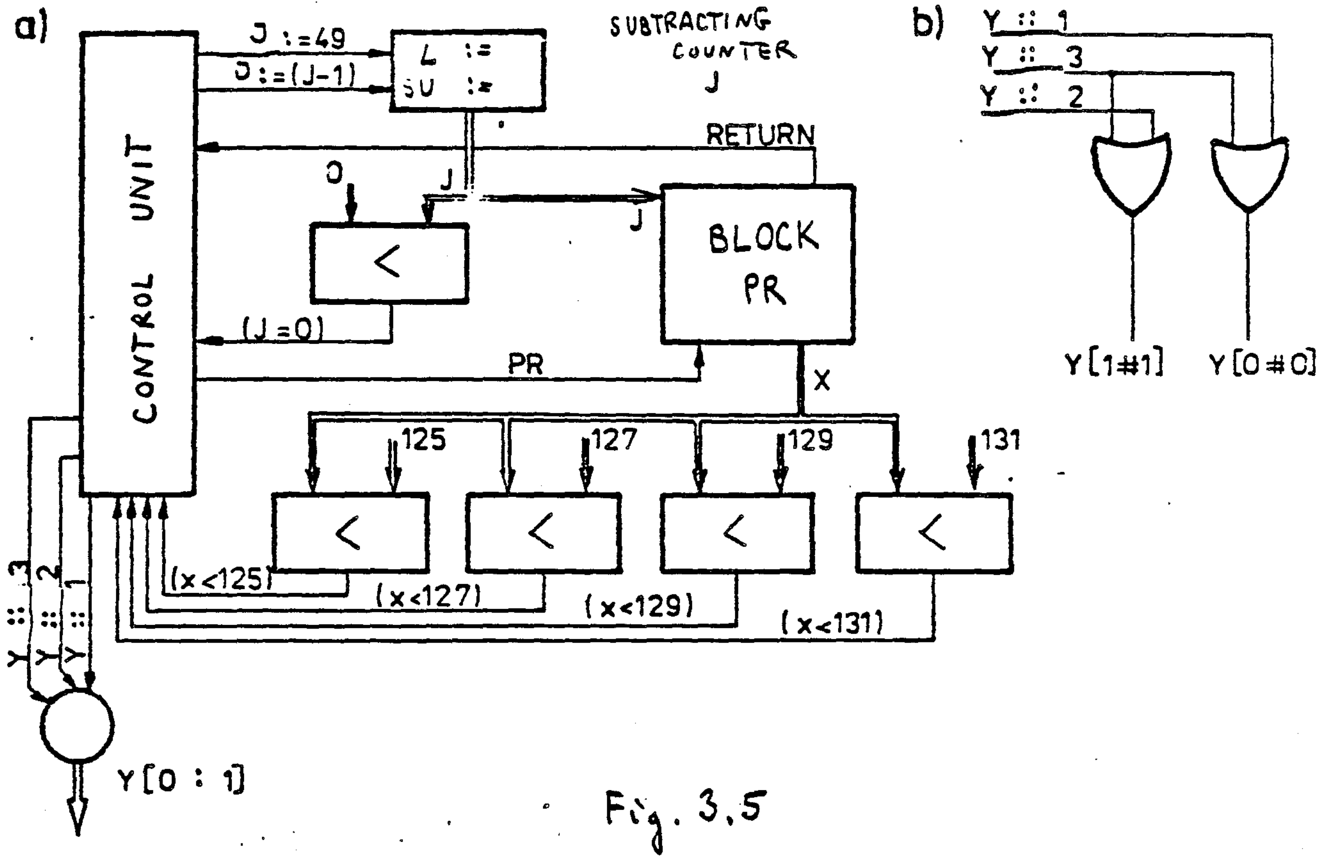
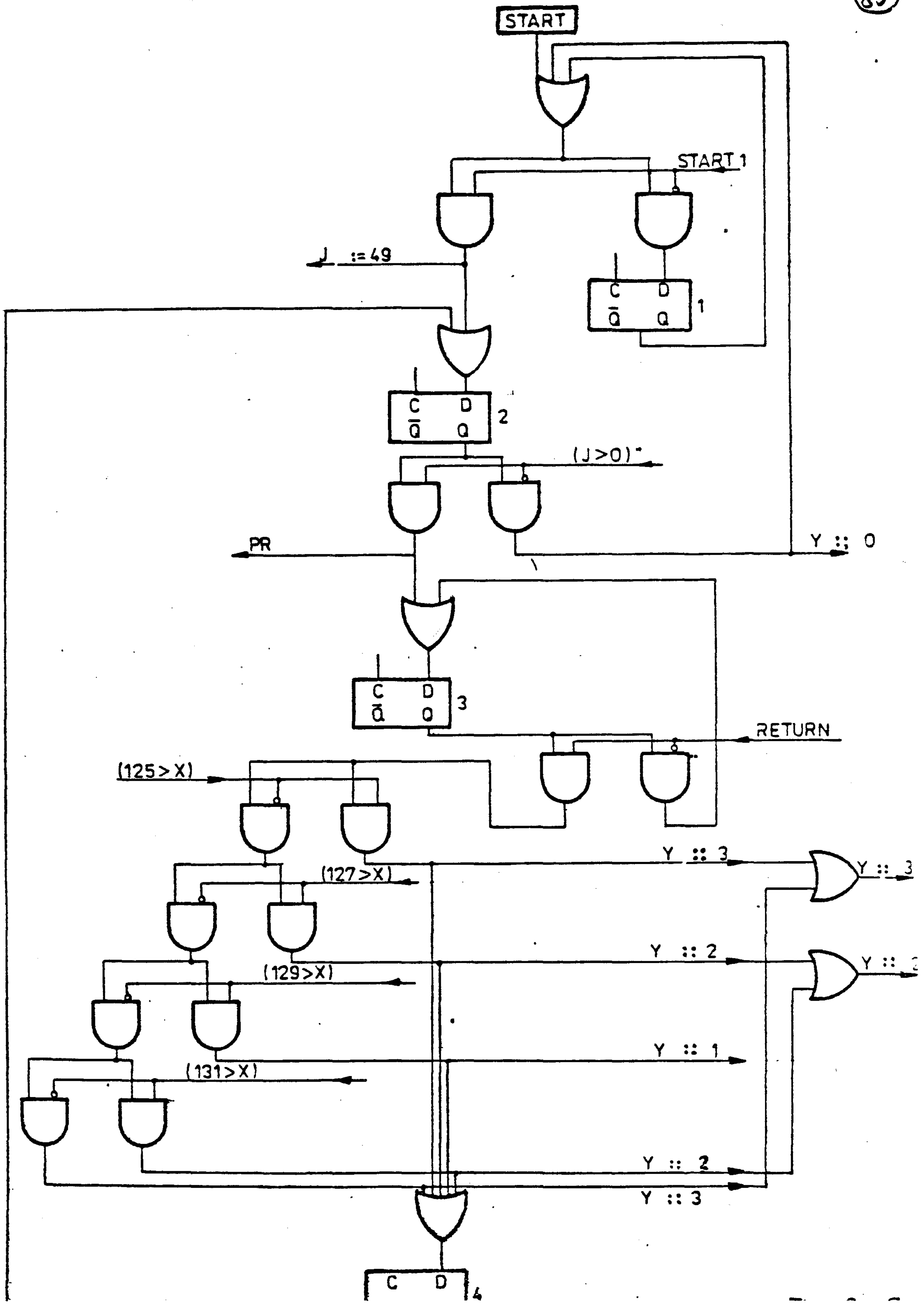

a)
CONTROL UNIT
J :=49
J :=(J-1)
L :=
SU :=
SUBTRACTING COUNTER J
RETURN
0
J
J
<
BLOCK PR
(J=0)
PR
X
125
127
129
131
(x<125)
(x<127)
(x<129)
(x<131)
Y :: 3
Y :: 2
Y :: 1
Y[0 : 1]
b)
Y :: 1
Y :: 3
Y :: 2
Y[1#1]
Y[0#0]

Fig. 3.5


START
START 1
J := 49
C D
Q̄ Q
1
2
(J>0)
PR
Y :: 0
3
RETURN
(125>X)
(127>X)
(129>X)
(131>X)
Y :: 3
Y :: 2
Y :: 1
Y :: 2
Y :: 3
4

<u>DEXOR</u> describes a node which is passed in the moment when the control from any of the inpointing branches comes. The controls from some (described below) branches is removed.

In the scope of FORK statement arbitrary statements can occur. For instance:

```
(FORK (d1 (Go 111))(d1 d3(DROP))(d4 d5)(d6 d7) DEXOR)
```

The first branch ends with the jump outside the branch. Both: the jump to the place inside other branch of the same FORK, and the jump outside FORK can occur here. The second branch ends with the DROP statement. Joining operator DEXOR concerns then only the third and fourth branch.

As it was already written node DEXOR in the graph can be either created from the last element of FORK, or from the separate statement. In both cases control coming to such a node causes the removal of the controls from some nodes, but these nodes can differ in both cases.

<u>Case I</u>

If DEXOR is the last element of the FORK statement then execution's ending of any single branch will cause ending of the all other statements <u>inside</u> the scope of <u>this</u> FORK. Next transition to the execution of the statements next to FORK occurs. This ending concerns only the statements inside FORK branches. It means that the execution of statements outside FORK to which jumps from inside FORK have occured - are just not interrupted. In the case of nesting one FORK statement in another FORK statement this rule concerns of course only this FORK statement in which given DEXOR is a joining of control, and all statements in its scope.

<u>Case II</u>

If (DEXOR L1 L2 L3 ... Ln) is a separate, labeled statement then it is assumed that it removes control from all the nodes corresponding to the labels L1,...,Ln being the arguments of this statement and the node corresponding to this statement.

The above rule gives the designer flexible possibilities of removing control. The set of the nodes from which the control is removed with respect to a certain DEXOR node will be called the <u>scope</u> of this node. Let us consider now some examples of FORK statements.

<u>Example 3.2</u>

To the statement

```
(FORK (d1 d2 d3 (DROP))
      (d4 d5)
       d3
      (d2 (GO 1))
                   DEXOR)
```

corresponds the subgraph from Fig. 3.7. By application of the above description, the control occuring in node 10 removes control from nodes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. If we describe this graph as follows:

```
     10(FORK (d1 d2 d3 (DROP))
             (4 d4 5 d5 (GO 3))
             (6 d3 (GO 3))
             (d2 (GO 1))       )
       ...
     3(DEXOR 10 4 5 6 3)
```

then according to the Case II the control is removed only from nodes 1, 6, 7, 8, 10, i.e., in all branches inpointing to node 10.

Example 3.3

Graphical representation of the statement:

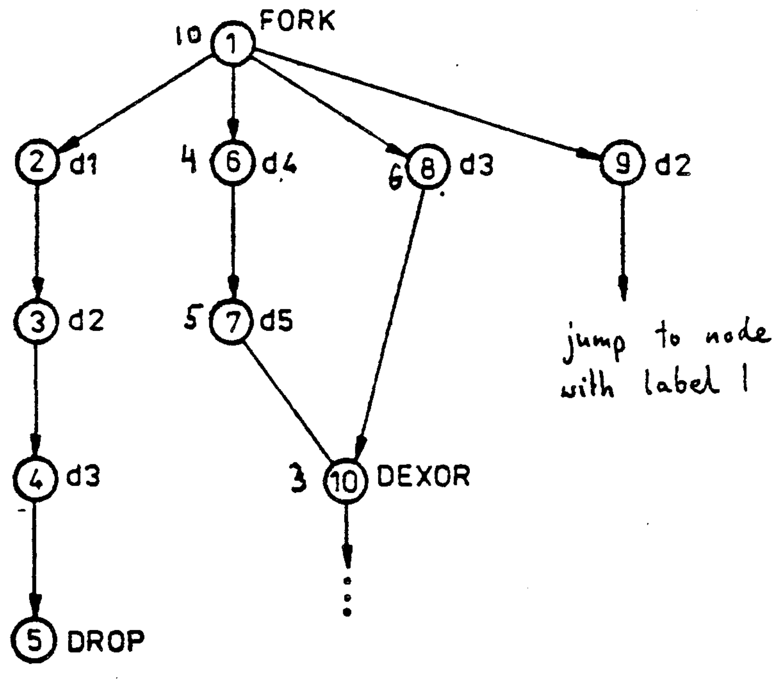
```
10 d4 (GO 2)
20 (FORK ( <FP1> 1 (DAND) <FP5>)
         (2<FP2> (FORK(<FP4>(GO 1))
                      <FP3>))
                            DEXOR)
```

is presented in Fig. 3.8a.

As it can be noticed the segment FP5 can be executed not before segments FP1, FP2 and FP4 are executed. Execution of FP3 or FP5 ends all other controls in the graph from Fig. 3.8a, except control in node 7. The description with separate (DEXOR) is:

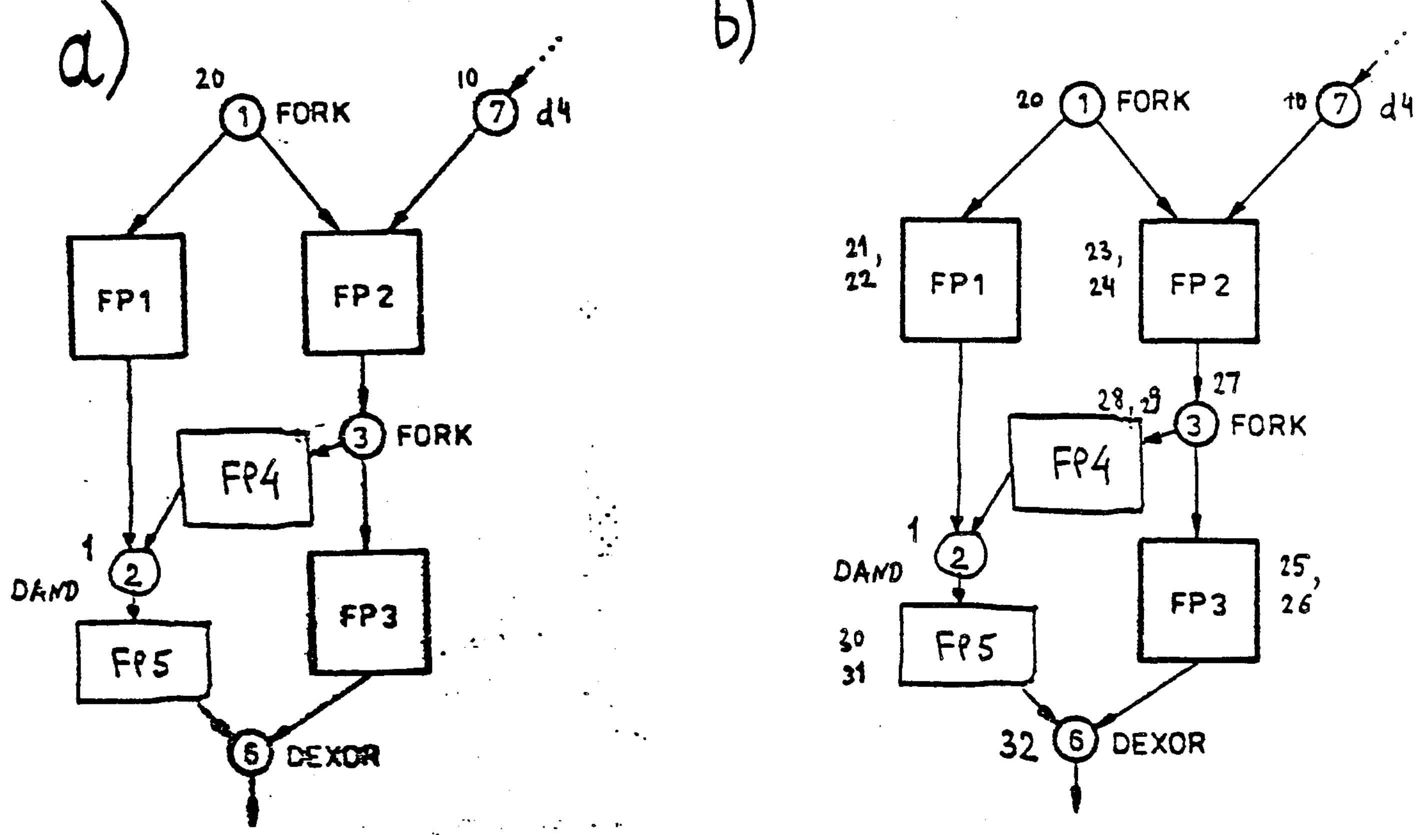
```
10 d4 (GO 2)
20 (FORK (<FP1> 1 (DAND) <FP5>)
         (2 <FP2> (FORK (<FP4> (GO 1))
                        (<FP3> (GO 32)))))
32 (DEXOR 25 26 27 28 29 30 31)
```

It is illustrated with Fig. 3.8b. The labels used inside the segments <FPi> stand near them. In this case the control is not removed from FP2 and node 7 when control comes to node 6. This example illustrates "the memory of the controls" which can exist in the parallel program. Let us assume that the control from node 1 has come to node 2; in one branch control had passed FP1, in the second branch it had terminated in some DROP statement in FP2. If now the control will come from node 7 and will pass FP2 then after coming to node 2 it will meet with the "waiting" control from the last run and they will together pass node 2. If this control after executing FP5 will come to node 6 before the control from FP3 comes then the control from FP3 will be removed. However, if control outcoming from FP2 will pass FP3 quicker then the FP4 is executed then the "waiting control" on output of FP1 and conrol in FP4 are removed. The descriptions of this type have extended possibilities.



10
FORK
2 d1
4 6 d4
6 8 d3
9 d2
3 d2
5 7 d5
jump to node with label 1
4 d3
3 10 DEXOR
5 DROP

Fig. 3.7.



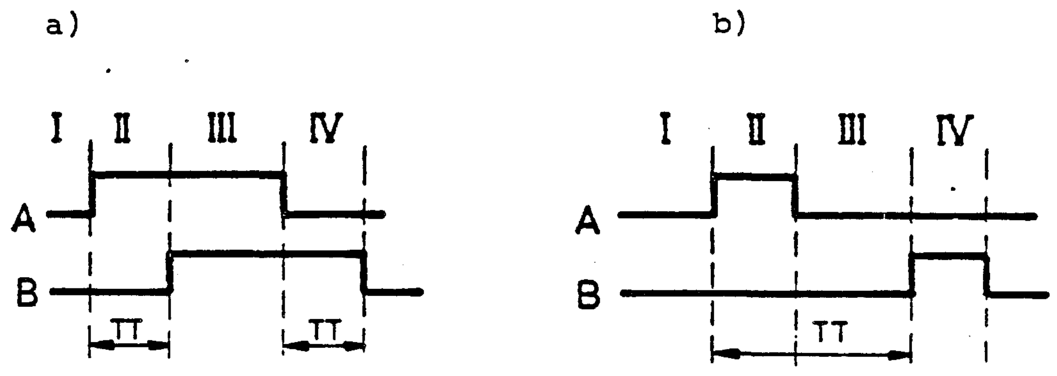
a)
20 1 FORK
10 7 d4
FP1
FP2
3 FORK
FP4
1 2 DAND
FP3
FP5
6 DEXOR
b)
20 1 FORK
10 7 d4
21, 22 FP1
23, 24 FP2
27
28, 29 FP4
3 FORK
1 2 DAND
FP3 25, 26
30 31 FP5
32 6 DEXOR

Example 3.4

Pulse delaying device

The device DELAYP shall be designed for delaying signals A by time TT. A device has signals input A, signals output B and control data input TT. A and B are type D variables and TT is the parallel variable. Its value is the required delay time TT calculated in the number of CLOCK pulses. The timing diagram for Case 1, when the delay time is less then the length of the signal A (the number of pulses when A=1) is presented in Fig. 3.9a. Diagram from Fig. 3.9b corresponds to the Case 2 when the delay time is greater than the length of the signal A=1. We assume that the following conditions are satisfied:

(i) In the initial moment A=0.
(ii) The time lag between two signals is longer than TT.
(iii) Value of TT does not change during the measurement.



Case 1 Case 2

Fig. 3.9.

The flowgraph of the device is shown in Fig. 3.10a, the program-graph in Fig. 3.10b and the respective ADL program in Fig. 3.11. To clarify, we do not enumerate statements but we give their full description.

