The Application of Parallel Computation to Process Simulation for the Structured Design of IC Fabrication Processes

Thesis submitted by

Walter James Cunningham Alexander

for the degree of

Doctor of Philosophy

Edinburgh Microfabrication Facility

Department of Electrical Engineering

University of Edinburgh

United Kingdom

1992

To my parents

Declaration

I declare that the research documented in this thesis is original and entirely of my own making, except where it has been indicated to the contrary.

Walter James Cunningham Alexander

Table of Contents

1 Introduction 1

1.1 Design Methodology 2

1.2 Motivation 5

1.3 Thesis Structure 5

2 Process Control in IC Fabrication 8

2.1 Introduction 8

2.2 Process Steps 9

2.2.1 Oxidation 9

2.2.2 Diffusion 12

2.2.3 Ion Implantation 14

2.2.4 Deposition 18

2.2.5 Lithography 21

2.2.6 Etching 24

2.3 Process Control and Design 26

2.4 Summary 29

3 Process Simulation 32

3.1 Introduction 32

3.2 One Dimensional Simulation 35

3.2.1 Oxidation 40

3.2.2 Diffusion 41

3.2.3 Ion Implantation 43

3.2.4 Deposition 46

3.2.5 Etch 48

3.2.6 Electrical 50

3.3 Two Dimensional Simulation 52

3.4 Computational Requirements of Process Simulation 54

3.5 Summary 56

4 Parallel Computational Systems 62
4.1 Introduction 62
4.2 Parallel Computer Architectures 65
4.2.1 SISD 68
4.2.2 SIMD 69
4.2.3 MISD 71
4.2.4 MIMD 71
4.3 The Inmos Transputer 74
4.3.1 Architecture and Concepts 74
4.3.2 Parallel Algorithms 78
4.3.3 Parallel Software - Languages and Tools 82
4.3.4 The Future 91
4.4 Summary 96

5 Parallel Implementation of Process Simulation Code 101
5.1 Introduction 101
5.2 One-Dimensional Simulation 102
5.3 Partitioning SUPREM II code 105
5.4 Parallel Matrix Solution Method 114
5.5 Hardware Architectures 117
5.6 Results 122
5.7 Discussion 125

6 Process Optimisation 130
6.1 Introduction - Design for Manufacturability 130
6.2 Experimental Design 1133
6.2.1 Design Objectives 135
6.2.2 Design Templates 137
6.3 RS/1 141
6.4 Optimised Process Design 142
6.5 Optimisation Study 146
6.6 Summary 159

7 Parallel Implementation of Process Optimisation 163

7.1 Introduction 163

7.2 System Overview 165

7.2.1 Communication Procedures 170

7.2.1.1 Introduction 170

7.2.1.2 The TINY Communications Harness 173

7.2.1.3 The INMOS Alien File Server Protocol 178

7.2.2 Worker Process 182

7.2.3 Driver Process 190

7.2.4 Control Procedures 194

7.3 System Performance 197

7.4 Conclusions 203

8 Conclusions 207

Appendix A Reprints of Published Papers 213

If you do not understand a particular word in a piece of technical writing, ignore it. The piece will make perfect sense without it.

Abstract

The ability of semiconductor process simulation to analyse the physical effects of individual fabrication steps and their interaction within an entire process has gained increasing recognition within the industry. Simulation has been applied to the synthesis of nominal operating points and has offered substantial reductions in both time and expenditure when compared to experimental runs for this role. Semiconductor companies are also realising that both performance and manufacturability must be designed into new technologies from their inception. This concept of Design for Manufacturability (DFM) can be implemented by linking process simulation with statistically based experimental design and analysis tools. However, neither the software framework nor the underlying computational resource currently exist to provide the level of system integration required to support DFM within a commercial environment.

This Thesis first introduces a method for enhancing the performance of process simulation software by utilising the power of parallel computing offered by the INMOS transputer. A parallel implementation of the one-dimensional simulator SUPREM-II has been developed which demonstrates the computational performance that is economically attainable and readily scalable using this technology. The system has then been extended to provide a fully functional DFM environment by automatically integrating the parallel process simulation capability with the experimental design and analysis software, RS/1.

A review of parallel computing systems, semiconductor fabrication control, process simulation and experimental design/analysis is also provided to compliment the presentation of the original contributions outlined above.

The progress of science varies inversely with the number of journals published.

Publications

Publications by the author relevant to this work include:

W. J. C. Alexander, A. J. Walton, J. M. Robertson and R. J. Holwill, "The Implementation of Process Simulation on Transputers for the Production of ASIC's", Microelectronics '89, Brisbane, Australia, July 1989.

M. R. Kump, S. W. Mylroie, W. J. C. Alexander, A. J. Walton, "Use of Process Simulators to Assist in the Design of Processes for Manufacturability", Conf. Proc. IEEE Advanced Semiconductor Manufacturing Conference, 1990, pp15-21.

G. J. Gaston, W. J. C. Alexander, L. J. Clarke, A. J. Walton, "PARTICS: A Parallel Taskfarm for Integrated Circuit Simulation", Edinburgh Parallel Computing Centre Technical Report, EPCC-TR91-08, 1991.

G. J. Gaston, W. J. C. Alexander, L. J. Clarke, A. J. Walton, "A General Task-farming Tool : Its Application to Semiconductor Fabrication", European Workshop on Parallel Computing, Barcelona, March 1992.

Acknowledgements

I wish to thank all those people within the Edinburgh Microfabrication Facility and the Edinburgh Parallel Computing Centre who have contributed to this Ph.D. and in particular, the efforts of my supervisor Dr. A. J. Walton.

I would also like to thank Clive Dyson and the original applications engineering group at Inmos, Bristol for their encouragement at the start of this project. Thanks too must go to Godfrey Gaston and Lyndon Clarke for their concurrent wit and technical support. I am also very grateful to Edinburgh Air Charter for their assistance and Dr. R. J. Holwill for the hardware upgrades when things went horribly wrong.

Much appreciation also goes to my parents for their patience, to Henry for her unmentionable inspiration and to Robbie for setting such a competitive literary pace, all of which have assured the completion of this text. Finally, I am indebted to Dr. D. Renshaw for his counsel and endless support throughout the preparation of this Thesis.

Financial sponsorship of this project came from the Science and Engineering Council, and, in the form of a CASE award, from INMOS Ltd.

In an R & D orbit, only 2 of the following parameters may be simultaneously defined:- task, time and resources.

If a task is defined and its timescale for completion known, then its cost can not be estimated.

If the timescale and resources are clearly defined, then it is impossible to know what R & D task will be performed.

If a task is defined and allocated a budget, then it is impossible to predict if and when the goal has been met.

If fortune prevails and it is possible to accurately define all three parameters at once, then whatever is being dealt with does not lie within the realms of R & D.

Chapter 1

Introduction

The semiconductor industry has grown significantly in recent years and now holds a prominent and influential position within the sphere of international commerce. The fabrication of integrated circuits (ICs) forms an intrinsic part of this and must itself be regarded as an important, high volume industry, with an average of 8 ICs produced for every person in the world during 1990. Whilst this market has obviously expanded, so too has the level of technology and the economic performance required to service it in a commercially viable manner.

The manufacture of ICs is extremely capital intensive and requires an immense initial investment in production facilities before fabrication can even begin [1]. This is further compounded by the similarly high costs which are associated with the development of new processes [2] that is necessary to support diminishing device geometries. The recovery of such development costs is complicated by these continual advances in technology, which have tended to shorten product life cycles and demands the implementation of a production policy which realises a high yield of reliable ICs on schedule. Profit margins are forever being eroded by increasing competition and can not afford to be further sacrificed unnecessarily. The significance of market forces must not be underestimated, as can be seen from table 1.1, which illustrates the substantial loss of revenue which can be incurred by a delayed market entry.

As a consequence of these technical and financial considerations, semiconductor companies have become increasingly aware that new technologies must be designed with both performance and manufacturability in mind. This has prompted them to re-assess the manner in which new processes are developed and has seen the emergence of advanced methodologies specifically aimed at addressing the issues of *Design for Manufacture* (DFM) and *Design for Quality* (DFQ).

Delay (months)	Market Window (months) 6	12	18	24	30	36
1	24%	12%	8%	6%	5%	4%
2	44%	24%	16%	12%	10%	8%
3	63%	34%	24%	18%	15%	12%
4	78%	44%	31%	24%	19%	16%
5	90%	54%	38%	29%	24%	20%
6	100%	63%	44%	34%	28%	24%
7	100%	70%	51%	39%	32%	27%
8	100%	78%	57%	44%	36%	31%
9	100%	84%	63%	49%	41%	34%
10	100%	90%	68%	54%	44%	38%
11	100%	95%	73%	58%	48%	41%
12	100%	100%	78%	63%	52%	44%

Table 1.1. Lost revenue for late arrival to marketplace. Taken from [3].

1.1 Design Methodology

The goal of every manufacturing facility has been and still remains that of producing the right product for the right price at the right time [4]. What has changed, however, is the means by which this is achieved. There are in essence three interrelated factors which influence the semiconductor manufacturing process: quality, manufacturability and economics, with the latter usually having the right of veto. Figure 1.1 provides a graphical representation of the way in which the issue of quality control has evolved in recent times.

The use of in-line and post-process inspection alone is very wasteful and more importantly makes no effort to improve the inherent quality of the process. More recently, this has been supplemented on a wide scale by the use of Statistical Process Control (SPC) techniques [5] which attempt to control and minimise variabilities within the process. This might involve redesigning an offending process step or