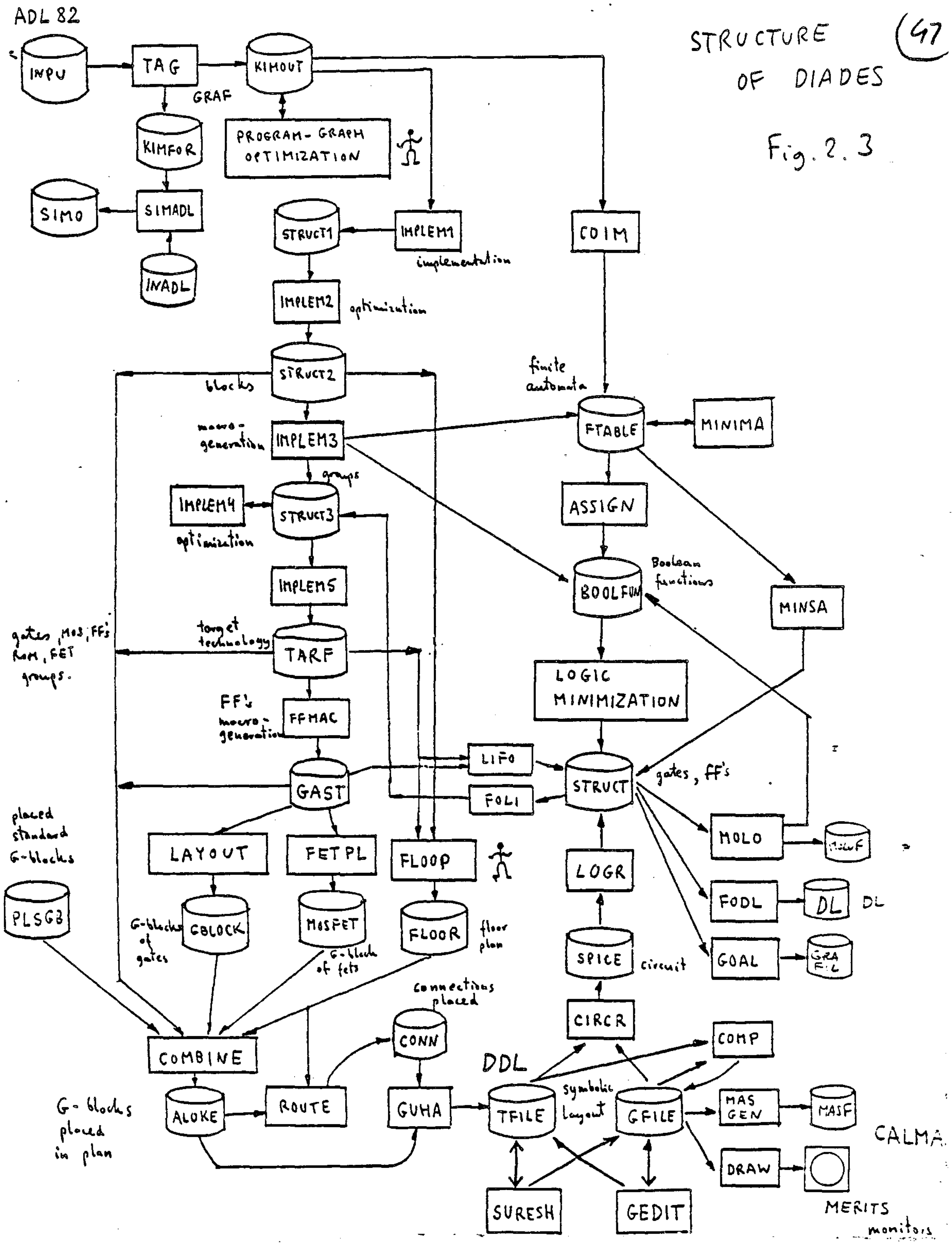
Behavior of this automaton will be described in more detail because it will be useful in further examples.

For logical variables (and single bits or Boolean functions playing the role of predicates) we shall adopt the following notation: (A=1) and (A=0) is replaced with A, and (A=0) and (A=1) is replaced with A.

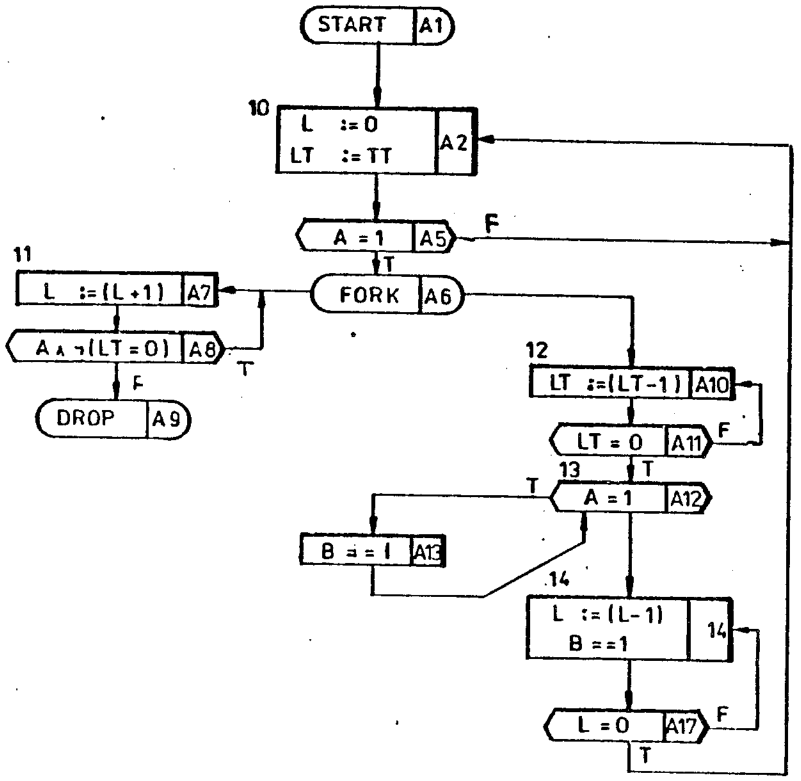
Statements A2 and A5 correspond to the state of the control automaton, in which signal A is initially equal to zero (phase I of Fig. 3.9). When A changes to 1 the branching is executed by FORK statement. In one branch ones are added to counter L

STRUCTURE OF DIADES

Fig. 2.3

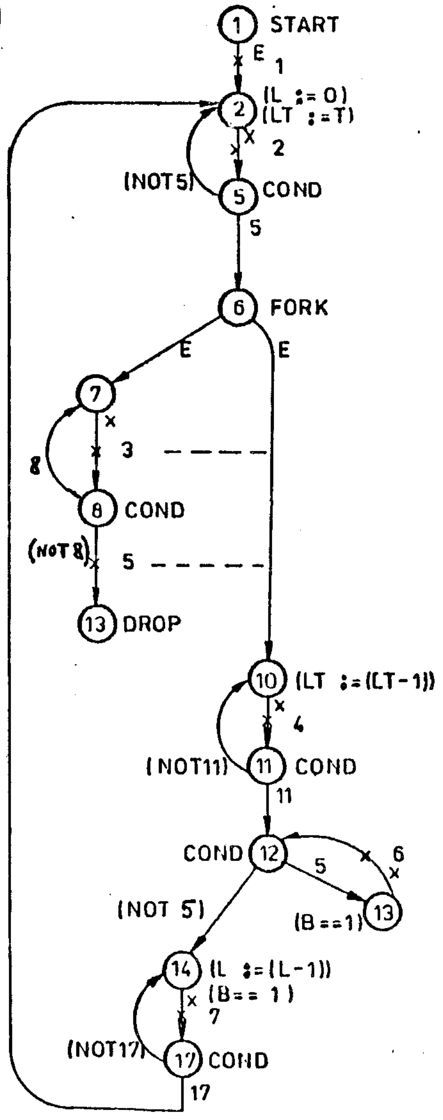


a)



START A1
10
L := 0
LT := TT
A2
A = 1 A5
F
T
11
L := (L+1) A7
FORK A6
A ∧ ¬(LT = 0) A8
T
F
DROP A9
12
LT := (LT−1) A10
LT = 0 A11
F
13
T
A = 1 A12
T
B == 1 A13
14
L := (L−1)
B == 1
14
L = 0 A17
F
T

b)



1 START
E
1
2 (L := 0)
(LT := T)
2
(NOT5)
5 COND
5
6 FORK
E
E
7
3
8
8 COND
(NOT 8)
5
13 DROP
10 (LT := (LT−1))
4
(NOT11)
11 COND
11
COND 12
5
6
(NOT 5)
(B==1) 13
14 (L := (L−1))
(B== 1)
7
(NOT17)
17 COND
17

Fig. 3.10

92

```
   (DELAYP)
   1     1
 1 1

 (((ADL C OPINP
 123
     ((CLOCK(1000)))
     45      5     654
     (INPUT(A(0))(TTP K1 A)))
     4     5 6 655 5       654
     (INTERN(L(P K1 A))(LT(P K1 A)))
     4      5 6     655  6      654
     (OUTPUT(B(0))))
     4      5 6 6543
    ((START) C
    34     4
10   (SIM (L #= 0) (LT #= TT ))
     4    5     5 5        54
     (IF(^ A)THEN(GO 10))
     4  5   5    5     54
     (FORK
     4
       (11 (L #= (L + 1))
       7   6     7    76
        (IF(AND A (^(LT = 0)))THEN (GO 11))
        6  7      8 9       987     7   76
        (DROP ))
        6     65
       (12(LT #= (LT - 1))
       7  6      7     76
        (IF(^(LT = 0))THEN(GO 12))))
        6  7 8      87    7     7654
       13(IF A THEN((B == 1)(GO 13)))
         4         56       66     654
       14(SIM
         4
         (L == (L - 1))
         5     6    65
         (B == 1))
         5      54
     (IF(L = 0)THEN(GO 10)ELSE(GO 14))
     4  5     5    5     5    5     54
     )))
     321
END
```

Fig. 3.11

while A=1 and LT≠0 (statements A7, A8) In the second branch ones are subtracted from the counter LT while LT≠0 (phase II - statements A10 and A11).

Let us consider case 1 from Fig. 3.9a. If A=1 when LT is set at 0, control goes on to statement A13. Next the device wait while A=1 and gives 1 on output B (statements A12 and A13 - phase III). When A changes to 0, the device counts time of delay stored in counter L, all the time giving 1 to output B (statements A14, A17 - phase IV). After counting the time, device goes to the phase I when L is set at 0 (jump to statement A2).

In the case of Fig. 3.9b the device executes statements A6, A7 in one branch and A10, A11 in the second one until A will change to 0. Counting up in L counter will be interrupted and counting down will continue in LT counter (statements A10, A11 - phase V). At the moment of LT changing to 0 (i.e., time TT has been counted) the value of A is checked. It is equal to 0, so the device executes statements A14 and A17 (phase IV) as in the first case.

Statement DAND corresponds to the existing in some parallel programming languages statement JOIN while DEXOR and DROP are the new ones. Their introduction results from the neccessity of describing wider classes of concurrent devices.

Let us notice that DAND statement can be replaced with statement W waiting for fulfillment of the logical product of conditions pi. Each of the conditions is set to 1 in the last node of each branch inpointing to statement W and reset after passing statement W.

Similarly node DEXOR can be replaced with an ordinary node with inpointing branches (DOR) but additional "producting" (as in Fig. 3.12) of all operations executed in nodes from

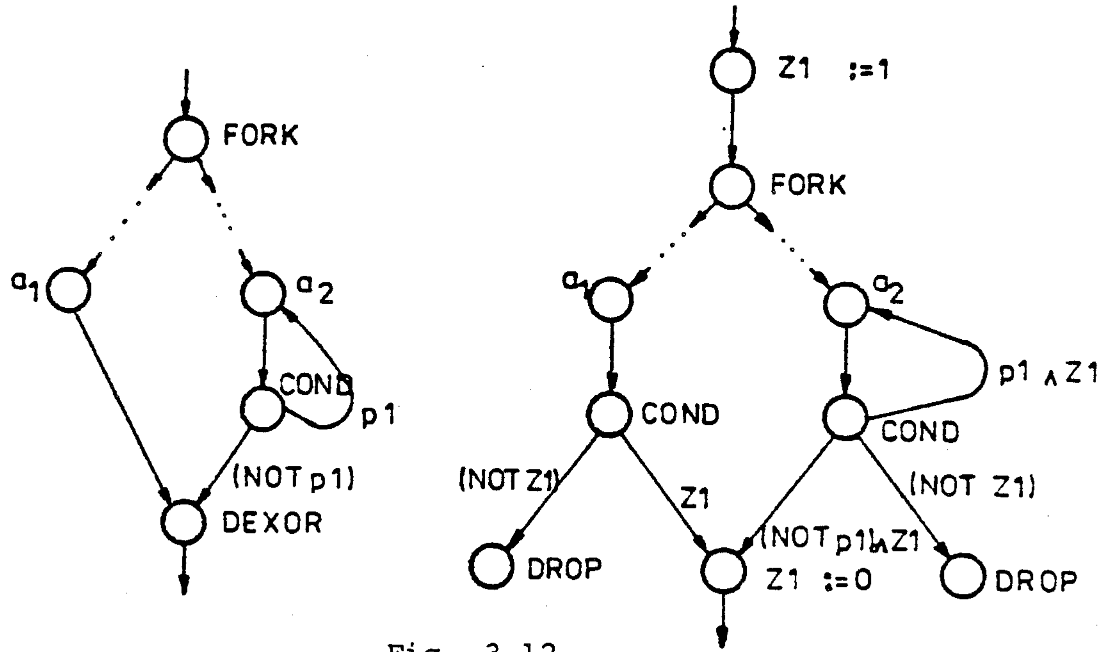


Fig. 3.12

mation of the state, interrupt, data transfer. The systems of this type are applied often in automatic control (for instance in all industrial processes where there is the cooperation of different assembly subprocesses). The many design problems considered by us has shown that the means of ADL are adequate for describing arbitrary parallel processes.*

* On the other hand the same problems requiring theoretical soluton occurs here, as in the case Petri nets E-nets, parallel program schemata and other models of parallelism. Other open problems are related to investigation of different conflicts among variables. Currently possible conflicts in SIM are detected during translation and part of conflicts in FORK is identified during implementation of the control unit.

3.6. Description of automata and Boolean functions

1. With respect to requirements S1 and S6 the standard data descriptions of the traditional logic design methods are also included in ADL. These are: abstract automata flow-tables and Boolean functions tables. These two forms of description are called functional descriptions and can cooperate with other descriptions of the language.

2. Description of both synchronous and asynchronous, Moore and Mealy automata is possible.

Example 3.5

Mealy synchronous automata from Fig. 3.13a can be described with program from Fig. 3.14 and asynchronous Moore automaton from Fig. 3.13b with program from Fig. 3.15.

a)

	x1	x2	x3	x4
1	–	3/1	4/1	2/1
2	4/0	–	–	–
3	6/0	6/1	–	–
4	–	6/0	1/0	5/1
5	–	–	2/1	–
6	3/0	–	2/0	3/1

b)

ab	00	01	11	10	y_1 y_2
1	①	3	2	4	10
2	②	3	②	4	0-
3	1	③	4	–	11
4	1	④	④	④	-1

Fig. 3.13

3. In certain cases it is also convenient and desirable to describe automata as program-graphs. The disadvantage of this type of description is that noncompletely specified output signals (don't cares) cannot be described. The advanatage being the possibility of describing mixed synchronous-asynchronous automata and the convenience of automata description with many unspecified transitions. The automaton from Fig. 3.13a is described using this type of description in Fig. 3.16 and the automaton from Fig. 3.13b in Fig. 3.17. As we see all the nonspecified signals take here the value zero. Applying the assignment operators := and =: one can describe changes occuring at different phases (:= corresponds to phase ϕ_2 and =: to phase ϕ_1). Such description permits also the specification of arbitrary combinational functions describing transitions among states and not only full products of literals as in the traditional flow-table descriptions.

```
(( ADL   C   LOGBLOCK   AUTOMAT1  (X1  X2  X3  X4  Y  CLOCK)
        (INPUT   (X1 (0)) (X2 (0)) (X3 (0)) (X4 (0))   )
        (OUTPUT  (Y (0)) ) )
 (( Y == ( MEALY   SYN   CLOCK   (X1  X2  X3   X4)
          1 (( X2   3  (1))   ( X3   4   (1)) ( X4   2  (1)))
          2 (( X1   4   (0)))
          3 (( X1    6   (0))  ( X2  6  (1)))
          4 (( X2   6   (0))  ( X3   1   (0)) ( X4   5  (1)))
          5 (( X3    2   (1)))
          6 (( X1   3   (0))  ( X3    2   (0)) ( X4   3  (1)))  )) ))
```

Fig. 3.14

```
(( ADL    A    LOGMACRO    AUTOMAT2    (A   B   Y1   Y2))
   ((  (Y2 , Y1)   ==   ( MOORE   ASY    (B  A)

1  (  ((0 0) 1)  ((0 1)  3) ((1  1) 2) (( 1  0)  4) )   (1  0)
2  (  ((0 0)  2)  ((0 1)  3) ((1  1)  2) (( 1  0)  4) )   (0  X)
3  (  ((0 0) 1)  ((0 1)  3) ((1  1)  4)                )   (1  1)
4  (  ((0 0) 1)  ((0 1)  4) ((1  1)  4) (( 1  0)   4) )   (X  1)))
                    ))
```

Fig. 3.15

```
((ADL  C  BLOCK  AUTOMAT1  B (X1 X2 X3 X4 Y)
    (INPUT  (X1 (D)) (X2 (D)) (X3 (D)) (X4 (D)))
    (OUTPUT (Y (D))
    (IDEN (Y1 (Y == 1)) (Y0 (CONTIN))))
((START) B
1 (COND (X2 (Y1 (GO 3))) (X3 (Y1 (GO 4))) (X4 (Y1 (GO 2))))
2 (COND (X1 (Y0 (GO 4))))
3 (COND (X1 (Y0 (GO 6))) (X2 (Y1 (GO 6))))
4 (COND (X2 (Y0 (GO 6))) (X3 (Y0 (GO 1))) (X4 (Y1 (GO 5))))
5 (COND (X3 (Y1 (GO 2))))
6 (COND (X1 (Y0 (GO 3))) (X3 (Y0 (GO 2))) (X4 (Y1 (GO 3))))  ))
```

Fig. 3.16

```
((ADL A LOGBLOCK AUTOMAT2 (A_B Y1 Y2)
   (INPUT (A (D)) (B (D)))
   (OUTPUT (Y1 (D)) (Y2 (D)))
   (IDEN (Y1T (ASY (Y1 == 1))) (Y2T (ASY (Y2 == 1)))))
   ((START) A
     1 Y1T (COND ((AND (¬ A)(¬ B)) (GO 1))
                 ((AND (¬ A) B)    (GO 3))
                 ((AND A B)        (GO 2))
                 ((AND A (¬ B))    (GO 4)) )
     2       (COND ((OR (AND (¬A)(¬ B))(AND A B)) (GO 2))
                 ((AND (¬ A) B)    (GO 3))
                 ((AND A (¬ B))    (GO 4)))
     3 (SIM (Y1T Y2T))
             (COND ((AND (¬ A)(¬ B)) (GO 1))
                 ((AND (¬ A) B)    (GO 3))
                 ((AND A B)        (GO 4)))
     4 Y2T (COND ((AND (¬A)(¬ B)) (GO 1))
                 ((OR A B)         (GO 4)))   ))
```

Fig. 3.17

the scope of this DEXOR with some signal Z1 is necessary. This "producting" corresponds in the description to introducing statements IF and DROP to the description, or COND with predicate Z1 (nonfulfilled Z1 corresponds to DROP). Signal Z1 would be assigned to 1 before parallel disjoint of control and reset to zero in the joining node.

Both of the above methods are very inefficient in realization and lead to not very clear descriptions, which advocates introducing DAND and DEXOR statements to ADL. It is however interesting that the considerations of this and previous paragraphs lead to the conclusion that all parallel statements can be replaced (with accuracy of single pulses) by combinations of parallel statements COND and assignment statements.

7. Arbitrary node with inpointing branches can be also the joining from parallel controls - <u>it transmits each incoming control</u>. Such node will be called DOR - type node. For instance in the program:

```
(FORK (d1 d2 (GO 10))
      (d2 d3 (GO 2 ))
        .
        .
        .
      (d5 d6 (GO 10)))
   ...
10 (X := (X + 1))
   (L := (L - 1))
   ...
```

node with label 10 plays such a role.

Such joining of control means that in the program segment following this node simultaneously two controls are trnsferred. This effect can sometimes be desired; it is however often necessary to prevent these controls from joining or approaching near to the other one. This can be achieved by application of so called "blockades" i.e., the controlling variables are assigned to some values and their state is monitored in some other place of the program.

The language mechanisms described above permit also the description of priority joinings of parallel control, where a certain process is guaranteed priority before the other one (see Appendix A).