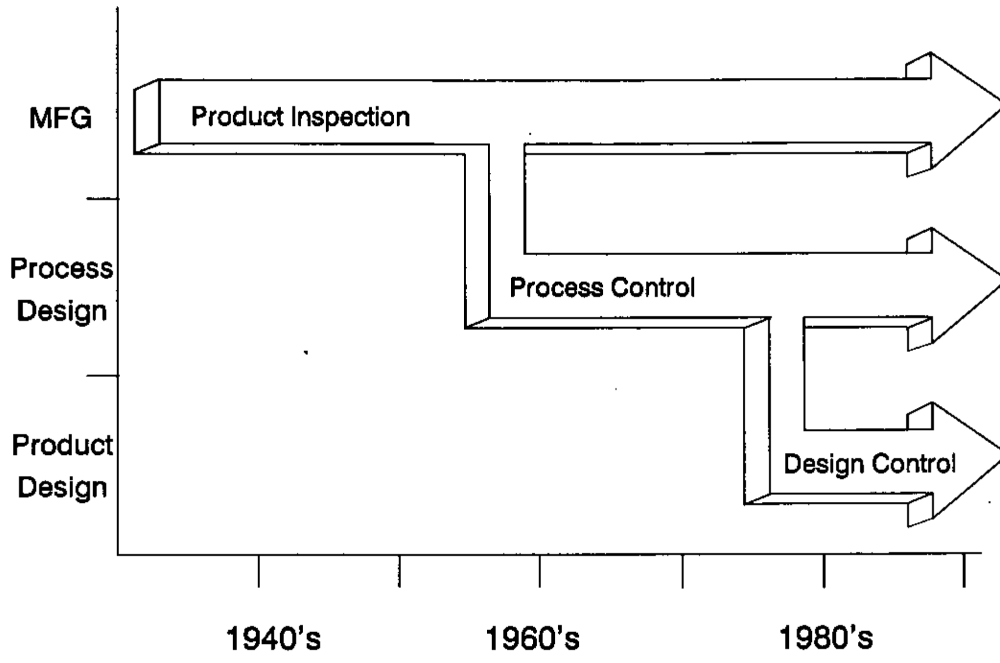


Figure 1.1. Evolution of quality control in manufacturing processes.

purchasing a superior piece of equipment, neither of which are particularly time efficient or cost effective. The logical extension of this is to design the product to be as insensitive as possible to these inevitable process variations and it is this concept which forms the basis of DFM.

The strategy employed under DFM is to concentrate investment at the design phase when alterations are most easily made and inflict minimal financial penalty. In the past, the stumbling block with DFM has been the large number of experiments required to characterise a process in terms of its sensitivity to random manufacturing variations. If however, a few split lot runs are used to calibrate a process simulator, then instead it may be linked with the relevant experimental design and analysis tools to provide the desired process evaluation. The use of Response Surface Methodology (RSM)[6] and Taguchi[7] analysis then enables

processes to be designed for optimal performance and manufacturability, with little or no loss of peak device performance, by the prediction and subsequent elimination of manufacturing problems before they occur.

Ultimately, wide spread acceptance of this powerful methodology by the semiconductor industry is dependent on evidence of its success and also the availability of a suitable system. The justification of DFM may be illustrated by figure 1.2 which compares the timing and quantity of design changes associated for a particular product line. It can be seen that the cost of the design associated with DFM is relatively lower, with the greatest proportion of the changes being incurred at an earlier stage. However, it is important to realise that there is also an increase in the perceived quality since the product is more stable when it is launched into the market.

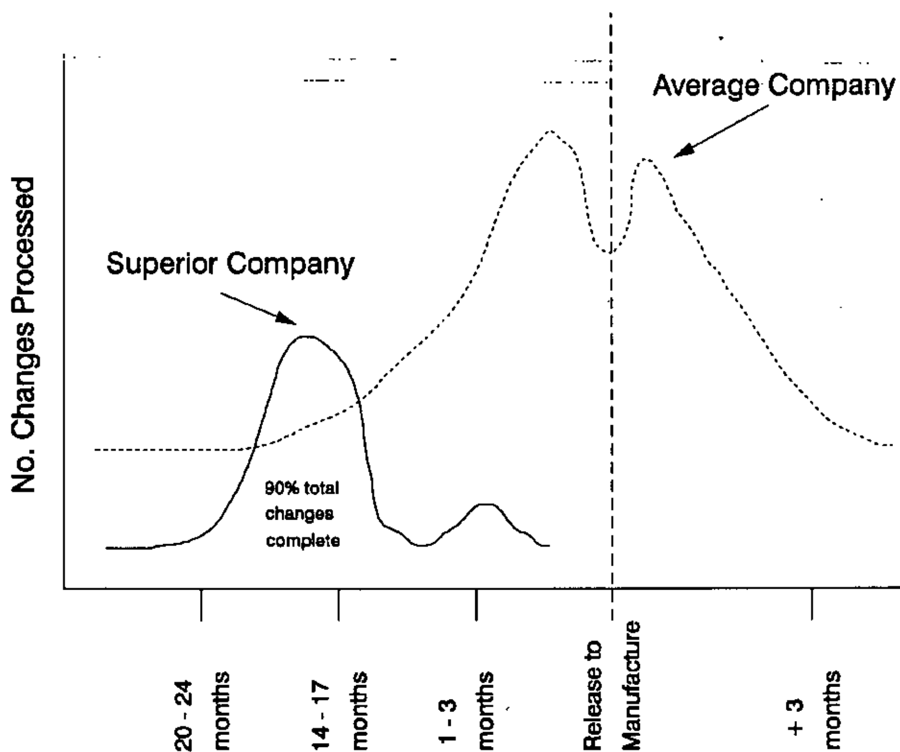


Figure 1.2. The effect of DFM on design changes. Taken from [8].

Although many of the statistical tools required to support DFM have been available individually for some time, the lack of an integrated system has not made the technique attractive in the commercial world. The investigations performed during this Ph.D. have therefore been aimed at redressing this situation.

1.2 Motivation

At the outset of this Ph.D. the investigation concentrated entirely on the application of parallel computing techniques to process simulation. The research goal being in the provision of a viable source of computation which would encourage the widespread use of process simulation required by the design of modern IC fabrication processes. During the course of this investigation the limited capabilities of the available parallel software tools became apparent, making the already complex task of customised parallelisation even harder.

Although process simulation has previously been employed to synthesise a nominal process operating point, this failed to recognise the natural process variations that occur during fabrication. However, the physical effect of process steps and their interaction within a complete process can be explored using process simulation. Therefore the emphasis of the research altered to combine this capacity with the previously proven power offered by a transputer based system.

The demonstration of a complete structured process design system was seen as an important step in affirming the role that DFM and DFQ have to play in ensuring the economic success and survival of the semiconductor fabrication industry.

1.3 Thesis Structure

This chapter has provided a brief introduction to the employment of structured design methodologies during the development of semiconductor processes. A more detailed discussion of IC fabrication control is presented in chapter 2 and proposes that the application of process simulation is suited to analysis of the complex interactions involved. A review of semiconductor process simulation is provided in chapter 3 and highlights the limitations which are often placed on its

widespread use due to the intense level of computation concerned. In anticipation of the solution offered by this research to the aforementioned dilemma, chapter 4 presents an overview of parallel computing concepts and systems before concentrating on one particular example of this technology, the INMOS transputer. In chapter 5, the work performed to apply the powerful transputer technology to one specific process simulator is described. This approach provides the computational resource required to support large scale simulation but still falls short of the structured design methods required. Chapter 6 illustrates how a combination of process simulation, experimental design and analysis techniques can provide a framework that promotes DFM and DFQ. The design of a transputer based system to support the simulations required by this strategy is described in chapter 7. Finally, some conclusions are drawn on the work which has been presented in this Thesis and a number suggestions for further work are briefly outlined.

References

1. W. C. Holton, J. Dussault, D. A. Hodges, C. L. Liu, J. D. Plummer, D. E. Thomas and P. F. Wu, "Computer Integrated Manufacturing (CIM) and Computer Aided Design (CAD) for the Semiconductor Industry in Japan", Japanese Technology Evaluation Program, National Science Foundation, Washington DC, USA, December 1988.

2. G. Moore, Invited Speech, Symposium on VLSI Technology, Honolulu, Hawaii, June 1990.

3. S. van Tyle, Electronics Design, Vol. 39, No. 12, June 1991, p111.

4. D. J. Miller and J. A. Fenimore, "The Application of Advanced Manufacturing Technology in Semiconductor Fabrication", IEEE/CHMT Int. Electronics Manufacturing Technology Symposium, Orlando, Florida, 1988.

5. P. K. Mozumder, C. R. Shyamsundar and A. J. Strojwas, "Statistical Control of VLSI Fabrication Processes: A Framework", IEEE Trans. Semiconductor Manufacturing, Vol. 1, No. 2, May 1988, pp62-71.

6. A. R. Alvarez, B. L/ Abdi, D. L. Young, J. Teplik, H. D. Weed and E. R. Herald, "Application of Statistical Methods to Computer-Aided VLSI Device Design", IEEE Trans. CAD, Vol. 7, No. 2, February 1988, pp272-288.

7. D. L. Young, J. Teplik, H. D. Weed, N. T. Tracht and A. R. Alvarez, "Application of Statistical Methods to Computer-Aided VLSI Device Design II: Desirability Functions and Taguchi Methods", IEEE Trans. CAD, Vol. 10, No. 1, January 1991, pp103-115.

8. "Concurrent Engineering" Motorola University Course Notes, Motorola Inc.

Under the most rigourously controlled conditions of pressure, temperature, volume, humidity and other variables, the process will do as it damn well pleases.

Chapter 2

Process Control in IC Fabrication

2.1 Introduction

Integrated Circuit (IC) manufacturing is a relatively new industry, having evolved since the development of the first IC produced in 1958 by Kilby [1]. Since its inception, this industry has expanded rapidly from the production of experimental devices into one of the most capital intensive world-wide manufacturing scenarios of the present day. The main impetus for this swift growth has come from three principal directions. Firstly, many powerful systems have been initially conceived and developed in software (e.g. Digital signal processing systems) but require much more advanced manufacturing technologies if they were to be produced to meet realistic size and cost constraints. Secondly there was the motivation within the industry to develop new technologies in order to maintain or acquire a leading edge commercially and technologically. Thirdly, there was initially a strong economy within this market sector to support such development, though formidable competition, investment and overheads have forced a much more cautious and structured approach in recent times. Throughout this evolution, there has been a limitation on systems development and production. This limitation is largely imposed by the available technology. Two restraints exist which oppose the driving forces mentioned above. Economic considerations can prevent the frequent refurbishment of production facilities often required to fabricate a new process as this may run to many millions of dollars. Secondly there are times when a new process could be implemented, but the market is not in a position to accept it.