8. Possibilities of ADL language application for description of parallel digital systems are very wide and discussed to a small extent in the present report. There exist for instance many ways of cooperation among various parallel statements and their cooperation with statements CONTIN, WAIT, WHILE, IF and the assignment statement. These mechanisms permit describing the systems composed of synchronous and asynchronous blocks interacting in a parallel mode, communicating among themselves by use of signals: calling, return, infor-

4. Because of the methods of synthesis of combinational networks in the description of combinational networks we specify the set of affirmative (uncomplemented) and the set of negative variables. For the uniqueness of the description it is necessary to enumerate the input and the output variables. According to the descriptions of cubes' arrays which is used internally inside the system, the functions are described in the form corresponding to the arrays of cubes.

Example 3.6

The logic function described with the cubes' array:

A	B	C	D	x[1]	x[2]
0	1	X	X	0	1
X	1	0	0	0	X
0	1	0	X	0	1
0	1	X	1	X	1

set of input variables {A,B,C,D}, set of affirmative variables {A,C}, set of negative variables {A,B,D} and set of output variables {X[0], X[1]} is described in ADL as in Fig. 3.18.

```
(X ==    (LOGFUN   (A B C D) (A C) (A B D)
          (((0 1 X X) (0 1))
           ((X 1 0 0) (0 X))
           ((0 1 0 X) (0 1))
           ((0 1 X 1) (X 1)))   ))
```

Fig. 3.18

3.7. Structure description

1. As it results from the requirments from Chapter 2 the language should include the possibility of describing the structure of the designed system and describing it on the several levels of abstraction.

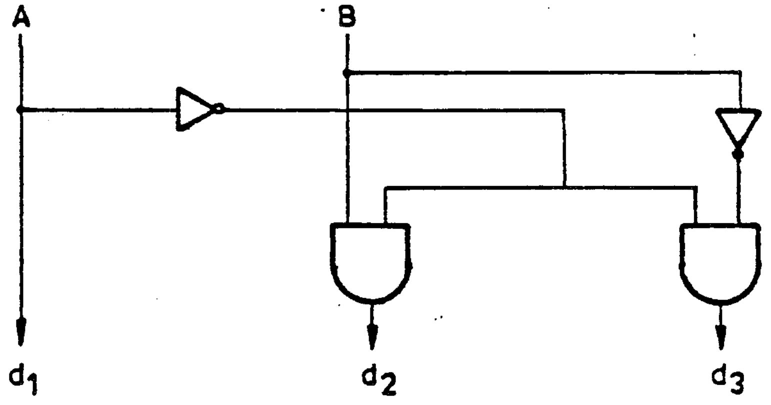
For description of structure serve in ADL type LOGBLOCK and type LOGMACRO subroutines as well as the descriptions inside the scope of operator STR in main program and type BLOCK and MACRO subroutines.

The structure describing statements are nonsequential, they describe components and their connections. The sequence of

1/ (IF A THEN d_1 ELSE (IF B THEN d_2 ELSE d_3))

2/ (IF A THEN (IF B THEN d_1 ELSE d_2) ELSE (IF D THEN d_3 ELSE d_4))

a)



b)

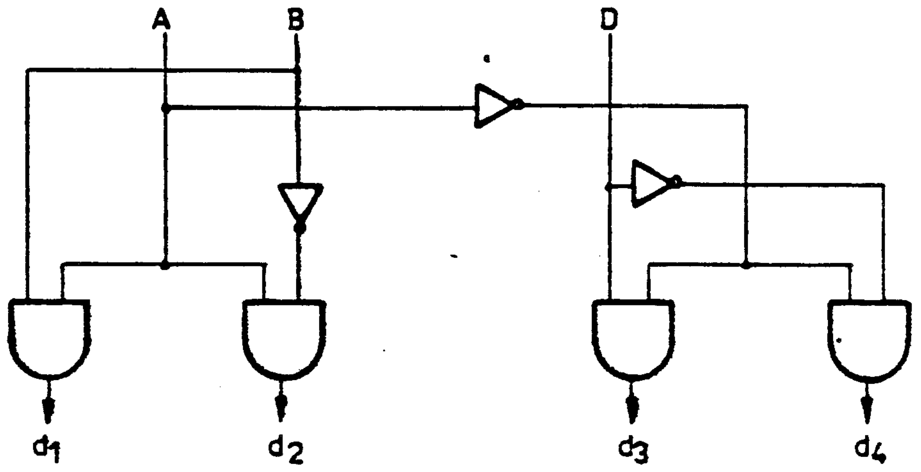


Fig. 3.19

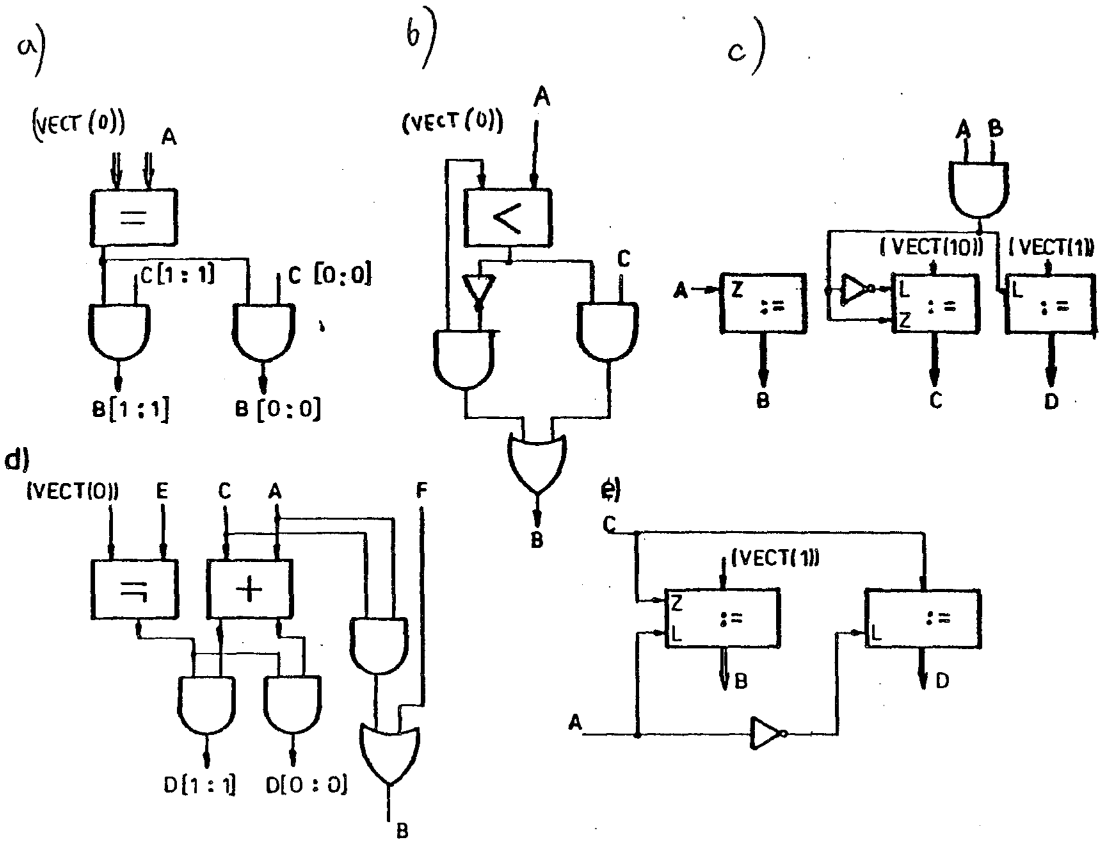

a)
(VECT (0))
A
=
C[1 : 1]
C [0 : 0]
B[1 : 1]
B [0 : 0]
b)
(VECT (0))
A
<
C
B
c)
A B
A
Z
:=
B
(VECT(10))
L
Z
:=
C
(VECT(1))
L
:=
D
d)
(VECT(0))
E
C
A
F
+
D[1 : 1]
D[0 : 0]
B
e)
C
(VECT(1))
Z
L
:=
B
L
:=
D
A

Fig. 3.20 a ÷ e

such statements has no meaning of sequential execution as in the behavior description. The order of statements is arbitrary and related only to the convenience of the designer. The statements have then no counterparts in the program-graph and have counterparts in the structure's graph only.

2. For description of galvanic connections only the operator == can be used in assignment statements, which means direct connection (joining) of the implementations of left and right sides of the statement. Operators :=, =: and :: can stand only in conditional expressions. They mean writing of the right side's value into the respective register, which implements the left side of the statement. They correspond respectively to: back slope, leading slope and high level (asynchronous transfer) of the conditional signal. Meaning of these operators is the same as in the sequential description but the respective assignments are gated not with the clock pulse but with the output of the corresponding prdicate. Generally, the assignment, conditional and other statements used for structure description do not differ syntactically from their counterparts used in the sequential description.

3. Together with the usually used IF statement it is advised also to use the COND statement for structural description. It plays in such a case the role of the CASE statement from some languages. Statement

```
(IF A THEN B)
```

sets variable B to state 1 if variable A is in this state. Statement (IF A THEN B ELSE C) additionally set variable C in state A. Statement (COND (P1 B1)(P2 B2) ... (Pi Bi) ...) sets variables Bi in states Pi. To increase the applicability of the statements of this type they can be nested to arbitrary depth one inside the other.

<u>Examples</u>

1) (IF (A = 0) THEN (B == C)) where B and C are two-bit variables - structure from Fig. 3.20a.

2) (IF (0 < A) THEN (B == C) ELSE (B == A)) - structure from Fig. 3.20b.

3)
```
(IF A THEN (B := 0))
(IF (AND A B) THEN ((D := 1)(C := 0))
               ELSE (C := 2))
```
- structure from Fig. 3.20c where B, C and D are the sort P variables. Z is the zeroing input to registers.

4) Structure corresponding to statement:
```
(COND (A (B == C)) ((E = 0)(D == (C + A)))
      (F (B == 1)))
```
is presented in Fig. 3.20d. A, B, C, E, F - single-bit variables.

5) Structure corresponding to statement:

```
(COND (A (B := 1))(C (B := 0))
      ((NOT A)(D := C)))
```

is presented in Figure 3.20e.

4. Although the above structure-describing mechanisms permit for describing wide classes of systems, they can lead to unnecessarily long and repeating descriptions. With respect to requirements L4, L5, L9 and L14 it was decided to introduce to ADL of two new mechanisms for describing structures:

   1) Subscripted structure variables. The subscripts are here INDEX type variables. The meaning of such variables is the mutual area placement of variables and not their time sequence. These variables, together with the "do" statement permit for easy description of iterative circuits.

   2) Hierarchical description of structures. The macros are defined whose names correspond to black boxes, formal parameters to certain variables (mainly input or output), and bodies to the more detailed description of these units from gates and other macros. Next the names of parameters are called with different values of actual parameters. The actual parameters correspond to the connections among the circuits implementing macros and the other units.

   The values of the subscripts are set up with the operator ::. In the "do" statement's predicate the subscript must stand on the left side of the relational operator and the numerical constant on its right side. Arbitrary structure-describing predicates can occur in the statement's sequence following symbol DO. Let us notice that the memory element must correspond not to each internal variable in the structure description. Such element is needed only in the case of assigning a value with an operator different than ==. This property permits one to assign names to connections, which is useful in the description of structures composed of substructures and simplifies the decomposition as well.

5. The library macros are declared which describe static flip-flops: D, T, JK and SR (latch) and dynamic flip-flops D, T, JK.

   The following statements are also available for structure description:
   - ROM - for the "read only memory" description,
   - DECODE - for describing the units with addressing, for instance multiplexers and demultiplexers,
   - MPX - for the description of multiplexers,
   - DMPX - for the description of demultiplexers,
   - LOGFUN, MOORE, MEALY (explained previously).

   All these statements are rewritten into the respective structures from components at different implementation stages.

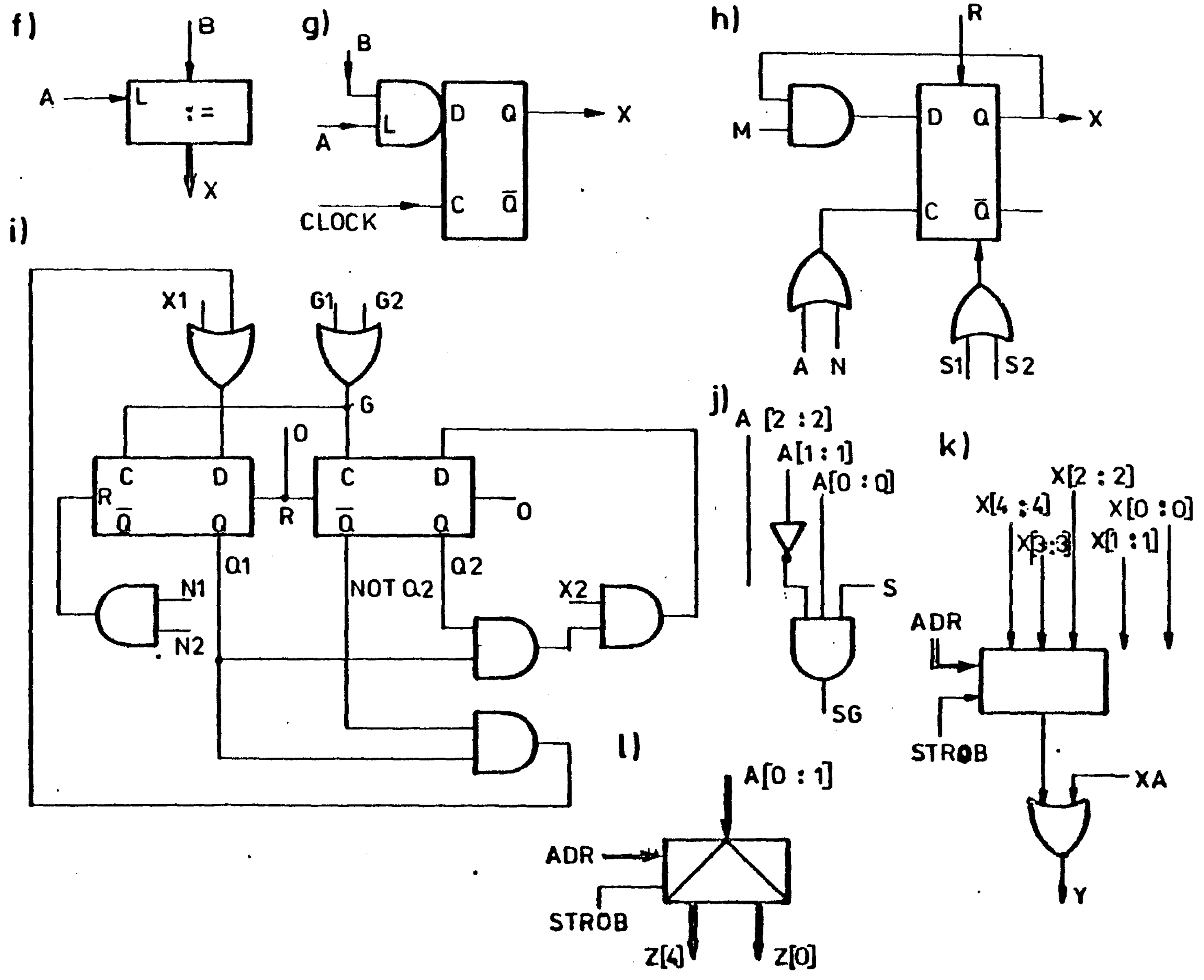

f)
A
B
L
:=
X
g)
B
A
L
D
Q
X
CLOCK
C
Q̄
h)
R
M
D
Q
X
C
Q̄
A
N
S1
S2
i)
X1
G1
G2
G
C
D
R
Q̄
Q
O
R
C
D
Q̄
Q
O
Q1
NOT Q2
Q2
N1
N2
X2
j)
A [2 : 2]
A[1 : 1]
A[0 : 0]
S
SG
k)
X[2 : 2]
X[4 : 4]
X[0 : 0]
X[3 : 3]
X[1 : 1]
ADR
STROB
XA
Y
l)
A[0 : 1]
ADR
STROB
Z[4]
Z[0]

Fig. 3.20 f ÷ l

6. In the discussed above assignment statements the change of the flip-flop's state is done in the moment of the basic clock CLOCK change (respectively - during the phase signals for various MOS technologies). The block schematics of such operation for statement (IF A THEN (X := B)) is presented in Fig. 3.20f and the logic diagram for single bit in Fig. 3.20g. At the clock input of the flip-flop no any logic network can hang (which results from the dynamic MOS technologies). Yet to enable the description of devices from static MOS flip-flops the macros can be used.

Examples

The circuit from Fig. 3.20h can be described as:

```
(D (AND MX)(OR A N) R(OR S1 S2) X())
```

The statements:

```
     (G == (OR G1 G2))
     (D (OR X1 (AND Q1 NOTQ2)) G 0
                  (AND N1 N2) Q1 ())
     (D (AND X2 (AND Q2 Q1)) G 0 0 Q2 NOTQ2)
```

describe the circuit from Fig. 3.20i.

The statement (SG == (AND S(DECODE A 2 K1 5))) has implementation from Fig. 3.20j.

(DECODE A N1 CODE1 N2) means leading of N1 youngest bits of variable A: if i-th from right bit of number N2 in code CODE1 is zero - then the negation of the i-th bit of variable A is taken, if it is one then the uncomplemented value is taken.

The statement:

```
   (Y == (OR XA (MPX ADR STROB (X [ 2 : 4 ] J ))))
```

describes multiplexer from Fig. 3.20k.

The statements:

```
   (((Z [ 0 ])' (Z [ 1 ])' (Z [ 2 ])'
     (Z [ 3 ])' (Z [ 4 ])) ==
     (DMPX ADR STROB (A [ 0 : 1 ]) P K1 10))
or
   (Z == (DMPX ADR STROB (A [ 0 : 1 ]) P K1 10))
```

describe demultiplexer from Fig. 3.20L. In the last case Z is declared as the array of variables: (Z [0 : 4])(P K1 2).

Example 3.7

Iterative circuit

Iterative circuit from Fig. 3.21 can be described in ADL as the LOGBLOCK type subroutine from Fig. 3.22.

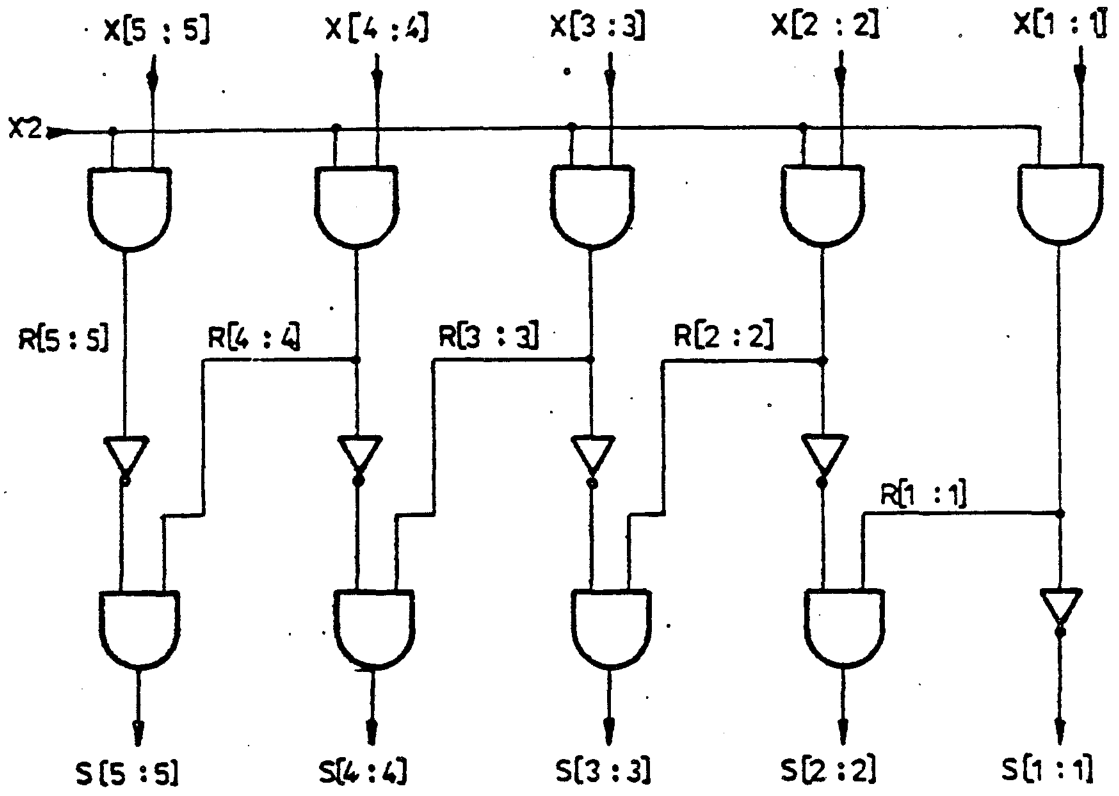


Fig. 3.21

In this description I is the type INDEX subscript variable. No counter corresponds then to it in the structure. Assignment statements (I :: 2), (I := (I + 1)) mean only specification of some substructure. The subscripted variables R and S are of INTERN type. KX means the nonspecified code.