The system performance available from a given technology is determined by the characteristics of the devices which implement the circuit functions. In turn the device performance is defined by the processing technology used in its fabrication. Thus, it can be seen that IC process technology is a prime factor governing the scope of the next generation systems. There is continual pressure to increase the functional performance of such systems. To satisfy this, device packing density

must be increased by reducing device feature sizes, which in turn requires the development of new or modified processes. As processes have evolved, so too has the complexity associated with their production schedules.

Each IC device and production technology has inherent physical characteristics which determine its performance and hence its appropriate application. The main technologies are based on silicon although materials such as GaAs are finding increased application in specialised areas. A more detailed description of the different technologies may be found in Hillenius [2]. At present and for the foreseeable future (5-10 years) the greatest volume of IC production will continue to utilise MOS technology and this thesis therefore concentrates on nMOS and CMOS processes.

This chapter first provides a brief description of the main process steps within an MOS fabrication schedule. The implementation and function of each step within wafer fabrication is discused along with its associated control and monitor parameters. Then the integration of these steps to perform a complete process is considered from a control and stability perspective. Finally the influence of step interactions on designing a manufacturable process is demonstrated.

2.2 Process Steps

The complexity of current semiconductor fabrication processes is very high due to both the number of individual steps involved and the interrelated nature of these steps. The overall objective of constructing functional electrical circuitry across the wafer surface with a high yield and low cost has remained constant. However, as technologies have advanced the tolerances and economic constraints placed upon the semiconductor manufacturing industry have been severely tightened. In order to meet these conditions, a full understanding of each process step and its role within the overall process must be gained.

2.2.1 Oxidation

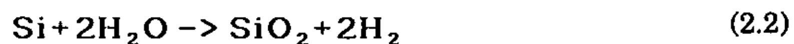
Silicon oxidation plays a very significant role during the IC manufacturing process. The properties of silicon dioxide, SiO_2 , and in particular its physical

attributes, have had a major influence on the development of planar processing technologies. SiO_2 performs many functions within the fabrication schedule: as a masking layer against implant or diffusion of dopants into silicon; as an inter- and intra-device isolation layer or region and to provide surface passivation. However, it is imperative that the formation of high-quality oxides is a controllable and repeatable operation to enable silicon IC manufacturing technologies to be successfully scaled down to submicron levels. For this, a deep and formal understanding of oxide formation, its physical and its electrical properties is required.

There are various methods of producing oxides, each having its optimum application; such as wet anodisation, vapour deposition, plasma anodisation and thermal oxidation. Although the latter is the most widely used and preferred technique for the growth of high-quality, low-charge density oxides, chemical vapour deposition (CVD) is also essential. (E.g. in the formation of an oxide layer onto a metalisation layer where no silicon exists to be oxidised by thermal process).

Thermal oxidation can be described by two chemical reactions, each pertaining to a different set of external conditions.

$$Si + O_2 \rightarrow SiO_2 \quad (2.1)$$



$$Si + 2H_2O \rightarrow SiO_2 + 2H_2 \quad (2.2)$$

The former equation is referred to as *dry* oxidation and the latter as *wet* oxidation. Which technique is chosen depends on the thickness and oxide properties required. Dry oxidation is generally employed for relatively thin oxides ($< 0.5\mu m$) and/or those that need a low interface charge density. In addition to this, HCl may be included in the reaction if there is a risk of sodium ion contamination. Wet oxidation techniques are applied for the growth of thicker oxides due to the increased growth rate at moderate temperatures.

Wafers must be cleaned to remove any organic and inorganic residues from previous processing operations or from handling around the facility. Such contamination can degrade the electrical characteristics of the devices and increase the defect density in the finished product. Typical cleaning procedures are described by Burkman [3]. After cleaning, oxidation can then be performed inside

a diffusion/oxidation furnace tube, which will be microprocessor controlled to provide repeatable sequencing, temperature control and gas flow (mass flow control). It is vital to maintain consistent process conditions throughout the oxidation to achieve uniform oxide growth. The microprocessor provides a versatile feedback control loop to continually adjust any necessary parameters as is required.

Dry oxidation is a relatively simple operation using microprocessor controlled equipment and produces a slow rate of oxide growth at moderate temperatures (c. 950°C). Modification of the ambient parameters may lead to enhanced oxide growth rates, for example raising the temperature will result in a thicker oxide for the same time. However, such a move will considerably alter the redistribution of dopants and this fact must be considered. Alternatively an increase in pressure ambient will positively affect the growth rate, and this time the dopant redistribution will be more or less unaltered.

The wet ambient for wet oxidation can be produced by one of two techniques, namely pyrogenic or steam oxidation. In the former high-quality steam is generated by the direct combination of H_2 and O_2 within the furnace tube with microprocessor controlled gas flow. Steam oxidation requires that oxygen be bubbled through hot water before entering the furnace. Although both methods provide an enhanced oxide growth rate in comparison to dry oxidation, concern for water vapour purity tends to view the steam technique less favourably due to the possible contamination within the bubbler system. As with dry oxidation, growth rates may be enhanced by the use of elevated temperatures and pressures.