Example 3.8

Parallel adder description from repeating components

The circuit of the half adder HA from Fig. 3.23a can be described with LOGMACRO HA in the program from Fig. 3.24. The elementary adder with carry from Fig. 3.23b can be described as the sequence of statements in loop of LOGMACRO FA (see program from Fig. 3.24).

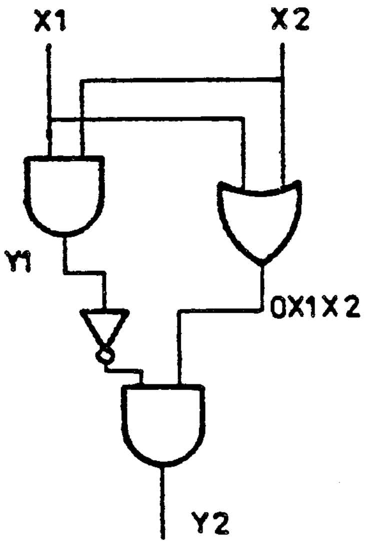
The main program describes the four-bit parallel adder. Let us notice that number 5 is here the actual parameter of subroutine FA, corresponding to the formal parameter F from the end's predicate of the loop. Some actual parameters of calling FA are marked in Fig. 3.22 by the corresponding formal parameters. Implementation of the entire adder is presented in Fig. 3.25. Let us notice the way of carry signal description, where actual parameter X corresponds to two formal parameters X3 and X1.

7. According to the principle of the multilevel description of systems there exists the possibility fo digital structures'

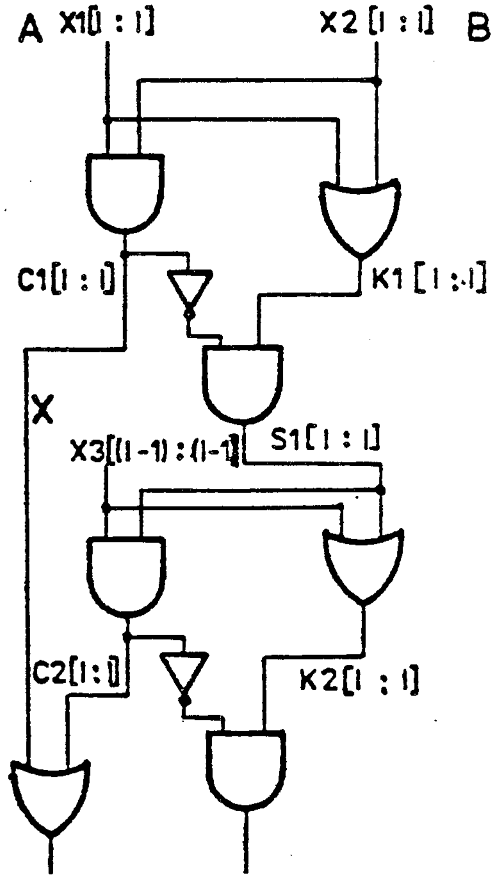
```
(( ADL   B   LOGBLOCK   ITERATIVE_CIRCUIT_1   ( X  X2  S)
   ( INPUT  (X  (P  KX  5))  (X2  (D)))
   ( INTERN  (R (P  KX  5))  (S  (P  KX  5)))
   ( INDEX  I ))
 ( (( R  [ 1  :  1 ] ) == ( AND  X2  (X1 [ 1 : 1 ] )))
  (( S  [ 1  :  1 ] ) == ( NOT  (R [ 1  :  1 ]  )))
   ( I  ::  2 )
   ( WHILE  ( I  <  6 )  DO  (
      (( R  [  I  ] )  == (AND  X2  (X1  [ I  :  I ] )))
      (( S  [  I  ] )  == (AND  (R  [ (I - 1) : (I - 1)
                                  (NOT  (R [ I  :  I ] ))))
      (I  ::  (I  +  1))   ))   ))
```

Fig. 3. 22.

a)



b)



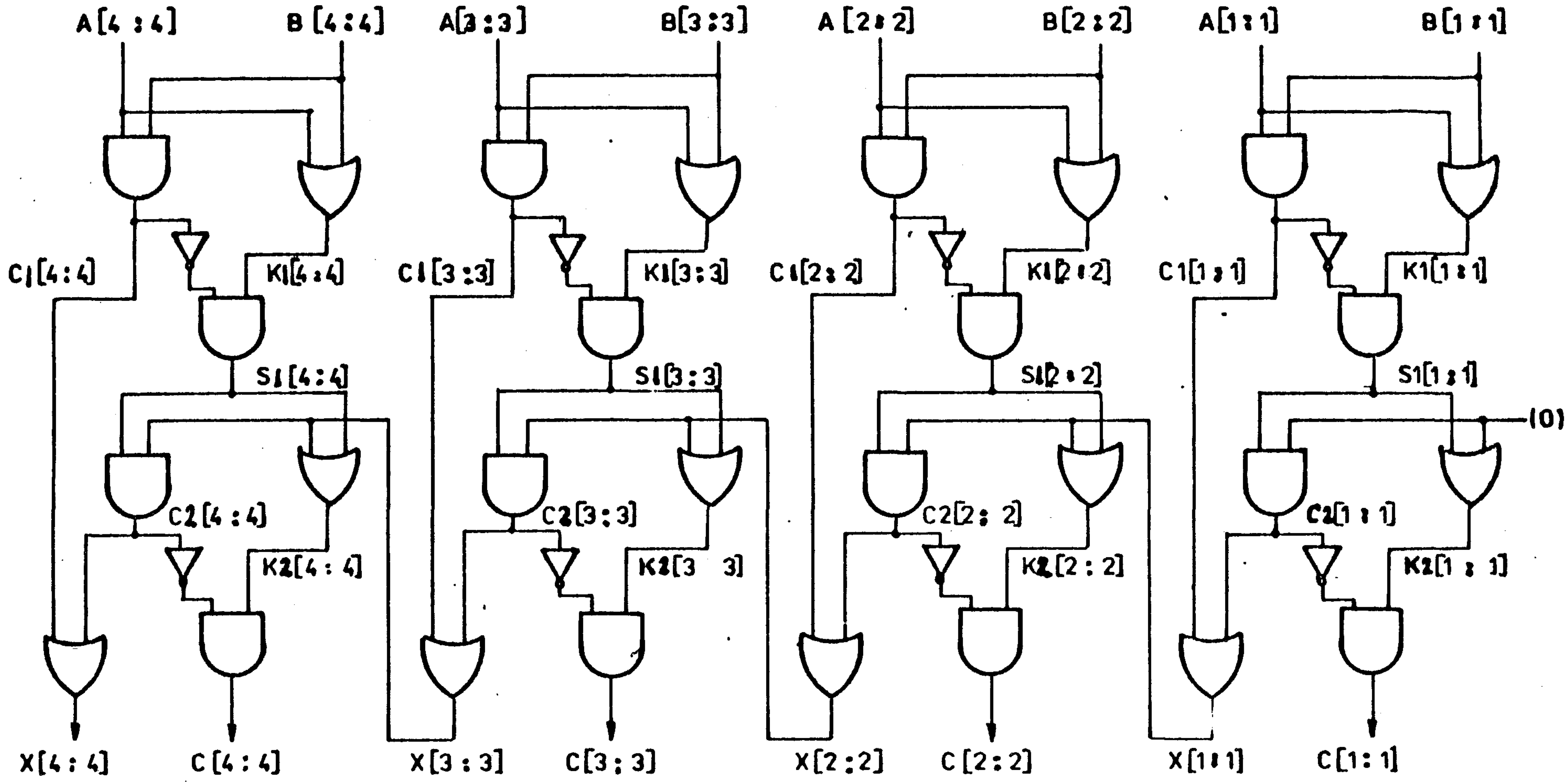
(PARALLEL ADDER - STRUCTURAL DESCRIPTION)

```
4(                                     1
1  1

(((ADL  A ADDER
123
  (INPUT (A  (P K1  4))  (B  (P K1  4)))
  4        5  6         65 5   5         654
  (INTERN  (K1(P K1  4))(K2(P K1  4))(C1(P K1  4))(S1(P K1  4))
  4          5  5         655   5         555   5        555   5         65
              (C2(P K1  4))(X(P K1  4)))
               5  6        555 6        654
  (OUTPUT (C  (P K1  5)))
  4         5  6        654
   (SUBR (LOGMACRO C HA  (X1  X2  Y1  Y2  OX1X2))
    4      5                6                    65
          (LOGMACRO B FA  (X1  X2  X3  Y1  Y2  K1  K2  C1  S1  C2  F)))
           5                6                                        654
  (INDEX  I )  )
  4          4  3
  (  (START)  A
  3  4       4
   (STR
   4
   (FA(A  B  X  X  C  K1  K2  C1  S1  C2  5)))
   5  5                                654
   (STOPADL)  ))
   4         4  32           A   B  X  X   C  K1  K2  C1  S1  C2  5
 ((ADL  B LOGMACRO FA  (X1  X2  X3  Y1  Y2  K1  K2  C1  S1  C2  F)
 23                    4                                           4
     (INDEX  I))
      4       43
   ((HA((X1  [  1  :  1  ])(X2  [  1  :  1  ])(C1  [  1  :  1  ])(S1  [  1  :  1  ])
   34   55              55                66                  56                 5
                (  1  [  1  :  1  ]))))
                5                654
   (HA (0(S1  [  1  :  1  ])(C2  [  1  :  1  ])(Y2  [  1  :  1  ])(K2  [  1  :  1  ]))))
   4   5  5               55                66                  56                 654
   ((Y1  [  1  :  1  ])==(0((C1  [  1  :  1  ])(C2  [  1  :  1  ]))))
   45                  5    5    5                66                554
   (I  :=  2)
   4        4
   (WHILE(I  <  F)  D(
   4       5        5   5
     (HA((X1  [  I  :  I  ])(X2  [  I  :  I  ])(C1  [  I  :  I  ])
      -    78                88                85                 8
                (S1  [  I  :  I  ])(K1  [  I  :  I  ]))))
                 8                 88                 876
     (HA((X3  [(I  -  1):(I  -  1)])(S1  [  I  :  I  ])(C2  [  I  :  I  ])
      -    78     9       9  -       9  88               88                 8
                (Y2  [  I  :  I  ])(K2  [  I  :  I  ]))))
                 8                 88                 876
     ((Y1  [  I  :  I  ])==(OR(C1  [  I  :  I  ])(C2  [  I  :  I  ])))
     67                    7   7   8                98                    876
     (I  ::(I  +  1))  ))  ))
     6     7       76  54  32
   ((AND C LOGICAL C [illegible] (X1  X2  Y1  Y2  OX1X2)
   23                            4                    4
      )

   ((Y1  ==  (AND  X1  X2))
   34        5         54
    (OX1X2  ==  (OR  X1  X2))
    4           5          54
    (Y2  ==  (AND  (  NOT  Y1)  OX1X2))  )))
    4        5     5       5        54  321
END
```

Fig. 3.24



A[4 : 4]
B [4 : 4]
C1[4 : 4]
K1[4 : 4]
S1[4 : 4]
C2[4 : 4]
K2[4 : 4]
X[4 : 4]
C[4 : 4]
A[3 : 3]
B[3 : 3]
C1[3 : 3]
K1[3 : 3]
S1[3 : 3]
C2[3 : 3]
K2[3 3]
X[3 : 3]
C[3 ; 3]
A [2 : 2]
B[2 : 2]
C1[2 : 2]
K1[2 : 2]
S1[2 : 2]
C2[2 : 2]
K2[2 : 2]
X[2 : 2]
C [2 : 2]
A[1 : 1]
B[1 : 1]
C1[1 : 1]
K1[1 : 1]
S1[1 : 1]
(0)
C2[1 : 1]
K2[1 : 1]
X[1 : 1]
C[1 : 1]

Fig. 3.25

description strictly referred to the MOS technology. It is related to the desire of enabling the reader to create the special or the highly optimized descriptions, which couldn't be obtained through the implementation of the higher level descriptions. ADL includes then in the structural description (as well as in some other statements of the main program or type BLOCK and LOGBLOCK subroutines) the logical exrpessions which take into account the phases.

To enable this property without referring to the necessity of describing single transistors two methods of description are used. In the first of them the phase stands as the first of th arguments of operator PHASE. The second argument being the expression, preceding the transmission transistor being supplied with this phase. This method is used for instance to describe dynamic registers. In the second method (applied in the ratioless circuits) the phase is the only argument of the PHASE statement, being in turn an argument of the logic operator supplied with this phase.

8. In the case of circuit description from negative gates each AND, OR, NAND or NOR component of such gate should be described separately. Creating of the negative gates is next done automatically in the process of implementation. The typical negative gates can be also declared as LOGMACRO and use in the calls with actual parameters.

   In the circuit including the negative gates most of AND and OR gates lead its outputs to single gates. Also some other case can be implemented without changing the gates' structure, but generally the equivalent network transformations for target technology must be executed.

9. Certain class of MOS circuits (for instance the bridge circuit) cannot be described by direct mapping of their structures with an aid of the already introduced ADL tools (their function can be described equivalently - by specifying equivalent Boolean function - this is utilized for instance in simulation). To describe networks of this type it was necessary to introduce descriptions of single transistors (statement MOS, FET, RESISTOR). The statement COLUMN was introduced to give the designer a tool for transmitting information to symbolic layout placing programs. The statement informs one that the transistors being its arguments are placed in one column. By describing the MOS circuits the user must remember (with respect to the proper cooperation of the subunits of the entire system described with different methods) that positive logic is assumed in ADL and DIADES.

10. ROMs and PLAs are often used in VLSI. By ROM we will understand the single array of NANDs (in positive logic). A description of ROM should be done with use of the ROM statement. By PLA we will understand two arrays of NANDs (i.e., two ROMs) linked in such a way that they realize the alternative normal form of multioutput Boolean function [Carr76].

The two-level multioutput functions can also be realized with ROM and additional NAND gates which separately realize sums of the implicants (in PLA the sums are realized in a separate array). This method is often used in the practical designs.

Example 3.9

The circuit from Fig. 3.26 can be described as follows

```
     (PHASE F2 (NOR (PHASE F1A) C
          (PHASE F1 (NAND (OR G
                            (AND I J ))
                        (OR H L ))) ))
```

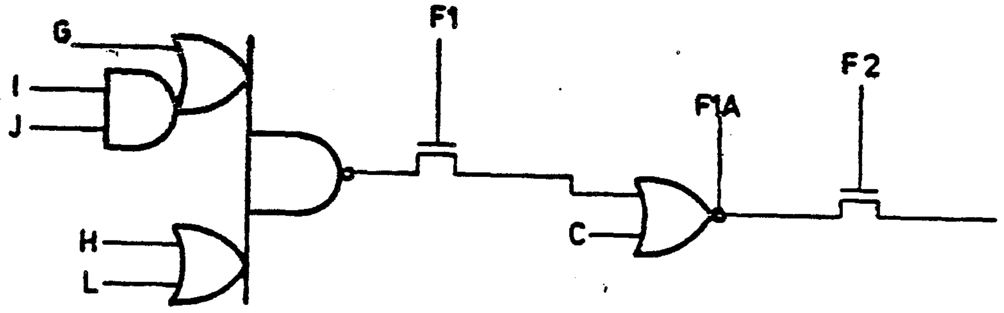


Fig. 3.26

This example present the method of describing gates in the ratioless circuit (statement (PHASE F1A) arising as the first argument of the respective operator), the method of describing transmission transistors among gates (statements (PHASE F2 ...) and (PHASE F1 ...)). The description method for the negative gate leading its output to the , run with F1 phase, transmission transistor is also shown.

Example 3.10

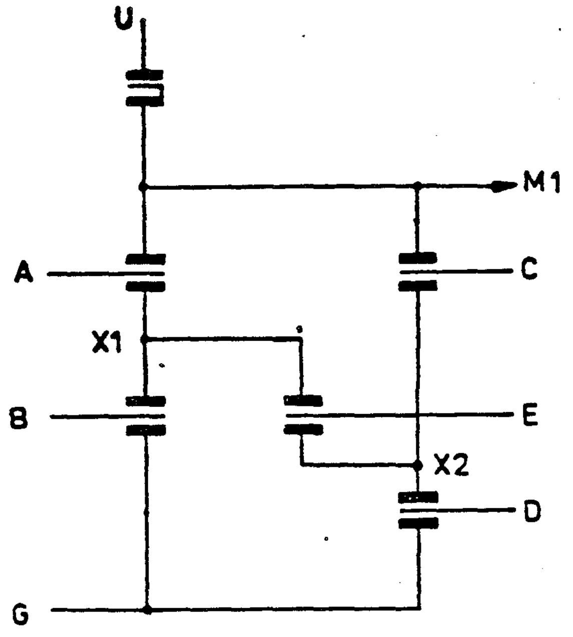


Fig. 3.27

The bridge circuit from Fig. 3.27 can be described as follows:

```
(M1 == (MOS (A B C D E) (M1)
  (COLUMN
     (RESISTOR   M1)
     (FET    A    M1    X1)
     (FET    B    X1    G))
  (COLUMN
     (FET    E    X1    X2))
  (COLUMN
     (FET    C    M1    X2)
     (FET    D    X2    G))  )
```

<u>Example 3.11</u>

Diagram from Fig. 3.28a can be described (positive logic!) as follows:

```
(X  ==  (NOR  A  (OR  B  C)))
(Y  ==  (NOR  D  (OR  B  C)))
```

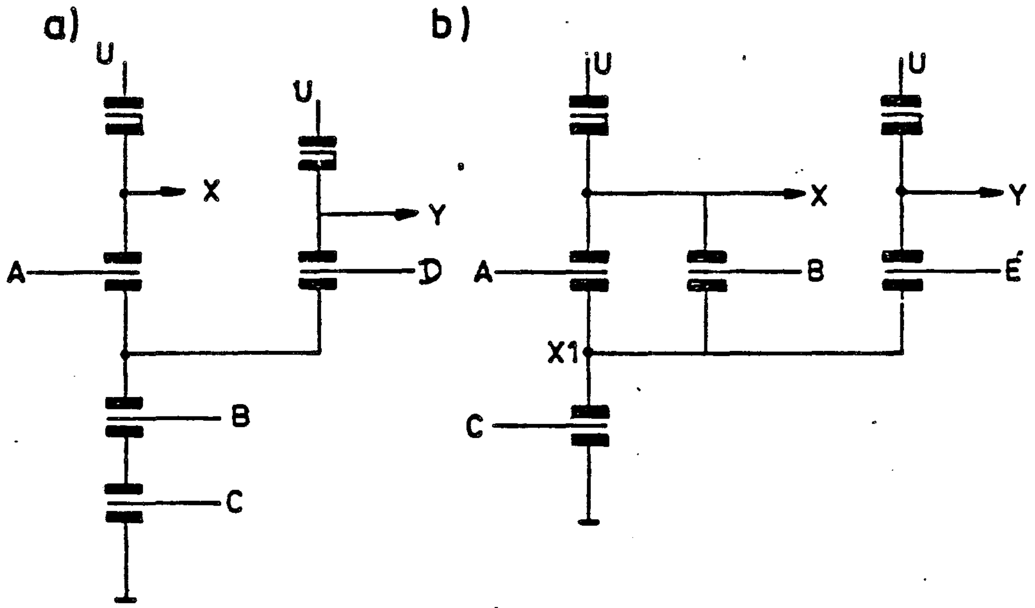


Fig. 3.28

Let us notice that similar diagram from Fig. 3.28b cannot be described in such a way and the MOS statement is necessary to apply, because of the necessity to describe separately each transistor.

```
(XX == (MOS  (A B C E) (X Y)
       (COLUMN (RESISTOR X)(FET A X X1)(FET C X1 G))
       (COLUMN (FET C X1 G))
       (COLUMN (RESISTOR Y)(FET E Y X1))))
(X == (XX [ 0 ]))
(Y == (XX [ 1 ]))
```

Example 3.12

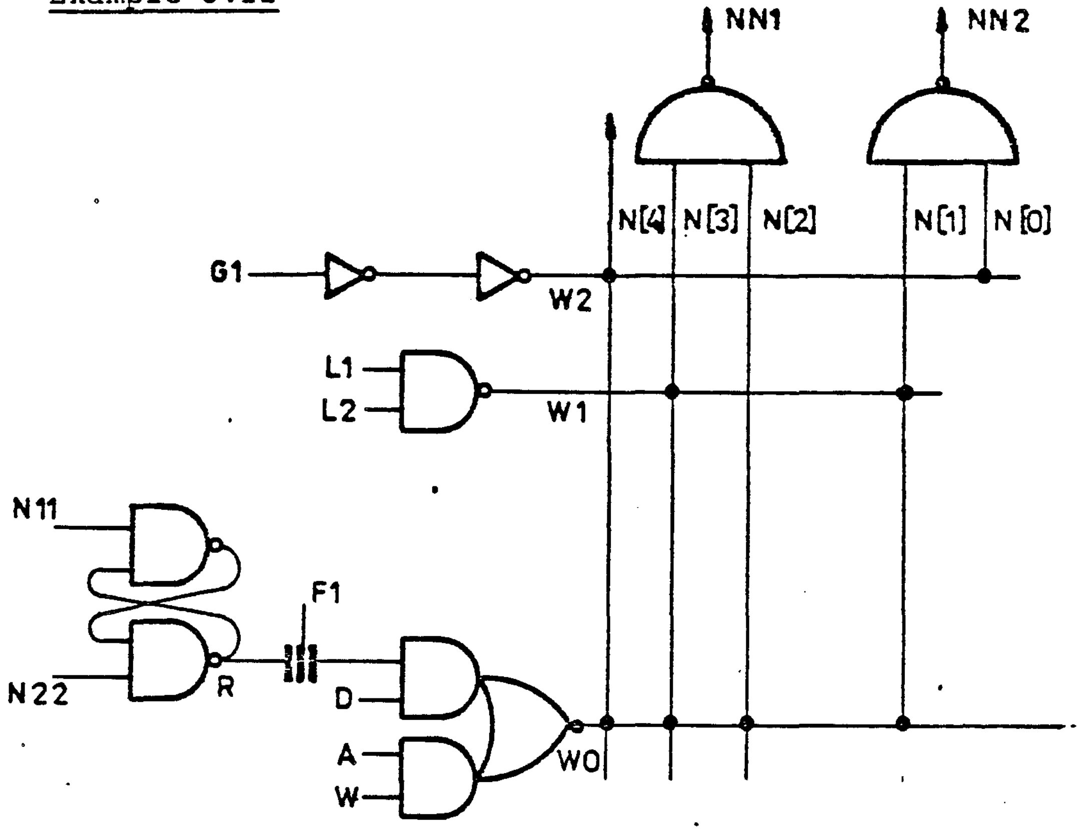


Fig. 3.29

The circuit from Fig. 3.29 can be described as follows:

```
   ...
(N == (ROM (WO W1 W2)
           (W2)(WO W1)(WO)(WO W1)(WO W2)))
(NN1 == (NAND (N [ 2 : 2 ])(N [ 3 : 3 ] )))
(NN2 == (NAND (N [ 1 : 1 ])(N [ 0 : 0 ] )))
(W2 == (  (  G1)))
(W1 == (NAND L1 L2))
(WO == (NOR (AND (PHASE F1 R) D) (AND A W )))
(R == (NAND N22 (NAND N11 R)))
```

In Fig. 3.29 the schematic diagram of ROM is shown. The logically equivalent description of ROM is:

```
((N [ 0 ]) == (NOT W2))
((N [ 1 ]) == (NAND WO W1))
((N [ 2 ]) == (NOT WO))
((N [ 3 ]) == (NAND WO W1))
((N [ 4 ]) == (NAND WO W2))
```

As we see the implicant WO•W1 is here taken twice in sums NN1 and NN2 which corresponds to the second method of PLA realization.

11. Statements MOS, ROM, DECODE, LOGFUN, MPX, MOORE and MEALY can be applied both in structure description and in behavior description in the same way as logic operators in expressions, for instance

```
(R =: (MOS (A B)....))
(X =: (AND STROBE(ROM....)))
(A =: (AND X (LOGFUN....)))
(B =: ((MOORE....) + (A-B)))
(IF (A < B) THEN (R =: (DECODE....)))
```

5.1. Simulation control language INADL

Description of simulated devices includes the hardware description of the device (both behavioral and structural) in ADL82 (Automata Description Language) and the description of the simulation conditions in the language INADL (INputs to ADL). INADL permits:

- assigning of initial values for variables,
- specifying input sequences of variables,
- defining the list of printouts,
- defining the type and frequency of printouts,
- setting the values of the parameters which control the simulation process.

5.1.1. Syntax of INADL

```
<symbol> ::= <letter> | <digit> | <delimeter> | <space>;
<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P |
             Q | R | S | T | U | V | W | X | Y | Z;
<digit>  ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9;
<delimeter> ::=  ( | ) | [ | ] | : | $ | * | , | =;
<space> ::=   (rank)
<name> ::= <letter> | <name><letter> | <name><digit>;
<number> ::= <natural number> | <octal number>:
<octal number> ::= <natural number> B;
<variable> ::= <simple variable> | <subscripted variable>;
<Simple variable> ::= <name>;
<index> ::= <variable> | <number>;
<list of variables> ::= <variable> | <table> | <list of variables>,
                        <variable> | <list of variables>, <table>;
<table> ::= <variable> [<number> : <number>];
<comment> ::= $<text> ;
<text> ::= <symbol> | <text><symbol> ;
<data for simulation> ::= <statement of beginning of data><list of
                          statements><statement of end of data>
                          <numerical data> ;
<statement of beginning of data> ::= ADLDATA ;
<statement of end of data> ::= END;
<list of statements> ::= <statement> | <list of statements>
                         <statement> ;
<statement> ::= <LOW statement> | <HIGH statement> | <CONST statement> |
                <INPUT statement> | <extended INPUT statement> |
                <logic signal> | <parallel signal> | <value assignment
                statement> | <print statement> | <frequency of print
                statement> | <RESTART statement> | <TIMEASYNCH statement>
                | <time-limiting statement> ;
<LOW statement> ::= LOW (<list of variables>);
<HIGH statement> ::= HIGH (<list of variables>);
<CONST statement> ::= CONST (<CONST list>);
<CONST list> ::= <value> <list of variable> | <CONST list>, <value>
                 <list of variables> ;
```

```
<value> ::= <number>;
<INPUT statement> ::= INPUT (<list of variables>);
<extended INPUT statement> ::= INPUT (<list of time intervals>{*})
                                <list of variables> ;
<list of time intervals> ::= <number> | <list of time intervals>,
                             <number> ;
<logic signal> ::= <variable>(<list of time intervals>{*});
<parallel signal> ::= <variable>(<list of changes>{*});
<list of changes> ::= <value> <time interval> | <list of changes>,
                      <value> {<time interval>} ;
<time interval> ::= <value> ;
<value assignment statement> ::= <variable>=<initial value>,{<final
                                 value>,<step>} ;
<initial value> ::= <number> ;
<final value> ::= <number> ;
<step> ::= <number> ;
<numerical data> ::= <number> <number> | <numerical data> <number> ;
<print statement> ::= PRINT(<print list>) ;
<print list> ::= {<format identifier>},<list of variables> | <print
                 list>,<format identifier>,<list of variables> ;
<frequency of print statement> ::= PRINTSTEP(<list of time intervals>+)
<format identifier> ::= 0 | 1 | 2 ;
<RESTART statement> ::= RESTART(<list of time intervals>+) ;
<TIMESYNCH statements> ::= TIMESYNCH   <number> ;
<time-limiting statement> ::= TIME   <number> .
```

means space (blank). {<terminal or nontermial symbol>} means that this symbol is an option.

5.1.2. Semantics of INADL

Statements which specify input signals

Statement ADLDATA

ADLDATA   1)

begins the sequence of data for simulation. All possible text which precede work ADLDATA are printed on output file, without analyzing their meaning.

Statement LOW

LOW(var$_1$,var$_2$,...,var$_n$)   2)

assigns value "zero" to variables var$_1$,var$_2$,...,var$_n$.

Statement HIGH

HIGH (var$_1$,var$_2$,...,var$_n$)   3)

assigns value "one" to variables var$_1$,var$_2$,...,var$_n$.

Statements HIGH and LOW serve mainly for assignment of initial values for logic (two-valuable) variables.

Statement CONST

CONST (m_1 Lv_1{,m_i Lv_i}) 4)
serves to assign initial values to variables. It ensures that value m_1 is assigned to all variables from Lv_1 list, value m_2 is assigned to variables from Lv_2, etc. The symbols in { } parantheses can be iterated an arbitrary number of times or can be omitted.

Example
```
    CONST (125 R1, R2, R[5:20], 177B R[1:4], R[21:25])
    LOW (A, B, POWER)
    HIGH (CLOCK, START1)
```

In this case the variables R1, R2, R[5], R[6],...,R[20] obtain value 125. Value 128 (177B in octal) is assigned to words from 1 to 4 and to words from 21 to 25 of subscripted variable R. Value 0 is assigned to variables A,B and POWER. Value 1 is assinged to variables CLOCK and START1.

Statement Var

Var = m_1, m_2, m_3 5)
ensures assignment to variable Var the subsequent values from m_1 to m_2 with step m_3 before each beginning of the simulation process. This statement is utilized for declaring stimuli for the simulation processes for which the timing analysis has not been required. If numbers m_2 and m_3 are omitted the constant m_1 value is assigned.

Statement

Var(m_1 t_1, m_2 t_2,...,m_n t_n {*}) 6)
serves for declaration of stimuli. It describes input variable var (of parallel type) which has value m_1 for time t_1, next value m_2 for time t_2, etc. Placing symbol * on the end of the list causes cyclical repetition of changes from the list. For the nonperiodic signals the last declaration of time t_n can be omitted which means that the value m_n assigned as the last one will remain constant until the end of the simulation process.

Statement

Var(t_1,t_2,...,t_n {*}) 7)
serves for declaration of the waveforms of logic variables. The subsequent numbers t_1,t_2,...,t_n declare the length of time intervals after which the Var variable is complemented (from 0 to 1 and from 1 to 0). The initial value of Var is specified with HIGH or LOW statement.

Example

DAN and RX1 are parallel variables and ST is a binary variable.

```
    DAN(5 6, 4 2, 3 1, 0 2,*)
    RX1(10 2, 24 7, 8)
    HIGH(ST)
    ST(2,*)
```

Statement INPUT

INPUT $(var_1,\ldots,var_n)$ 8)
serves to declare list of variables var_1, $var_2,\ldots,var_n$, which values are read from cards before beginning each simulation process for which the timing analysis has not been declared. Data on cards are written in free format (A format). As a first number in each set of the stimuli stands the number of variables, next the values of the variables follow. These data are preceded by the END card in the deck.

Statement

END 9)
ends the sequence of data for simulation.

Example

```
ADLDATA
LB = 5
MN = 4,15,3
INPUT (D, NZ,01)
.
.
.
END
3   7   9  1
3   2  12  4
3  13   1  7
.
.
.
```

For this sequence the variables will have the following values in the subsequent runs of simulation.

No. of run	LB	NN	D	NZ	C1
1	5	4	7	9	1
2	5	7	2	12	4
3	5	10	13	1	7

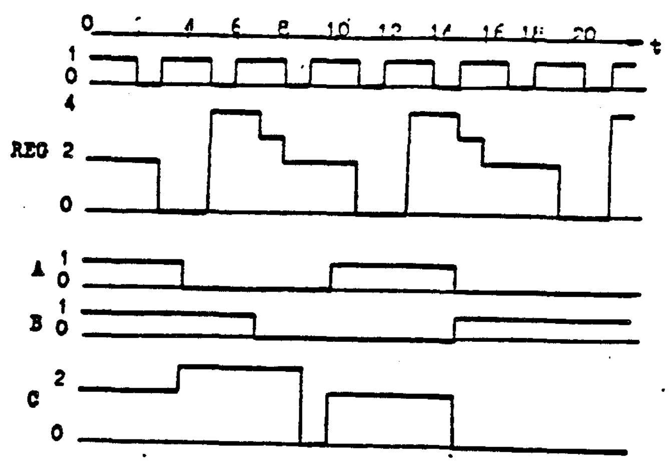
Statement

INPUT$(t_1,t_2,\ldots,t_k\{*\})$ $var_1,var_2,\ldots,var_n$ 10)
is an extension of statement (8) and have application in the simulation with timing analysis. It enables declaring complex signals, which are hard to describe with statements NOS. 6) or 7). The numbers $t_1,t_2,\ldots,t_k$ correspond to the time intervals, after passing of which the subsequent values of variables $var_1,var_2,\ldots,var_n$ are read from the input. On the list of time intervals the symbol of iteration * can stand as the last one. In the data card sequence only one INPUT statement can be applied.

Example

The signals from Fig. 5.1a are described as in Fig. 5.1b (A and B are the binary variables).

a)



b)

```
HIGH(CLO,A,B) C = 2
REG(2 3, 0 2, 4 2, 3 1,*)
CLO(2, 1,*)
INPUT(4,7,9,10,15)A,B,C
...
END

3     0     1     3

3     0     0     3

3     0     0     0

3     1     0     2

3     0     1     0
```

Fig. 5.1

In this example application of an INPUT statement is not obligatory, the equivalent description is:

```
HIGH (CLO, A,B)   CLO(2,1,*)
REG (2 3, 0 2, 4 2, 3 1,*)  B(7,8)
A (4,6,5) C(2 4, 3 5, 0 1, 2 5, 0)
```

The print-controlling statements

Statement PRINT

PRINT $(f_1, Lv_1, f_2, \ldots, f_n, Lv_n)$ 11)
serves to specify the format and the list of variables for print. The variables from list Lv_1 are printed according to format f_1, variables from list Lv_2 according to format f_2, etc. The order of placing names on lists corresponds to the sequence in which the results are printed. Specified are the following formats:

0 - binary print wit shifted one,
1 - binary print,
2 - octal print.

The format is specified globally, i.e., format specified in one print statement and not specified in the other one is still valid. The standard format has number 0. In the print as many bits of the variable are taken into account as was declared in the description of the simulated system.

Print statement can be also utilized for print of the contents of some tables of data base. In such a case in the list of print statement the names of tables are placed with * prefix. For instance the statement

```
   PRINT (*TABNAZW, *LWART)
```

induces printing the contents of tables TABNAZW and LWART according to the formats specified in the simulator. The number of elements printed by the given tables dependes on the values of subscripts which specify the number of reservated or occupied words. The indexes are set up by the simulator. In such a way the following tables can be printed: TABNAZW, LWART, ANLIST, NALIST, COP, TW, WYW, NZM. To print the contents of the tables with data for simulation the name * DANE must be given.

The tables are printed when the print statement is recognized. The tables would then not regard the data introduced in the next statements. The statement to print the data tables should be placed just before the END statement.

Simulation controlling statements

Statement TIME

```
   TIME m                           12)
```
specifies m simulation time's units as a time of the simulation process.

Statement TIMEASYNCH

```
   TIMEASYNCH  m                    13)
```
specifies maximal number of m asynchronous complements of variables' values in the simulation time's unit.

Statement FREQ

```
   FREQ  m                          14)
```
specifies the frequency of calculating the system's state during single pulse of the basic clock CLOCK. The standard declaration is CLOCK(1,*) which corresponds to the synchronous simulation. Declaration FREQ 4 means CLOCK(4,*) i.e., the state of the system is calculated four times during 1/2 of the clock period, and the asynchronous simulation is applied.

Statement RESTART

RESTART ($t_1,\ldots,t_n${*})   15)

restarts the simulation process. As the result of its execution the initial node of the program graph is written into the stack of active nodes in moments $t_1,\ldots,t_n$. The values $t_1,\ldots,t_n$ describe the time intervals among the subsequent repetitions of starts of simulation.

The following data sequence is expected for simulation:

```
ADLDATA
<list of statements>
END
{<numerical data>}
```

The numerical data refer to the INPUT statement. All statements can be printed starting from an arbitrary column in the card (there is no format limitation). Excluding the directives ADLDATA and END they can stand also in an arbitrary order. Each statement can be continued into the next lines, the names cannot be however separated i.e., cannot include a space and cannot be split inot two lines.

Symbol $ is the beginning of a comment. It can be placed in an arbitrary column on the card - the text from $ to the end of the card is treated as a comment.

5.2. SIMADL simulator

It is assumed tha SIMADL is use for checking the variants of design, when the device under design is described in the source language ADL and the detailed logic implementation has not yet been specified. The input to SIMADL program is the description in the internal list language GRAF, being the result of ADL compilation by TAG compiler. The syntactic correctness verification and error diagnostics of the ADL program is done by TAG, as well as other initial optimizing transformations. Thanks to existence of TAG the simulator program is simpler and (because of description optimization done by TAG) - the memory of device's description decreases. The user of SIMADL deals only with ADL description, so SIMADL will be called the ADL simulator, however it should be remembered that it requires at first creating of the respective disc file by TAG compiler. The expressions of the simulation language are represented as a data base in the form of tables and lists (interpretive simulation). ADL has <u>parallel statements</u>, it is then obigatory to present the device's control as oriented labeled multigraph and it requires special method of evaluating expression while simulation. The interpretive method of simulation is then much more convenient than the compilation approach in this case.

SIMADL is a <u>binary simulator</u> - the parallel values of variables are treated as compositions (concatenations) of binary (logic) signals. Two modes of operation can be declared for SIMADL: it can work either as synchronous or as asynchronous simulator. The user can declare arbitrary ratio of the unit of simulation time to the basic clock's period. SIMADL reads the following GRAF lists from KIMFOR file: DECSET, COPLISSET, NALISSET, PLISSET, ANLISSET, STRUCTLISSET. While reading the lists NALISSET, PLISSET and STRUCTLISSET the assignment statements, the predicates and the structure descriptions are respectively translated into reverse polish notation. Such notation makes the expressions' values evaluation simple. The simulation process consists in a step by step transmission of the control points through the parallel program graph of control. The descriptions of the nodes actually activated are investigated. For the conditional nodes of the graph the node to which the control point (control) is transmitted depends on fulfillment or not fulfillment of the respective predicate corresponding to that node. In the parallel nodes (control dividing nodes) the control is simultaneously transmitted to more than one branch of the program graph. In joining nodes the joins control coming from parallel branches. The controls coming to such nodes are - dependening on the node's type - cancelled, transmitted further, or reduced to single control. While transmitting of the control points through the graph the assignment statements are executed which correspond to the activated nodes. The time relations are realized through distinction of synchronous and asynchronous transitions among nodes and by taking into account the types of the assignments. Three main stages of SYMADL execution can be distinguished:

1) reading of GRAF lists and creation of respective tables
2) reading of simulation data in language INADL
3) simulation.

The GRAF language lists are read by SIMADL using the respective subroutines which utilize character (lexical) text analysis. Reading of the simulation data is done by DANESYM subroutine, and also utilizes the character text analysis. In this case however also the syntactic analysis is performed as well as the error diagnostics.

When data reading is completed the initial node of the program graph is inserted into the vector of active nodes, which initializes the simulation process. The simulation have two nodes, which are related to one of the methods of the simulated system description.

1) Only the description of the structure is specified. In such a case two nodes are distinguished - the initial and the final one. Simulation consists of calculation of the entire device structure in one step, next the results are printed, new values of stimulating signals assigned and the results are calculated repeatedly. The simultion is halted after so many structural calculations as it was declared in the TIME statement. In such cases the TIME statement specifies the number of repetitions rather then the time of simulation.

2) The description of the device takes into account the timing. In such a case the devices control is presented as the graph with certain transistions among nodes and descriptions of statements subordinated to the particular nodes. Simulation of such description is based on the determination of actual active nodes WWA and their successors, i.e., future activity nodes.