So far consideration has only been given to enhancing the oxide growth rate. However as device geometries shrink to one micron and submicron levels, there is a necessity to produce very high quality thin oxides in the 50-200 Å range. These have a much greater sensitivity to structural defects and underlying non-uniformity. Both wet and dry oxidation techniques can be modified to produce thin oxides. For dry oxidation a combination of reduced temperatures and pressures has been successfully used to overcome the problem of the initial rapid growth of around 200 Å. For wet oxidations it has been found that consistent low growth rates can

achieved at lower temperatures in a high pressure steam ambient. As will become apparent in the next section, which discusses the diffusion of dopants, the use of lower temperatures is essential to reduce the lateral impurity redistribution in small geometry devices. Rapid thermal processing is therefore becoming a much more popular technique and a review of this technology may be found in Sedgwick[4].

Monitoring the thickness of a grown oxide and its uniformity at both wafer and batch levels is performed using reflective/refractive techniques such as spectroscopy and elipsometry on the product wafers [5].

As can be seen from the brief description of the oxidation process there are many parameters which can influence the oxide growth rate and the resulting quality. The prime parameters are those of temperature, pressure, ambient conditions and gas / silicon quality. It is therefore paramount that their role in this process is fully understood so that a set of optimum conditions may be derived for any desired oxide. A development of this topic can be found in chapter 3.

2.2.2 Diffusion

The introduction and subsequent redistribution of impurity atoms within the silicon crystal structure is an important procedure in IC processing. The type and electrical conductivity of silicon can be modified by adjusting the dopant element, concentration and distribution. This principle is used to form the base and emitter of integrated bipolar transistors and to form the source/drain regions of field effect transistors and dope polysilicon in MOS technologies. Therefore it is vital that diffusion of dopants is controllable, uniform and reproducible if device specifications are to be met.

Dopants are commonly introduced into silicon by one of the following methods:

(1) diffusion from a chemical source in a vapour phase at high temperature.

(2) diffusion from a doped-oxide source.

(3) and diffusion and annealing from an ion-implanted layer.

The choice of technique is dependant on the desired impurity profile and surface concentration. Two distinct doping profiles can be achieved. These depend on the surface concentration of the dopant during its exposure to thermal operations, which take place in furnace tubes under similar conditions to those used for oxidation. The use of ion implantation of impurity atoms will be considered separately in §2.2.3.

Diffusion steps that are performed with a constant surface concentration are normally referred to as *pre-deposition* steps. These are normally carried out at temperatures around 900-1000°C for 30-60 minutes and result in a complementary error function (erfc) distribution of the dopant.

$$C(x,t) = C_0 \operatorname{erfc}\left(\frac{x}{2\sqrt{Dx}}\right) \pm C_B \qquad (2.3)$$

C_B = Background doping concentration

C_O = Surface doping concentration

D = Diffusion constant

Dopant sources used for pre-deposition can be solids, liquids or gases. Phosphorus and boron are generally deposited in the form of their respective oxides, P_2O_5 and B_2O_3, which react with silicon to release the dopant atoms. Wafers of phosphorus oxide or boron nitride are interleaved with the product wafers to provide a uniform distribution across the surface. Arsenic compounds, due to their excessively toxic nature, are usually deposited by ion implantation.

Once the calculated quantity of dopant atoms has been introduced to the silicon surface, it is then necessary to redistribute them to give the required junction depth. This process called *drive-in* is generally performed at temperatures of 1000-1200°C in either inert or oxidising ambients, and results in a gaussian distribution profile.

$$C(x,t) = \frac{Q}{\sqrt{\pi D t}} \exp\left(\frac{-x^2}{4Dt}\right) \qquad (2.4)$$

Q = Number of impurity atoms deposited

This high temperature process not only disperses the dopant atoms but also electrically activates them (i.e. places them in substitutional sites within the silicon lattice [6]) and the technique is referred to as *annealing*.

Diffusion of dopants through not only single-crystal silicon but materials such as polysilicon and silicon dioxide also needs to be considered. Polysilicon is used both as a conductor between semiconductor layers and for self-aligned gates. Boron, phosphorus or arsenic doping of polysilicon is frequently used to reduce the resistivity of polysilicon. Silicon dioxide is often used to provide barrier regions and thus enable the silicon to be selectively doped. The diffusion of most dopants through silicon dioxide is fortunately comparatively slow and thus the use of a suitably thick layer of oxide will prevent doping of the substrate where this is not desired. A detailed discussion of dopant distribution in silicon is given by Fair [7].

It has been assumed so far that any redistribution of the dopants has been caused by vertical diffusion alone (i.e. normal to the silicon surface), however significant lateral diffusion will also be associated with any high temperature processing performed. At large device geometries ($>5\mu m$) one-dimensional analysis is sufficiently accurate that such lateral effects may be considered to be negligible. Once geometries approach $1\mu m$ then lateral analysis is vital. For example, when fabricating short channel MOS devices, it is imperative that the source and drain diffused regions do not meet during or after the drive-in, otherwise a short circuit will result. Depending on the precise drive-in conditions the lateral diffusion will occur for approximately 75% of the vertical diffusion extent. Such issues emphasise the need for analysing process steps in at least two dimensions in small geometry VLSI devices.

Another problem associated with the pre-deposition of dopants is that it tends to result in high surface concentrations which in turn means that the formation of shallow junctions is difficult to perform.