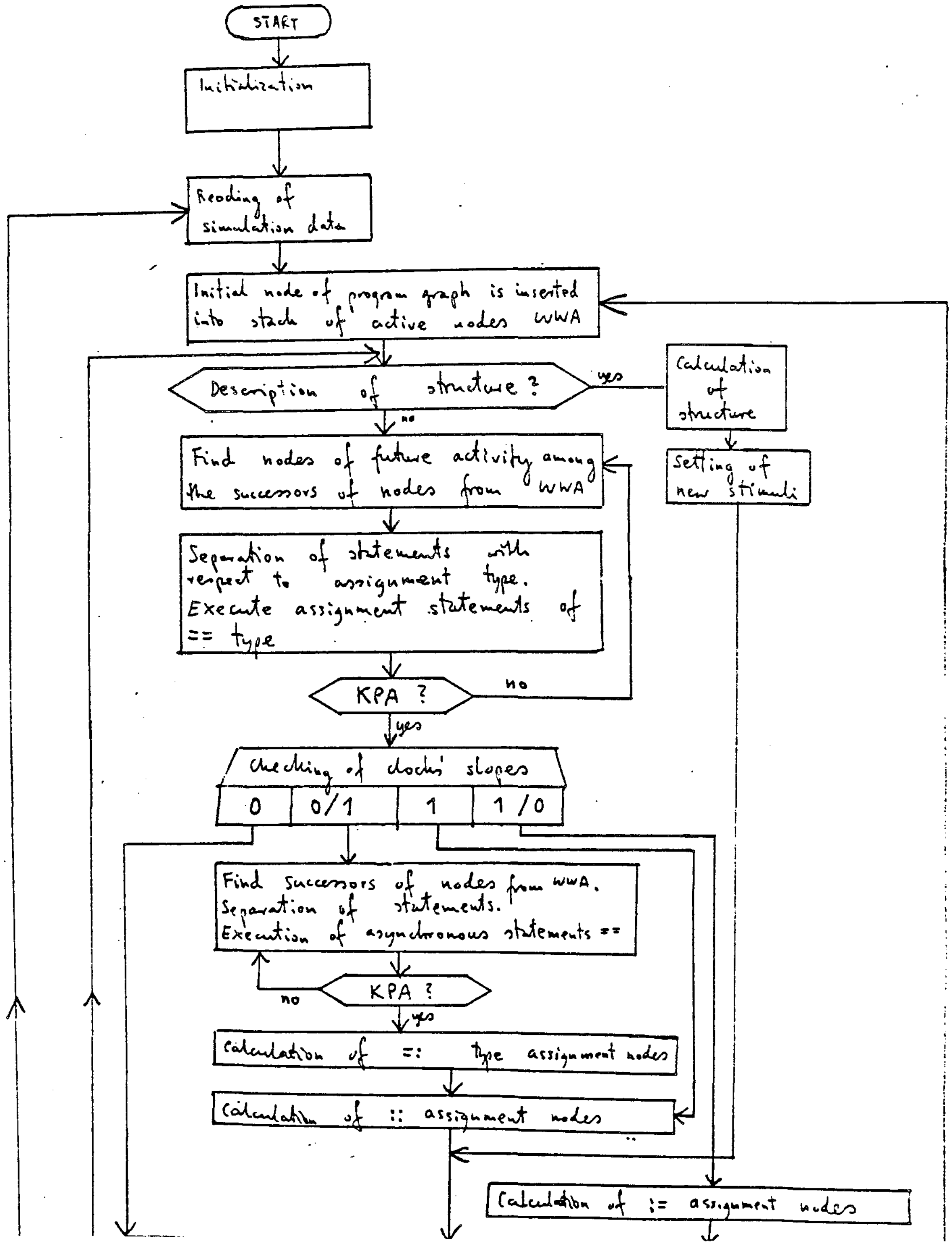
Each simulation cycle is started with determination of the nodes of future activity. Depending on the type of transition which leads to these nodes (asynchronous E or synchronous X) their numbers are written in the tables ETAB and XTAB. After each emptying of stack WWA the description of nodes in array ETAB area checked and the assignment statements of type == are executed. Next these nodes are rewritten to the stack of actual activity and the process of determination of their successors is repeated.

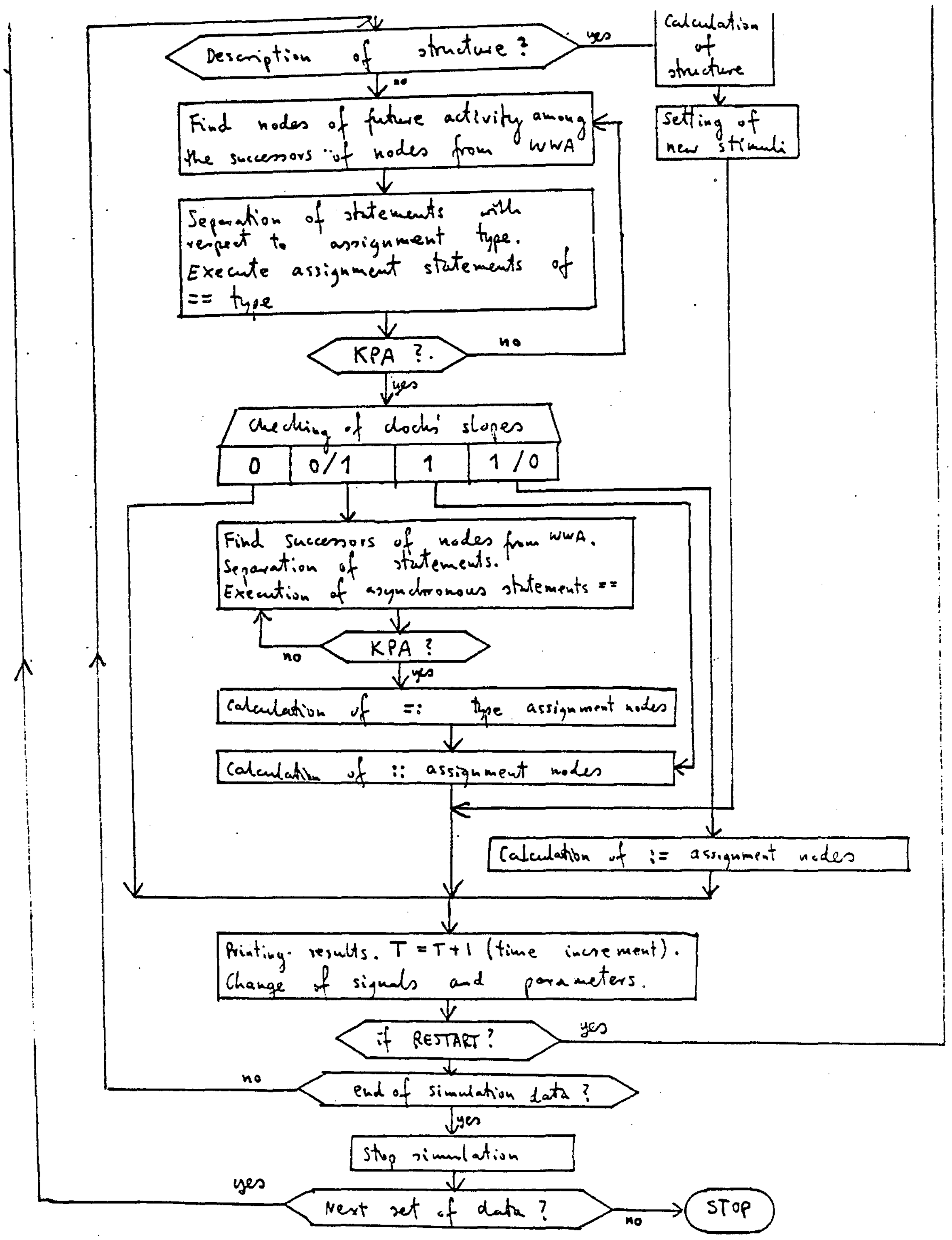
In the case when no node was written to stack ETAB or when the number of asynchronous transitions executed is equal to the number specified in TIMEASYNCH statement checking of the state of clocks is done.

For blocks, in which the leading slope of the clock has occured - the selection of nodes in vector XTAB is done. Numbers of assignment statements =:, ::, := belonging to these blocks are rewritten from XTAB to vectors respectively XTAB1, XTAB2 and XTAB3. Next the process of determining and calculating of statements in nodes from asynchronous transitions is once more repeated.

Statements from vectors XTAB1, XTAB2, XTAB3 are executed only by the respective state of the clock. The statements from XTAB1 are calculated after occuring of the leading slope, the statements from XTAB3 - of back slope, and from XTAB2 always when the clock has state "1". In the fixed phase of simulation cycle the time is increased by one time unit and eventually the change of input signals is done. One cycle is executed in one unit of simulation time. Simulation is stopped after recognition of STOPADL node or when the time specified by statement TIME was overpassed.

In the flowdiagram presented below condition KPA means checking if the cell asynchronous transistions were done or the number of transitions specified by statement TIMEASYNCH was already executed.


START
Initialization
Reading of simulation data
Initial node of program graph is inserted into stack of active nodes WWA
Description of structure ?
yes
Calculation of structure
Setting of new stimuli
no
Find nodes of future activity among the successors of nodes from WWA
Separation of statements with respect to assignment type. Execute assignment statements of == type
KPA ?
no
yes
Checking of clocks' slopes
0
0/1
1
1/0
Find successors of nodes from WWA. Separation of statements. Execution of asynchronous statements ==
KPA ?
no
yes
Calculation of =: type assignment nodes
Calculation of :: assignment nodes
Calculation of := assignment nodes


Description of structure ?
yes
Calculation of structure
Setting of new stimuli
no
Find nodes of future activity among the successors of nodes from WWA
Separation of statements with respect to assignment type. Execute assignment statements of == type
KPA ?.
no
yes
Checking of clocks' slopes
0
0/1
1
1/0
Find successors of nodes from WWA. Separation of statements. Execution of asynchronous statements ==
KPA ?
no
yes
Calculation of =: type assignment nodes
Calculation of :: assignment nodes
Calculation of := assignment nodes
Printing results. T = T+1 (time increment). Change of signals and parameters.
if RESTART ?
yes
end of simulation data ?
no
yes
Stop simulation
Next set of data ?
yes
no
STOP

6. Conclusion

This report gives a general presentation of the DIADES system and presents in detail some of its programs and languages. The topics discussed with more details are: translation of source language, abstract implementation and simulation on the register-transfer level.

ADL language has a complex syntax and many parantheses, which is the result of using LISP as a basic language of DIADES (LISP was selected to make writing translation and optimization programs easy).

Present language permits describing any digital system - often in many different ways. The choice influences the final result. Usually the choice depends on the user. It happens sometimes that the description of one kind has to be completed with the description of another kind, for instance the description of behavior requires sometimes additional description of structure of some subunit of the entire device. Some of the advantages of ADL language are shown in the program examples in the text and in the Appendices.

The compiler of ADL uses most of the LISP 9.1 facilities to reduce the storage area and execution time. Some possibilities are included in TAG to make it more user-friendly. The possibility of making listings of the ADL program with parantheses enumeration is one of them - it allows for quick finding of many types of errors. The present version of ADL language and its translator were formed after many successive improvements of the language. We are not going now to extend the description possibilities of the language but rather to make it more user-oriented and easy to use for the designers who do not have a knowledge of high level programming languages.

Still the language's use can be confusing for less experienced user. To make his life easier we are going to write the interactive preprocessor to ADL, a sort of "Programmer's Apprentice" which will ask questions of the circuit's description to the designer and will accept incrementally introduced, simplified syntax. This program will also include extended possibilities of floor plan description related to ADL primitives.

The SIMADL simulator is devoted for verifying and evaluating design concepts in an early stage of design. SIMADL is the hardware description language simulator, two-valued, asynchronous table-driven interpreter. All its properties were constructed so that it could be used in the DIADES system, it requires cooperation with TAG. The simulator applies the data base of a relatively complex structore. Most of information in arrays and lists requires access to single bits. This sparingly uses memory and permits simulation of relatively large circuits. The current limitations are not critical because greater dimensions for all these arrays can be easily declared. For the test circuits descriptions applied the simulator's speed was quite sufficient.

The time spent on simulation depends mainly on the complexity and number of conditional statements in the description rather than on the total size of the device.

The next reports of the series will cover:

a) optimization of program-graphs,

b) implementation and floor-plan generation,

c) logic and automata synthesis,

d) gate level simulation,

e) symbolic layout and mask design.

It is possible that together with delivering new reports some corrections to the previously edited reports will be done. They will be added in form of erratas.

[Adel75] Adelson-Velskiy G.M., ARlasarov V.L., Donskoy M.V.: "On the Structure of an Important Class of Exhaustive Problems and on Ways of Search Reduction for Them." Advance Papers of the Fourth International Joint Conference on Artificial Intelligence. Section 5. Search Techniques.Tbilisi. Georgia, USSR, 3-8 September 1975.

[Aho 73] Aho, A.V., Ullman, J.D.: "The Theory of Parsing, Translation and Compiling," Vol. 2. Compiling. Prentic Hall, Engl. Cliffs. N.J., 1973.

[Albi73] Albicki, A.: "Computer Method for Minimization of Number of Internal States of an Automaton." Ph.D. Thesis, Faculty of Electronics, Warsaw Technical University, Warsaw, 1973.

[Alek75] Alexandrov, E.A.: "The basis for theory of heuristic problem-solving. An approach to learn natural and design artificial intelligence," Coviet Radio, Moscov, 1975 (in Russian).

[Alle81] Allen, J., Penfield, P.Jr.: "VLSI Design Automation Activities at M.I.T." IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 7, pp. 645-653, July 1981.

[Bacz74] Baczinski, W.: "Comparison of Optimization Methods in the Space of Equivalent Algorithms," M.Sc.Th. (M. Perkowski superv.), Institute of Automatics, Warsaw Tecnical Univ. (IAWTU) (in Polish).

[Badi79] Badimirowski, K., Pienkos, J., Piestrozynski, W.: "Digital MOS-LSI Circuits," WKt Warsawa 1979 (in Polish).

[Bala65] Balas, E.: "An Additive Algorithm for Solving Linear Programs with Zero-One Variables." Operations Research. Vol. 13, No. 4, pp. 517-546, 1965.

[Bali69] Balinski, M.J., Spielberg, K.: "Methods for Integer Programming: Algebraic, Combinatorial and Enumerative," Progress in Operations Research, Vol. 3, John Wiley and Sons, New York, 1969.

[Bana74] Banasik, Z.: "Structural Synthesis of Microprogrammed Automata in the System for Automatic Design of Digital Systems DIADES," As [Bau 74], 1974 (in Polish).

[Bane69] Banerji, R.: "Theory of Problem Solving. An Approach to Artificial Intelligence," American Elsevier Publishing Company, New York, 1969.

[Bara76] Baranowski, J.: "Organization of Design Automation System," As [Bau 74], 1976 (in Polish).

[Bara81] Baranowski, J., Perkowski, M.: "Programming in LISP," Student textbook, IA WTU, 1981 (in Polish).

[Barb75] Barbacci. M.: "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems," IEEE TC Vol. C-24, No. 2, February 1975.

[Barr72] Barrow, H.C., Ambler, A.P., Burstall, R.M.: "Some Techniques for Recognizing Structures in Pictures," In Watanabe ed. Frontiers of Patter Recognition, Academic Press 1972.

[Basi76] Basinski, J.: "Method of decompositon of integrated circuit for the design automation system," MSc. Th. (W. Kuzmir, superv.), Institute of Electron Technology, Warsaw Technical University, 1976 (in Polish).

[Baug72] Baugh, Ch.R.: "Generation of Representative Functions of the NPN Equivalence Classes of Unate Boolean Functions," IEEE TC Vol. C-21, No. 12, pp. 1373-1379, December 1972.

[Behe77] Behe, W., Sansen, W.M.C., Van Overstraeten, R.: "CALMOS A Computer-Aided Layout Program for MOS/LSI," IEEE Journal on Solid-State Circuits, pp. 281-282, June 1977.

[Bell71] Bell, C.G., Newell, A.: "Computer Structures: Readings and Examples," New York, McGraw-Hill, 1971.

[Bell67] Bellman, R.E., Dreyfus, S.E.: "Dynamic programming"

[Bien79] Bienkowski, K.: "About some structural and procedural descriptions of digital computers. Proposition of the description language," Archiwum Automatylin : Telemechaniki, Vol. XXIV, No. 1, pp. 65-87, 1979 (in Polish).

[Birm74] Birman, A.: "On Proving Correctness of Microprograms," IBM J. Res. Develop. Vol. 18, pp. 250-266, May 1974.

[Blac81] Black, K.M., Harage, P.K.: "Advanced Symbolic Artwork Preparation (ASAP)," Hewlett Packard Journal, pp. 8-10, June 1981.

[Blik72] Blikle, A., Mazurkiewicz, A.: "An Algebraic Approach to the Theory of Programs, Algorithms, Languages, and Recursiveness," Proc. of an Intern. Symp. and Summer School on Math. Foundations of Computer Science, Warsaw-Jabonna, August 21-27, 1972.

[Blik73] Blikle, A.: "An Algebraic Approach to Programs and Their Computations," Proc. of II Symp. and Summer School on MF of CS. High Tatras, Sept. 3-8, 1973.

[Blik76] Blikle, A., Budkowski, S.: "An Algebraic Program Verification Method Applied to Microprograms," Res. Rep. CS-76-31, Univ. of Waterloo, June 1976.

[Bobr74] Bobrow, D. G., Raphael, B.: "New Programming Languages for Artificial Intelligence." Computer Surveys, vol. 6, No. 3, September 1974.

[Boru77] Boruslawski, K.: "Application of Automatic Theorem-Proving to Program Optimization." As [Bau 74], 1977 (in Polish).

[Breu65] Breuer, M.A.: "Implementation of Threshold Nets by Integer Linear Programming." IEEE TEC, Vol. No. 6, 1965.

[Breu66] Breuer, M.A.: "General Survey of Design Automation of Digital Computers." Proc. IEEE, Vol. 54, No. 12, 1966.

[Breu72] Breuer, M.A.: "Recent Developments in the Automated Design and Analysis of Digital Systems," Proc. IEEE, Vol. 60, str. 12-27, 1972.

[Breu72] Breuer, M.A.(ed.): "Design Automation of Digital Systems," Vol. 1, Prentice Hall, Engl. Cliffs, N.J., 1972, Vol. 2, 1976.

[Broo81] Brooksby, M.W., Castro, P.L.: "University and Industrial Cooperation for VLSI,:" Hewlett-Packard Journal, pp. 29-35, June 1981.

[Broo81] Brooksby, M.W., Castro, P.L., Hanson, F.L.: "Benefits of Quick-Turnaround Integrated Circuit Processing," Hewlett-Packard Journal, pp. 33-35, June 1981.

[Brow77] Brown, R.: "Use of Analogy to Achieve New Expertise," MIT, Ph.D. Thesis, AI-TR-403, April 1977.

[Budk78] Budkowski, S., Dembinski, P.: "Verification, Design, Description Oriented Microprogramming Language," Proc. EUROMICRO-78, Munich, 1978.

[Carr76] Carr, W.N., Mize, J.P.: MOS LSI Design and Application," McGraw Hill, New York, 1972.

[Cern74] Cerny, E., Marin, M.A.: "A Computer Algorithm for the Synthesis of Memoryless Logic Circuits," IEEE TC, Vol. pp. 455-465, May 1974.

[Chan7] Chang, Ch., Lee, R.: "Symbolic Logic and Proving," Academic Press, New York, London 197 .

[Chak70] Chakrabarti, K.K., Choudhury, A.K., Basu, M.S.: "Complementary Function Approach to the Synthesis of Three-Level NAND Networks," IEEE TC, Vol. C-19, pp. 509-514, June 1970.

[Chou67] Choudhury, A.K., Chakrabarti, K.K., Sharma, D.: "Some Studies on the Problem of Three Level NAND Networks Synthesis," Int. J. Control, Vol. 6, No. 6, pp. 547-572, 1967.

[Darr69] Darring, I.: "The Description Simulation and Automatic Implementation of Digital Computer Processors," Electrical Engineering Department, Carnegie-Mellon University, Pittsburg, Pennsylvania, May 1969.

[Davi69] Davidson, E.S.: "An Algorithm for NAND Decomposition under Network Constraints," IEEE, TC, Vol. C-18, pp. 1098-1109, December 1969.

[Ders77] Dershowitz, N., Manna, Z.: "Inference Rules for Program Annotation," Stanford Artificial Intelligence Laboratory Menno AIM-303, October 1977.

[Diet67] Dietmeyer, D.L., Schneider, P.R.: "Identification of Symmetry, Redundancy and Equivalence of Boolean Functions," IEEE TEC, Vol. EC-16, pp. 804-817, December 1967.

[Dire81] Director, S.W., Parker, A.C., Siewiovek, D.P., Thomas, D.E.,Jr.: "A Design Methodology and Computer Aids for Digital VLSI Systems," IEEE Transactions on Circuits and Systems Vol. CAS-28, No. 7, pp. 634-644, July 1981.

[Dosh78] Doshi, Dietmeyer: "Automated PLA Synthesis of the Combinational Logic of a DDL Description," Univ. of Wisconsin, ECE-78-17.

[Dule68] Duley, J. R., Dietmeyer D. L.: "A Digital System Design Language (DDL)," IEEE TC, Vol. C-19, No. 9, 1968.

[Dwor77] Dworak, J.: "Global Optimization of Programs in the System for Automatic Design," As [Ban 74], 1977 (in Polish).

[Dysk78] Dysko, P.: "Implementation of Multipurpose, Multistrategic Combinatorial Problem Solver in FORTRAN," Inst. Autom. Techn., Univ. Wasaw, 1978 (in Polish).

[Edmo72] Edmonds, J., Karp, R.: "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," JACM, Vol. 19, No. 2, pp. 248-264, 1972.

[Ersh73] Ershov, A. P.: "Present State of the Theory of Program Schemes," Problems of Cybernetics, Vol. 27, Moscow, 1973, pp. 87-110 (in Russian).

[Chou67] Choudhury, A.K., Chakrabarti, K.K., Sharma, D.: "Some Studies on the Problem of Three Level NAND Networks Synthesis," Int. J. Control, Vol. 6, No. 6, pp. 547-572, 1967.

[Darr69] Darring, I.: "The Description Simulation and Automatic Implementation of Digital Computer Processors," Electrical Engineering Department, Carnegie-Mellon University, Pittsburg, Pennsylvania, May 1969.

[Davi69] Davidson, E.S.: "An Algorithm for NAND Decomposition under Network Constraints," IEEE, TC, Vol. C-18, pp. 1098-1109, December 1969.

[Ders77] Dershowitz, N., Manna, Z.: "Inference Rules for Program Annotation," Stanford Artificial Intelligence Laboratory Menno AIM-303, October 1977.

[Diet67] Dietmeyer, D.L., Schneider, P.R.: "Identification of Symmetry, Redundancy and Equivalence of Boolean Functions," IEEE TEC, Vol. EC-16, pp. 804-817, December 1967.

[Dire81] Director, S.W., Parker, A.C., Siewiovek, D.P., Thomas, D.E.,Jr.: "A Design Methodology and Computer Aids for Digital VLSI Systems," IEEE Transactions on Circuits and Systems Vol. CAS-28, No. 7, pp. 634-644, July 1981.

[Dosh78] Doshi, Dietmeyer: "Automated PLA Synthesis of the Combinational Logic of a DDL Description," Univ. of Wisconsin, ECE-78-17.

[Dule68] Duley, J. R., Dietmeyer D. L.: "A Digital System Design Language (DDL)," IEEE TC, Vol. C-19, No. 9, 1968.

[Dwor77] Dworak, J.: "Global Optimization of Programs in the System for Automatic Design," As [Ban 74], 1977 (in Polish).

[Dysk78] Dysko, P.: "Implementation of Multipurpose, Multistrategic Combinatorial Problem Solver in FORTRAN," Inst. Autom. Techn., Univ. Wasaw, 1978 (in Polish).

[Edmo72] Edmonds, J., Karp, R.: "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," JACM, Vol. 19, No. 2, pp. 248-264, 1972.

[Ersh73] Ershov, A. P.: "Present State of the Theory of Program Schemes," Problems of Cybernetics, Vol. 27, Moscow, 1973, pp. 87-110 (in Russian).

[Feld72] Feldman, J. A., Low, J. R., Swinehart, D. C., Taylor, R. H.: "Recent Developments in SAIL - an ALGOL - Based Language for Artificial Intelligence," FJCC, Vol. 41, Part 2, pp. 1193-1201, 1972.

[Fitz71] Fitzwater, D. R., Hintz, C. A.: "A System for the Formal Definition of Digital Systems," Computer Sciences Technical Report 141, November 1971.

[Fran74] Frankowski, K. S., Dec, L.: "Basic Information on System CYBER," PWN, Warsawa 1974 (in Polish).

[Frac72] J. Frackowiak: "The Synthesis of Minimal Hazardless TANT Networks," IEEE TC, Vol. C-21, No. 10, pp. 1099-1108, October 1972.

[Free71] Freeman, P., Newell, A.: "A Model for Functional Reasoning in Design," Proc. IJCAI 2, p. 621, 1971.

[Frie74] Friedman, A. D., Menon, P. R.: "Fault Detection in Digital Circuits," Prentice Hall, 1971.

[Frie69] Friedman, T. D., Yang, S.: "Methods Used in an Automatic Logic Design Generator (ALERT)," IEEE Transactions on Computers, Vol. C-18, No. 7, July 1969.

[Gare76] Garey, M. R., Johnson, D. S., So, H. C.: "An Application of Graph Coloring to Printed Circuit Testing," IEEE Transactions on Circuits and Systems, Vol. CAS-23, No. 10, pp. 591-598, October 1976.

[Gare77] Garey, M. R., Hwong, F. K., Johnson, D. S.: "Algorithms for a Set Partitioning Problem Arising in the Design of Multipurpose Units," IEEE TC, Vol. C-26, No. 4, pp. 321-328, April 1977.

[Gilb68] Gilbert, E. N., Pollak, H. O.: "Steiner Minimal Trees," SIAM Journal of Applied Mathematics, Vol. 16, No. 1, 1968.

[Gimp67] Gimpel, J. F.: "The Minimization of TANT Networks," IEEE TEC, Vol. EC-16, pp. 18-38, February 1967.

[Gize76] Gize, J.: "Methods of Description of Parallel Processes," M.Dc.Th. (supen. P. Misiarewik), Warsawa 1976 (in Polish).

[Gemp75] Gempel, J.: "Language to Generate Masks - 7511, Description of Languge and Sytem," Inst. Electron. Technology, Warsaw Tech. Univ., 1975 (in Polish).

[Geye71] Geyer, J. M.: "Connection Routing Algorithm for Printed Circuit Boards," IEEE Transactions on Circuit Theory, Vol. CT-18, No. 1, pp. 95-100, January 1971.

[Glus70] Glushkov, V. M.: "Synthesis of Microprograms and Stack Automata," From "Computers with Extended Interpretation Systems," Naukova Dumka, Kiev, 1970 (in Russian).

[Glyn76] Glynn, D. R. M.: "Microprocessors Technlogy, Architecture and Application," Wiley Interscience Publ., J. Wiley and Sons, 1976.

[Gluc78] Gluch, M.: "Implementation of Extended Version of MICROPLANNER for CDC CYBER Comuter," As [Baer 74], 1978.

[Gras70] Grasselli, A., Montanari, U.: "On the Minimization of Read-Only Memories in Microprogrammed Digital Computers," IEEE TC, Vol. C-19, No. 11, pp. 1111-1114, November 1970.

[Halb74] Halbauer, G.: "Procedures for State Reduction and Assignment in One Step in Synthesis of Asynchronous Sequential Circuits," Proc. Intern. IFAC Symp. on Discrete Systems, Riga, 30 September - 4 October, pp. 272-282, 1974.

[Hamm69] Hammer, P. L., Rudeanu, S.: "Pseudo-Boolean Programming," Operations Research, Vol. 17, No. 2, 1969.

[Hart76] Hartenstein, R.: "Hardware Description Languages /HDL/ to Cope With Complexity of Large Digital Systems," INFOPOL, March, 1976, Warszawa.

[Hayd81] Haydamach, W. J., Griffin, D.J.: "VLSI Design Strategies and Tools," Hewlett-Packard Journal, pp. 5-12, June 1981.

[Henn68] Hennie, F. C.: "Finite-State Models for Logical Machines," John Wiley, New York, 1968.

[Hewi72] Hewitt, C.: "Description and Theoretical Analysis (Using Schemata) of PLANNER: A Language for Proving Theorems and Manipulating Knowledge in a Robot," AI Laboratory, MIT, Cambridge, Massachusetts, 1972.

[Hewi75] Hewitt, C.: "How to Use What You Know," Proc. IVth A.I. Conf., Tbilisi, Georgia, pp. 189-198, 1975.

[High] Hightower, D. W.: "A Solution to the Line-Routing Problems on the Continuous Plane," Preprint.

[High] Hightower, D. W.: "The Interconnection Problem - A Tutorial."

[Hill72] Hill, T. I., Peterson, G. R.: "Digital Systems: Hardware Organization and Design," J. Wiley and Sons, Inc., New York, 1972.

[Hill72] Hill, T. I., Peterson, G.R.: "Digital Systems: Hardware Organization and Design," J. Wiley and Sons, Inc., New York, 1972.

[Henn68] Hennie, F. C.: "Finite-State Models for Logical Machines," John Wiley, New York, 1978.

[Hlaw71] Hlawiczka, A.: "Realization of n-variable Function by Hazardless Three-level NOR Combinational Logic Networks," Proc. of the 5th National Conference on Automatic Control, Vol. 2, Automata Theory, pp. 19-26, Gdansk 1971 (in Polish).

[Hopc69] Hopcroft, J. E., Ullman, D.: "Formal Languages and their Relation to Automata," Addison-Wesley Publ. Company, Reading, Mass., 1969.

[Hopc74] Hopcroft, J. E.: "Complexity of Computer Computations," IFIP Congress 74, Math. Aspects of Inform. Processing, Vol. 3, pp. 620-626, Stockholm, 5-10 August, 1974.

[Hort81] Horton, R., Thomas, D., Rozeboom, R.: "Design Automation Speeds through Customization of Logic Arrays," Electronics, pp. 132-137, July 14, 1981.

[Hsia72] Hsiao-Peng Lee, Davidson, E. S.: "A Transform for NAND Network Design," IEEE TC, Vol. C-21, No. 1, pp. 12-20, January, 1972.

[Ibar76] Ibaraki, T.: "Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms," International Journal of Computer and Information Sciences, Vol. 5, No. 4, pp. 315-344, 1976.

[Iver62] Iverson, K. E.: "A Programming Language," New York, Wiley, 1962.

[Jabl74] Jablonskij, S. V., Lupanov, O. B. (ed.): "Discrete Mathematics and the Questions of Cybernetics," Science, Moscov, 1974.

[Jack75] Jackson, M. A.: "Introduction to Artificial Intelligence," Petrocelli, New York, 1975.

[Jank75] Jankowski, K.: "Optimization of Algorithms in the Class of Equivalent Algorithms - Operator Organizing Program Loops in the Transformed Algorithm," As [Baer 74], 1975 (in Polish).

[Jele75] Jeleniewski, T., Sielicki, A.: "Methodology and Computer Aided Technical Design," WASC Library, Technical University of Wroclaw, 1975 (in Polish).

[Jozw75] Jozwiak, L.: "Graphical Language KAMA," IA WTU, 1975.

[Jozw76] Jozwiak, Kw: "Correction of Syntactic Errors," [illegible] Superv.) IA WTU, 1976

[Jozw77] Jozwiak, L.: "Correction of Syntactic Errors for Language ADL," Proc. VII Nat. Conf. Autom., Rzeszow, Poland, 15-17 Sept. 1977, Vol. 1, pp. 766-777 (in Polish).

[Karp72] Karp, R.M.: "Reducibility Among Combinatorial Problems," Complexity of Computer Computation, Plenum Press, ed. Miller etc., pp. 85-103, New York, 1972.

[Katz78] Katz, S.: "Program Optimization Using Invariants," IEEE Transactions on Software Engineering, Vol. Se-4, No. 5, September 1978, pp. 378-389.

[Kauf68] Kaufman, A.: "Intrduction a la combinatorique en vue des applications," Dunod, Paris, 1968.

[Kern72] Kerntopf, P., Michalski, A.: "Selected Problems in Synthesis of Combinational Logic Circuits," PWN, Warszawa, 1972 (in Polish).

[Kime75] Kime, Ch. R.: "Fault-Tolerant Computing: An Introduction and Perspective," IEEE TC, Vol. C-24, No. 5, pp. 457-460, May 1975.

[King76] King, J.: "Symbolic Execution and Program Testing," CACM, No. 7, 1976.

[Koha70] Kohavi, Z.: "Switching and Finite Automata Theory," McGraw-Hill, 1970.

[Korb74] Korbut, A. A., Finkelstein, J. J.: "Discrete Programming," PWN, Warszawa, 1974 (in Polish).

[Kohl74] Kohler, W. H., Steiglitz, K.: "Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems," JACM, Vol. 21, pp. 140-156, 1974.

[Komo76] Komorowski, W.: "Languages for Description of Architecture and Structures of Digital Systems," Informatyka 1976, No. 2, (in Polish).

[Kozm78] Kozminski, K. A.: "Investigation of Possibilities of Application of System DIADES to MOS LSI Design," M.Sc.Th. (W. Kaimin, Superv.) Institute of Electron Technology, WTC, 1978 (in Polish).

[Kras79] Kraslinski, A., Otwinowski, A., Perkowski, M.: "The System ORTOCARTAN for Analytic Calculations - Detailed Description," Report of Centrum Astronomii PAN, 1979.

[Kulp70] Kulpa, Z.: "Design of Quasi-Minimal Logic Circuits of Many Variables with NAND or NOR Gates," M.Sc.Th. (W. Traugh, Superv.) IA WTU, 1970 (in Polish).

[Lang77] Lange, E. E., Fricnovitsch, T. F.: "On Concurrent Solution to the Minimization and State Assignment of Asynchronous Automata," Proc. II IFAC Symposium on Discrete Systems, Vol. 1, pp. 146-153, Dresden, 14-19March, 1977.

[Levi75] Levi, G., Sirovich: "A Problem Reduction Model for Non Independent Subproblems," Proc. IV A. I. Conf., Tbilisi, Georgia, pp. 340-344, 1975.

[Lang78] Langer, R., Dugan, T.: "Say it in a High-Level Language with 64-K Read-Only Memories," Electronics, pp. 119-124, April 1978.

[Lato76] Latombe, J. C.: "Artificial Intelligence in Computer-Aided Design: The "TROPIC" System," Proc. of the IFIP Working Conference on Computer-Aided Design Systems, North-Holland, Amsterdam, pp. 61-120, 1977.

[Lato77] Latombe, J. C. (ed.): "Artificial Intelligence and Pattern Recognition in Computer-Aided Design," Proceedings of the IFIP Working Conference Organized by Working Group 5.2, Computer Aided Design, Grenoble, France, March 17-19, 1978, North-Holland Publ. Comp., Amsterdam, New York, Oxford.

[Latt81] Lattin, W. et al.: "A Methodology for VLSI Chip Design," Lambda, Second Quarter 1981, pp. 34-44.

[Lant81] Lanther, V.: "An O(N Log N) Algorithm for Boolean Mask Operations," Proc. 18th Design Automation Conference, 1981.

[Lawl66] Lawler, E. L., Wood, D. E.: "Branch-and-Bound Methods, A Survey," Operations Research, Vol. 14, No. 4, pp. 699-719, 1966.

[Lee82] Lee, E. B., Perkowski, M.: "A New Approach to Structural Synthesis of Automata," in preparation.

[Lee72] Lee, H. P., Davidson, E. S.: "A Transform for NAND Network Design," IEEE TC, Vol. C-21, No. 1, str. 12-20, January 1972.

[Leem75] Leeman, G. G.: "Some Problems in Certifying Microprograms," IEEE TC, Vol. C-24, No. 5, May 1975, pp. 545-553.

[Lewi79] Lewinski, A.: "Algebraic Simulation as a Method of Verification and Synthesis of Microprograms," Archiwum Automatyki i Telemechaniki (in Polish), submitted for publication, 1979.

[Liag68] Liagett, ., Cain, .: "Analysis of Algorithms for the Zero-One Programming Problem," CACM, Vol. 11, p. 837, December 1968.

[Liu70] Liu, T. K., Nakagana, T., Muroga, S.: "Synthesis Networks of MOS Cells by Integer Programming," Internal Memo Dep. Sci., Univ. Illinois, Urbana, 1970.

[Lynn71] Lynn, D. K.: "Computer-Aided Layout System for Integrated Circuits," IEEE TC, VCT-18, No. 1, pp. 128-139, 1971.

[Loku77] Lokucijewski, R.: "Criteria of Selection of the Combinational Network Structure," As [Kulp 70], 1977 (in Polish).

[Male63] Maley, G. A., Earle, J.: "The Logical Design of Transistor Digital Computers," Engl. Cliffs, N.I., Prentice-Hall, 1963.

[Maje77] Majewski, W., Albicki, A. (red. red.): "Design of Digital Systems for Telecommunication," Wydavmictwa Komunikacji i tgunoiu, Warszawa, 1977 (in Polish).

[Maci77] Macicjaugh, J.: "Organization of Subsystem of Synthesis of Combinational Circuits," As [Baer 74], 1977.

[Mank74] Mankiewicz, Cz.: "Source Language of the System for Automatic Design of Block-Oriented Digital Systems of Automatic Control and its Compiler to the Graph Language, As [Baer 74], 1974 (in Polish).

[Marc74] Marczyusk, R., Pulayn, W., Sockacki, J.: "Language OSM - Aspects of Design of a Language for Hardware Simulation," CO PAW Works, No. 160, Warszawa, 1974 (in Polish).

[Mart75] Martelli, A., Montanari, U.: "From Dynamic Programming to Search Algorithms with Functional Costs," Proc. IV A. I. Conf., Tbilisi, Georgia, str. 345-350, 1975.

[Matw] Matwin, S.: "Optimizing Transformations on Progrms," PWN, 1977 (in Polish).

[Mays71] Mays, C. H.: "A Brief Survey of Computer-Aided Integrated Circuit Layout," IEEE Transactions on Circuit Theory, Vol. CT-18, No. 1, January, 1971.

[Mazu74] Mazurkiewicz, A.: "Foundations of the Programming Theory," From "Problems of Information Processing," WNT, Vol. 2, Warsaw, 1974, pp. 39-94 (in Polish).

[McCa62] McCarthy, J. (red.): "LISP 1.5 Programmers Manual," The MIT Press, Cambridge, Massachusetts, 1962.

[McCl65] McCluskey, E. I.: "Introduction of the Theory of Switching Circuits," New York: McGraw-Hill, 1965.

[McDe73] McDermott, D. V., Sussman, G. J.: "The CONNIVER Reference Manual," AI Memo 259 MIT-AI Laboratory, July 1973.

[McDe77] McDermott, D.: "Flexibility and Efficiency in a Computer Program for Designing Circuits," AI-TR-402, AI Lab, MIT, June 1977.

[Mead80] Mead, C. Conway, L.: "Introduction to VLSI Systems," Addison-Wesley Publishing Company, 1980.

[Meli74] Melichov, A. N., Berstein, L. S., Kureicik, W. M.: "Application of Graphs to the Synthesis of Discrete Systems," Nauka, Moscov 1974 (in Russian).

[Mich69] Michalski, R. S.: "About Quasi-Minimal Solution of the Covering Problem," Archiwum Automatyki i Telemechaniki, No. 4, 1969 (in Polish).

[Mich72] Michalski, R. S., Kulpa, Z.: "A system of programs for the synthesis of combinational switching circuits using the method of disjoint stars," Proc. of IFIP 71 Congress, Vol. TA-2, North Holland, 1972.

[Mich72] Michalowski, K.: "Computer-Aided Design of Switching Functions," As [Kulp 70], 1972.

[Mill65] Miller, R. E.: "Switching Theory, Vol. 1-2, John Wiley, New York, 1965.

[Misi76] Misiurewicz, P., Perkowski, M.: "Theory of Automata," PW, Ed. 3, 1976 (in Polish).

[Min72] Min Wen Du.: "A Way to Find a Lower Bound for the Minimal Solution of the Covering Problem," IEEE TC, Vol. C-21, pp. 317-318, March, 1972.

[Mont70] Montanari, B. U.: "Separable Graphs Planar Graphs and Web Grammars," Information and Control, Vol. 16, p. 243, 1970.

[Morr67] Morreale, E.: "Partitioned List Algorithms for Prime Implicants Determination from Canonical Forms," IEEE TEC, Vol. EC-16, pp. 611-620, May 1967.

[Morr70] Morris, J. B., Singleton, D. J.: "The University of Texas 6400/6600 LLSP 1.5.9," Revision III, Comp. Center Univ. of Texas, Austin, Texas, 1970.

[Muro72] Muroga, S., Ibaraki, T.: "Design of Optical Switching Networks by Integer Programming," IEEE TC, Vol. C-21, pp. 573-582, June 1972.

[Nan] Nan, N., Feuer, M.: "A Method for the Automatic Wiring of LSI Chips."

[Naur66] Naur, P.: "Proof of Algorithms by General Snapshots," B.I.T.6, pp. 310-317, 1966.

[Newe69] Newell, A.: "Heuristic Programming: Ill-Structured Problems," Progress in Operations Research, Vol. 3, John Wiley and Sons, New York, 1969.

[Newe72a] Newell, A., Simon, H. A.: "Human Problem Solving," Prentice Hall, Engl. Cliffs, New Jersey, 1972.

[Newe72b] Newell, A.: "Production Systems: Models of Control Structure," Visual Inform. Processing, Chase, W. G. (ed.) Academic Press, New York, 1972.

[Newt81] Newton, A. R., Pederson, D. O., Sangiovanni-Vincentelli, A. L., Sequin, C. H.: "Design Aids for VLSI: The Berkeley Perspective," IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 7, July 1981, pp. 666-680.

[Nils71] Nilsson, N. J.: "Problem-Solving Methods in Artificial Intelligence," McGraw Hill, New York, 1971.

[OPTI72] "OPTIMOS" MOS Fairchald-Semiconductor 1972.

[Ore66] Ore, O.: "Introdution to Graph Theory."

[Penn72] Penney, W. M., Lau, L.: "MOS Integrated Circuits," Microelectronics Series, Van Nostrand Reinhold Company, New York, 1972.