Monitoring diffusion operations involves the measurement of the doping profile, junction depth and sheet resistance of the wafer once the process step has been completed. Analysis of the dopant concentration profiles may be performed

by use of (i) the spreading resistance technique [8], (ii) the Secondary Ion Mass Spectrometry (SIMS) [9] or (iii) capacitance-voltage (CV) [10] technique. In general the analysis of the doping profiles is only performed while a process is being developed or characterised.

Junction depths can be measured by the bevel and stain technique where a wedge is ground from the wafer and subsequent staining of the exposed sample allows the p- and n- type materials to be identified. The problem with this test is that, being both slow and destructive, it can only be applied to test wafers.

The use of four-point probes or Van der Pauw techniques [11] for the measurement of sheet resistance provides a quick and non-destructive method, which can be performed in-line on product wafers.

The deposition and diffusion of impurities in silicon are both dependant on temperature, time and concentration and, if uniform and reproducible diffusion profiles are to be achieved, then the influence and control of these parameters must be understood. Of the three variables, concentration is the least versatile if uniformity is not to be sacrificed. Henry's Law relates the dopant concentration in the gas phase to the surface concentration [12]. Thus altering the partial gas pressure of the dopant will enable the surface concentration to be varied up to the point where the solid solubility of the dopant under the deposition conditions is reached. Therefore to achieve reproducible diffusion result simply it is necessary to ensure that the surface concentration be maintained at this solid solubility level.

The control of and dependence on temperature and time are generally the same as for oxidation since the drive-in stage is usually associated with the formation of field or passivation oxides. Also the wafers will be exposed to temperature ramp-up and ramp-down at each end of the process and this must be considered along with the respective timing.

2.2.3 Ion Implantation

Ion implantation is a highly controllable and reproducible method of introducing dopant atoms into semiconductor materials and provides an alternative to

the dopant diffusion technique discussed in §2.2.2. It has now become the predominant technique used in the fabrication of MOS ICs, especially at VLSI geometries (<3μm), due to the degree of control required at these device sizes.

Ion implantation involves the vaporisation of dopant gas molecules into charged fragments, which are then selectively accelerated to a specified energy. The ions are collimated into a focussed beam which is electrostatically deflected so as to scan the entire wafer surface. The process is accomplished in a high vacuum environment (c 10^{-6} Torr) to minimise ion scattering.

The dopant accepting areas of the wafer can be defined by photoresist, silicon oxide, silicon nitride or polysilicon masking layers. The thickness of the mask required to prevent penetration depends on the material used and the ion beam energy. For example, a 100KeV boron implant requires the following minimum mask thicknesses : 0.4μm Si_3N_4, 0.55 μm SiO_2, 0.7μm photoresist. To first order, the energy of the ion beam defines the depth of penetration and the beam current defines the dose implanted, and this gives rise to a high degree of control in the resulting total dose, junction depth, doping profile and uniformity.

As a result of superior controllability, it is possible to perform accurate and reproducible low dose pre-depositions ($<10^{15}$ atoms/cm^2) by the use of low beam currents. This is in contrast to chemical pre-deposition where the surface dopant concentration was maintained at solid solubility to achieve reproducibility. These low dose implants (c. 10^{12} atoms/cm^2) are typically used in MOS processing to perform threshold voltage adjustment. Higher dose implants (up to 10^{18} atoms/cm^2) are also used to provide accurate source/drain implants and well implants. Implantation energies vary from 1KeV to around 1MeV, giving rise to average dopant depths running from 100Å to 10μm. This means that this technology is capable of producing buried conductors and insulators.

There are a number of problems, including nuclear scattering damage to the silicon lattice, ion channelling, recoil damage and surface charging, which can effect the yield of implanted devices. Lattice damage is caused by the impact of the incident dopant ions displacing target atoms in the wafer and can partially amorphise the

implanted region. Such atomic disorder can be annealed out or prevented by applying a thin layer of oxide, nitride or photoresist before implantation. The regular atomic structure of a crystalline target means that in certain planes there are open spaces through which the implanted ions can travel without any significant scattering. Consequently, they will travel much further into the lattice before coming to rest as compared to an amorphous target and the final device doping profile may well vary considerably from that intended, unless this effect is taken into consideration. This effect can be avoided by tilting the wafer so that the ions strike the surface at approximately 7°-10° from the normal, at which angles the lattice appears like an amorphous sample. Recoil damage is concerned with the displacement of target atoms from one layer into the neighbouring layer, in the case when the target consists of two layers. In such instances, it is typically oxygen or nitrogen atoms which become deposited into the silicon from a thin oxide or nitride surface layer. High surface doses of oxygen can seriously degrade carrier mobilities. However recoil mixing can also be used positively to introduce dopant atoms for which no ion implantation source is readily available but that can be deposited in a thin film. If the wafer is not well grounded, it can charge up and the resulting field will distort the ion beam leading to irregularities in the dopant distribution. By directing a low-energy beam onto the wafer surface, these charge build-ups can be neutralised.