[Perk70] Perkowski, M.: "Application of Logic Schemata of Algorithms to the Synthesis of Automata," M. Sc. Thesis (supervised by W. Traczyk), Department of Automatic Control and Remote Control, Technical University of Warsaw, 1970 (in Polish).

[Perk72] Perkowski, M., Misiurewicz, P.: "Synthesis of Sub-minimal TANT Networks," Proceedings of Conference "Integrated Circuits in Industrial Control Systems," Katowice 19 October 1972, pp. 62-70 (in Polish).

[Perk73] Perkowski, M.: "A System for Automatic Design of Digital Systems," Proceedings of Conference "Modern Problems of Control and Computer Science," Section C, Gliwice 18-20 September 1973, pp. 277-304 (in Polish).

[Perk74a] Perkowski, M.: "Relational-Structure Language nd their Application in the System for Automatic Design of Block-Oriented Digital Systems of Automatic Control," Proceedings of Sixth National Conference on Automatic Control, Poznan 7-11, Sept. 1974 (in Polish), Extended English version: Institute of Automatic Control, Technical University of Warsaw, Technical Report No. 9/1975, Warsaw October 1975.

[Perk74b] Perkowski, M.: "Heuristic Optimization in the System for Automatic Design of Block-Oriented Digital Systems of Automatic Control," Proceedings of 1 Symposium "Methodology of Heuristics," Polish Cybernetical Society, Warsaw, 28 September 1974 (in Polish).

[Perk74c] Perkowski, M., Mankiewicz, Cz.: "ADL - Source Language of the System for Automatic Design of Block-Oriented Digital Sytems of Automatic Control," Abstrats of Second Symposium on Systems, Modelling and Control, Zakopane 27-30 May 1974, p. 91-93 (in Polish).

[Perk74d] Perkowski, M.: "Synthess of Logic Networks on SSI Integrated Circuits," Unpublished manuscript in Polish.

[Perk74e] Perkowski, M.: "A System for Automatic Design of Digital Systems," Proceedings of FCIP," Symposium INFORMATICA 1974, Bled. Yougoslavia, 7-12 October 1974, Paper 4.4.

[Perk75a] Perkowski, M., Baczynski, W., Jankowsi, K.: "Experiments with Heuristic Program Optimizing Program," Proceedings of II Symposium "Methodology of Heuristics," Polish Cybernetical Society, Warsaw, 27 September 1975 (in Polish).

[Perk75b] Perkowski, M.: "An Automated Approach to the Design of Block-Oriented Digital Systems," Institute of Automatic Control, Technical Univ. of Warsaw, Technical Univ. of Warsaw, Technical Report 8/1975, Warsaw October 1975.

[Perk76a] Perkowski, M., Jankowski, K.: "Verification and Optimization of Control Automaton Programs in the System for Automatic Design," Proceedings of Symposium "Fault Diagnosis of Digital Networks and Fault-Toleran Computing," Wisla 24-27 May 1976, pp. 104-112.

[Perk76b] Perkowski, M.: "An Example of Heuristic Programming Application in the Three-Level Combinational Logic Design," Materialy III Sympozjum "Metody Heurezy," Warszawa, Vol. 1, pp. 105-132, 26 Sept. 1976.

[Perk76c] Perkowski, M., Baranowski, J.: "Design of Digital Systems Using the Programming Language ADL," IA WTU, 1976 (in Polish).

[Perk76d] Perkowski, M.: "Design of Digital Systems Using the Programming Language ADL, Part II, Examples of Application," Unpublished manuscript in Polish, 1976.

[Perk76e] Perkowski, M.: "Synthesis of Multioutput Three Level NAND Networks," Proceedings of Seminar on Computer Aided Design, pp. 238-265, Budapest, 3-5 November 1976.

[Perk76f] Perkowski ,M.: "Multipurpose, Multistrategic, Combinatorial Problem Solver," Proc. 3rd Intern. Symp. Heuristic Methods, PTC, Warsaw, 1976, pp. 23-90, 1976.

[Perk76g] Perkowski, M.: "Design of Digital Systems in the System for Automatic Design DIADES," Proceedings of the VII National Conference on Automatic Control, Rzesiow, Poland, 15-17 Sept. 1977, Vol. 1, pp. 745-754 (in Polish).

[Perk77a] Perkowski, M., Rydzewski, A., Misiurewicz, P.: "Theory of Logic Circuits," Warsaw, WPW, 1977 (in Polish).

[Perk77b] Perkowski, M.: "A Method of Validation of Parallel Programs in the System for Automatic Design of Block-Oriented Digital Systems," Proceedings of the II IFAC Symposium on Discrete Systems, Dresden, 14-19 March 1977, t.2, pp. 71-88, 1977.

[Perk77c] Perkowski, M.: "ADL-Source Language of the System for Automatic Design," Prace konferencji "Organizacja maszyn cyfrowych i mikroprogramowania," COPAN i IMM 24-26, IX, 1975, PWN, Warszawa, 1977, t.1, pp. 167-180.

[Perk77d] Perkowski, M.: "An Application of General Problem-Solving Methods in Computer-Aided Design: The MULTICOMP System and Its Problem-Oriented Source Language," Referaty na IV Miedzyn. Sympozjum," Metody Heurezy, Vol. 3, pp. 55-102, Warszawa, 24 Sep. 1977.

[Perk78a] Perkowski, M.: "The State-Space Approach to the Design of a Multipurpose Problem-Solver for Logic Design," Proc. IFIP WG 5.2 Conf., "Artificial Intelligence and Pattern Recognition in Computer Aided Design," (Latombe ed.), IFIP North Holland Publishing Comp. 1978, pp. 123-140.

[Perk78b] Perkowski, M.: "Some Selected Problems of Automatic Design of MOS LSI Digital Systems," Inst. Automatic Control, Warsaw, Techn. Univ. 1978 (in Polish).

[Perk79a] Perkowski, M., Zasowska, A.: "Minimal Area MOS Asynchronous Automata," Proc. Intern. Symp. on Applied Aspects of Automata Theory," Varna, Bulgaria, pp. 284-299, 14-19 May 1979.

[Perk79b] Perkowski, M.: "Automatischer Entwurf von MOS-LSI-digitalen Schaltungen in System DIADES," Messen, Steuern, Regeln., No. 6, pp. 346-350, 1979.

[Perk79c] Perkowski, M.: "Design Automation System for Digital Systems," Magyar Tudomanyos Akademia, Szamitastechnikai Es Automatizalasi Kutato Intezete, Budapest Tanulmanyok 99/ , pp. 93-112, 1979.

[Perk80a] Perkowski, M.: "The Method of Solving Combinatorial Problems in Automatic Design of Digital Systems," Inst. Autom. Techn. Univ. Warsaw, 1980 (in Polish) p. 434.

[Perk80b] Perkowski, M.: "Digital Design by Problem-Solving Transformations," Proc. of Conf. "Artificial Intelligence and Control Information Systems of Robots," Smolenice, Czechoslovakia, June 30 - July 4, 1980.

[Perk80c] Perkowski, M.: "Design Automation of Digital Systems as a New Domain for AI Research," IA WTV, 1980.

[Perk80d] Perkowski, M.: "Selected problems of automatic design of digital systems," Institute of Automatic Control, Technical University of Warsaw, 1980 (in Polish).

[Perk80e] Perkowski, M., Goralski, A., Zielinski, G.: "Elements of Artificial Intelligence," Book in editorial preparation by WNT (in Polish).

[Perk80f] Perkowski, M.: "Multistrategical Problem Solver," Proceedings of the Second International Meeting on Intelligent Robotics and Knowledge Representation, Repino near Leningrad, 12-19 October 1980, To be published in Plenum Press.

[Perk80g] Perkowski, M.: "The Programming System for Symbolic Analysis, Optimization, and other Problems Related to the Semantics of Control Programs," Technical Report, Inst. Autom. Control, Warsaw, Techn. Univ., Warsaw, 1980.

[Perk80h] Perkowski, M.: "General Methods for Solving Combinatorial Problems," Chapter 4 in A. Gorslski (ed.) "Problem, method, solution," Collection 4, Scientific-Technical Publishers, 1981 (in Polish).

[Perk80i] Perkowski, M.: "Problem-Oriented Descriptive Language of System MULTICOMP," Unpublished manuscript in Polish.

[Perk80j] Perkowski, M.: "Design of Combinational Circuits in System DIADES," Unpublished manuscript in Polish.

[Perk80k] Perkowski, M.: "Design of MOS LSI Masks in System DIADES," Unpublished manuscript in Polish.

[Perk81] Perkowski, M., Goldstein, N. B.: "A New Algorithm for Multioutput Function Minimization Based on Reduction to Graph-Coloring Problem," Preprint.

[Perk82a] Perkowski, M., Tuszynski, A.: "Minimization of Manhattan Layout for MOS Digital Networks," In preparation.

[Perk82b] Perkowski, M.: "Description of Software Package of Boolean Function Minimization (Listings and Examples)," In preparation.

[Pete74] Petersohn, U., Voss, K., Weber, K. H.: "Genetische Adaptation - ein Stochastisches Suchverfahren fur Diskrete Optimierungsprobleme," Math. Operationsforschung und Statistik, Vol. 5, No. 7--8, 1974.

[Pfla69] Pflantz, J. L., Rosenfeld, A.: "Web Grammars," Proceedings of International Joint Conference on Artificial Intelligence, p. 609, May 1969.

[Piec78] Piechota, B.: "Automatic design of LSI circuits," As [Kulp 70], 1578 (in Polish).

[Piot80] Piotrowski, A.: "Simulator of digital systems described in ADL - a souce language of a design automation system DIADES, M.Sc.Th. Institute of Automatics, Warsaw Technical University, 1980.

[Pohl71] Pohl, I.: "Bi-directorial Search," Machine Intelligence 6, Meltzer, B. and Michie, D. (eds.), American Elsevier, New York, pp. 127-140, 1971.

[Popi73] Popiel, J.: "Languages for Digital Systems Description and Simulation," M. Sc. Thesis (supervised by P. Misiurewicz) Inst. Aut. Contr. Techn. Univ. of Warsaw 1973 (in Polish).

[Powe73] Powers, G. J.: "Non-Numerical Problem Solving Methods in Computer-Aided Design," Proceedings of the IFIP Working Conference on Principles of Computer-Aided Design, North Holland Publishing Company, Amsterdam, London, pp. 327-354, 1973.

[Prat71] Pratt, T. W., Friedman, D.: "A Language Extension for Graph Processing and Its Formal Semantics," CACM, Vol. 14, No. 7, July, 1971.

[Proc64] Proctor, R. M.: "A Logic Design Translator Experiment Demonstrating Relationships of Language to Systems and Logic Design," IEEE TC, Vol. EC-13, No. 4, August 1964.

[Rabi74] Rabin, M. O.: "Theoretical Impediments to Artificial Intelligence," IFIP Congress 74, Math. Aspects of Inform. Processing, Vol. 3, pp. 615-619, Stockholm, 1974.

[Rajl73] Rajlich, V.: "Relational structures and dynamics of certain discrete systems," Mathematical Foundations of Computer Science," Proceedings of Symposium and Summer School, High Tatras, September 5-8, 1973.

[Rama77] Ramamoorthy, C. V., Li, H. F.: "Pipeline Architecture," Computing Surveys, Vol. 9, No. 1, pp. 61-102, March 1977.

[Raym81] Raymond, T. C.: "LSI/VLSI Design Automation," IEEE Computers, July 1981.

[Reit65] Reiter, M., Sherman, G.: "Discrete Optimizing," J. Soc. Industr. and Appl. Math., Vol. 13, No. 3, 1965.

[Roy72] Roy Sinha, P. K., Sheng, C. L.: "A Decomposition Method of Determining Maximum Compatibilities," IEEE TC, Vol. C-21, No. 3, pp. 309-312, March 1972.

[Rous75] Roussel, P.: "PROLOG, Manuel de Reference et d'utilisation," Groupe d'Intelligence Artificielle, UER Marseille-Luminy, September 1975.

[Rudo74] Rudo, K.: "The Interconnections Routing in the Pictures of Logic Schematiz," As [Baer74], 1974.

[Ruli71] Rulifson, J. F.: "QA4 Programming Concepts," Tech. Note 60, Menlo Park, Calif.: AI Group, Stanford Res. Inst., 1971.

[Sahn80] Sahni, S., Bhatt, A., Raghavan, R.: "Complexity of Design Automation Problems," University of Minnesota, TR 80-28, 1980.

[Sauc72] Saucier, G.: "State Assignment of Asynchronous Sequentil Machines using Graph Techniques," IEEE TC Vol. C-21, pp. 282-288, March 1972.

[Scha73] Schaefer, M.: "A Mathematical Theory of Global Program Optimization," Prentice-Hall Englewood Cliffs, 1973.

[Sche81] Schetfer, L. K., Dowell, R. I., Apte, R. M.: "Design and Simulation of VLSI Circuits," Hewlett Packard Journal, June 1981, pp. 12-18.

[Schl64] Schlaeppi, H. P.: "A Formal Language for Describing Machine Logic, Timing and Sequencing (LOTIS)," IEEE TEC, Vol. EC-13, No. 4, August 1964.

[Schm74] Schmidt, D. C., Druffel, L. E.: "An Extension of the Clause-Table Approach to Multi-Output Combinational Switching Networks," IEEE TC, Vol. C-23, April 1974.

[Schm75] Schmidt, D. C., Metze, G.: "Modular Replacement of Combinational Switching Networks," IEEE TC, Vol. C-24, No. 1, pp. 29-48, 1975.

[Schn68] Schneider, P. R., Dietmeyer, D. L.: "An Algorithm for Synthesis of Multiple-Output Combinational Logic," IEEE TC, Vol. C-17, pp. 117-128, February 1968.

[Slag70] Slagle, J. R., Chang, C. L., R.C.T.: "A New Algorithm for Generating Prime Implicants," IEEE TEC., Vol. C-19, No. 4, pp. 304-311, April 1970.

[Slag71] Slagle, J. R.: "Artificial Intelligence: The Heuristic Programming Approach," McGraw-Hill, New York, 1971.

[Soch74] Sochon, A.: "Generation of Pictures of Graphs, Flowdiagrams and Block Schemata in the System for Automatic Design," AS [Bun 74], 1974 (in Polish).

[Stab70a] Stabler, E.: "System Description Languages," IEEE Transaction on Computers, Vol. C-19, No. 12, December 1970.

[Stab70b] Stabler, E. P.: "Microprogram Transformations," IEEE TC, Vol. C-19, No. 10, 1970.

[Step77] Stepien, J.: "Hierarchical System for Automatic Design of Combinational Networks Based on Arrays of Cubes," As [Bacz 74], 1976.

[Stor72] Story, J. R., Harrison, H. J., Reinhard, E. A.: "Optimum State Assignment for Synchronous Sequential Circuits," IEEE TC., Vol. C-21, No. 12, str. 1365-1373, December 1972.

[Stra77] Strawinski, T.: "Maxurkiewicz-Blikle Method in Automatic Optimization or Programs," Inst. Autom. Control, Warsaw Techn. Univ., Warsaw 1977 (in Polish).

[Suss71] Sussman, G. J., Winograd, T., Charniak, E.: "Micro-Planner Reference Manual," AI Memo 203, MIT-AI Lab., December 1971.

[Suss75] Sussman, G. J., Stallman, R. M.: "Heuristic Techniques in Compuer-Aided Circuit Analysis," IEEE Trans. on Circuits and Systems, Vol. CAS-22, pp. 857-865, No. 11, November, 1975.

[Toma75] Tomaszewska, K.: "Structural Integrated Implementation in the System for Automatic Design of Digital Systems," As [Baer 74], 1975.

[Tome72] Tomescu, I.: "A Matrix Method for Determining All Pairs of Compatible States of a Sequential Machine," IEEE TC Vol. C-21, No. 5, pp. 502-503, May 1972.

[Trac69] Traczyk, W.: "Design of Asynchronous Automata," Electrical Engineering, No. 2, WPW, Warsawa 1969 (in Polish).

[Trim81] Trimberger, S., Rowson, J. A., Lang, Ch. R., Gray, J. P.: "A Structured Design Methodology and Associated Software Tools," IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 7, pp. 618-633, July 1981.

[Trim81] Trimberger, S., Rowson, J. A., Lang, Ch. R., Gray, J. P.: "A Structured Design Methodology and Associated Software Tools," IEEE Transactions on Circuits and Systems, Vol. CAS-28, No. 7, pp. 618-633, July 1981.

[Tuch81] Tucher, M. G., Haydamack, W. J.: "VLSI Design and Artwork Verification," Hewlett-Packard Journal, pp. 25-29, June 1981.

[Ulug74] Ulug, M. E., Bowen, B. A.: "A Unified Theory of the Algebraic Topological Methods for the Synthesis of Switching Systems," IEEE TC, pp. 255-267, March 1974.

[Urba75] Urbanski, M.: "A Graphic Display Software Package for the System of Automatic Design of Digital Systems," As [Baer 74], 1975 (in Polish).

[Urba77] Urbanski, M.: "Software package for documentation of logic and electronic circuits," Informatyka, No. 3, pp. 10-12, 1977 (in Polish).

[Suss77] Sussman, G. J.: "Electrical Design - A Problem for Artificial Intelligence Research," Proc. IJCAI 5, 1977.

[Stei73] Wte h, M. E., Wte h, . E.: "Meto ma hho o poekt pobah u pobo a apatyp ," Mockba, Cobetckoe Pa o, 1973.

[Tety74] Tetyk, J.: "Abstract Implementation in the System of Automatic Block-Oriented Synthesis of Digital Systems," As [Bacz 74], 1974 (in Polish).

[Tobi81] Tobias, J. R.: "LSI/VLSI Building Blocks," IEEE Computer, August 1981, pp. 83-101.

[VanL73] Van Lier, M. C., Otten, R.: "On the Mathematical Formulation of the Wiring Problem," John Wiley & Sons, Ltd. 1973.

[Wagn77] Wagner, T. J.: "Hardware Verification," Stanford Artif. Intell. Lab., Memo AIM-304, CS. Dept. Report No. STAN-CS-77-632, Sept. 1977.

[Warr74] Warren, D. H. D.: "WARPLAN: A System for Generating Plans," Dept. of Computation Logic Memo No. 76, June 1974.

[Wegb74] Wegbreit, B.: "The Synthesis Loop Predicates," CACM, Bol. 17, No. 2, pp. 102-112, 1974.

[Wegn76] Wegner, P.: "Programming Languages - The First 25 Years," IEEE TC, Vol. C-25, No. 12, str. 1207-1225, December 1976.

[Wein67] Weinberger, A.: "Large Scale Integration of MOS Complex Logic - A Layout Method," IEEE J. of SSC, Vol. SC-2, No. 4, Dec. 1967.

[Wein68] Weiner, P., Dwyer, T. F.: "Discussion of Some Flows in the Classical Theory of Two-Level Minimisation of Multiple Output Switching Networks," IEEE TC, Vol. C-17, No. 2, February 1968.

[Weso74] Wesolowski, J.: "Transformtions of Behavior Programs in the System for Automatic Design of Block-Oriented Digital Systems of Automatic Control," As [Ban 74], 1974 (in Polish).

[West81] Weste, N. H. E.: "MULGA - An Interactive Symbolic Layout System for the Design of Integrated Circuits," The Bell System Technical Journal, Vol. 60, No. 6, July-August, 1981.

[West74] Westerberg, A. W., Stephanopoulos, G., Shah, J.: "The synthesis problem with some thoughts on evolutionary synthesis in the design of engineering systems," In Spillers W. R. (ed.): "Basic questions in design theory," North Holland, American Elsevier, 1974.

[Whee69] Wheeling, R. F.: "Heuristic Search: Structural Problem," Progress in Operations Research, Vol. 3, John Wiley and Sons, New York, 1969.

[Wolf73] Wolfendale, E.: "MOS Integrated Circuit Design," Butterworth London, 1973.

[Woju79] Wojutynski, J.: "Graph Colouring Algorithms," As [Bacz 74], 1979 (in Polish).

[Yaji74] Yajima, S., Inagaki, K.: "Power Minimization Problems of Logic Networks," IEEE TC, Vol. C-23, No. 2, str. 153-165, February 1974.

[Yosh] Yoshizawa, H., Kawanishi, H., Kani, K.: "A Heuristic Procedure for Ordering MOS Arrays," Preprint.

[Zaso78] Zasowska, A.: "Concurrent Minimization and Encoding of Synchronous and Asynchronous Automata," Inst. Autom. Control, Warsaw Techn. Univ., Warsaw 1978 (in Polish).

[Zaso79] Perkowski, M., Zasowska, A.: "The Computer-Oriented Method for Joint Minimization and State-Assignment in Synchronous and Asynchronous Automata," Proceedings of the conference "Application of digital computers in engineering design," Katowice 1979, pp. 31-42 (in Polish).