Depending on the energy of the incident ion beam, the wafer may be subjected to kinetic heating. This can be beneficial in providing a self-annealing effect to counter the lattice damage as it occurs though, if particularly high energies are employed, it may also result in parasitic thermal diffusion. Although ion implantation places dopant atoms into silicon, they must be electrically activated before they will alter the characteristics of a functional device. The wafers are *annealed* in furnace tubes in an inert ambient at c. 900°-1000°C,for times of the order of tens of minutes. Rapid Thermal Annealing (RTA) is also used as an alternative technique for annealing and encompasses a variety of methods, which involve heating the

wafers for times between hundreds of seconds down to a few nanoseconds, thus enabling lattice repairs with minimal diffusion. The heating can be provided by high-energy laser pulses, electron beams or tungsten-halogen lamps.

Monitoring and evaluating the quality of ion implantation processing can be done by the same family of techniques that are applied to thermal diffusion (see §2.2.2). In addition thermal wave analysis can provide information on the extent of crystal damage and has the advantage of being a non-contact, non-destructive method.

The use of ion implantation in VLSI technologies is continuing to expand as new applications are found, which require the level of control, reproducibility and versatility to scaling which it offers. The major drawback it poses, in comparison to thermal diffusion, is that of low throughput. This is being ameliorated by the automation of the wafer load, unload and transport operations. Furthermore, it is most important to refine the control methodologies employed in order to incorporate real-time feed-back capabilities.

2.2.4 Deposition

During the course of fabricating a particular IC technology, many materials are deposited as films to provide electrical conduction or insulation regions in the device structure. Deposited films include silicon dioxide, silicon nitride, polycrystaline silicon and aluminium based metallisation alloys. Commonly used methods for deposition in current fabrication processes can be classed as either *physical* or *chemical* deposition. Physical methods encompass evaporation and sputtering principles, whilst chemical methods cover the family of vapour deposition techniques (CVD). Whichever process is used, the resulting film must have good adhesive qualities, uniform thickness across devices and wafers, and consistent composition and structure. These attributes must be controllable and reproducible and preferably scalable so as to necessitate as few changes in processing methods and equipment when geometries shrink.

Evaporation processes involve the vaporisation of the film material from a solid or liquid source by thermal means and its subsequent condensation on the target surface. Thermal agitation is commonly provided by resistive heating for widespread coverage or by electron or laser beam for local heating. Evaporation is usually performed under a low pressure (c. 10^{-5} - 10^{-6} Torr) to enable the atoms to undergo a virtually collisionless path to the target. This technique is most effective for depositing low melting point materials such as aluminium and can achieve high deposition rates. However, problems arise when attempting to deposit alloys or high melting point materials by this method.

These difficulties can be overcome by the use of sputtering. This technique can be regarded as an extension of the evaporation method discussed, in which the source material is subjected to ion bombardment instead of thermal heating. Kinetic energy is exchanged and particles of the target material are ejected from the surface of its source and are deposited on the wafer surface. The incident species on the source material are usually heavy inert gas ions, the most commonly found being Argon. Ion generation can either be performed by glow discharge plasmas, most frequently using magnetrons, or by the use of ion beams. Magnetron sputtering produces a plasma glow of charged atoms which are then accelerated towards the target material dislodging the surface atoms which then travel to the wafer surfaces. High throughput, good step coverage and minimal substrate heating make this technique very popular. In the alternative method of ion beam sputtering, the inert gas ions are directed at the source material which then sputters on to the substrates as before. The deposited films which result are very pure, but the method suffers the major drawback of a low throughput, so is less commonly employed than that of magnetron sputtering.

Sputtering, in contrast to evaporation, can deposit films of metals, alloys, semiconductors and insulators in a highly controlled fashion making it a universally popular technique. This difference arises from the mechanical, as opposed to thermal or chemical, nature of vaporising the coating material. The limited damage to the wafer surface can be removed by a short low temperature anneal. Despite

the universality of this technique, the primary applications are associated with interconnect metallisation using aluminium or aluminium alloy films. Variation of chamber pressure, wafer temperature and substrate bias provide effective control of the resulting film thickness and deposition rate. Discussion of further sputtering techniques such as RF and Reactive sputtering may be found in [13]

CVD is extensively used in semiconductor processing to deposit pure silicon, its compounds and metallisation films on wafer surfaces. The source material is introduced to the reaction chamber in the gas phase. When this ambient is subjected to a high temperature a chemical reaction occurs which results in the desired film being deposited on the wafer surface. Most of the reactions are based on the oxidation of silane, SiH_4, or reaction of diclorosilane, $SiCl_2H_2$ with nitrous oxide, N_2O. CVD can be classed under one of three basic groups: atmospheric deposition (ATCVD), low pressure deposition (LPCVD) and plasma-enhanced deposition (PECVD). By introducing dopant species into the gas system it is possible to deposit layers with controlled impurity concentrations.

ATCVD is rarely associated with the fabrication of MOS device structures, but is of importance in bipolar processes. In this case it is used to deposit epitaxial silicon layers with the identical crystal orientation as the substrate. Such a process is typically achieved at temperatures around 1000° to 1200°C and results in poor thickness uniformity. It is also possible to selectively deposit expitaxial silicon by the use of silicon dioxide to block nucleation sites on the substrate [14]

Polysilicon, silicon nitride and silicon dioxide are deposited by the use of LPCVD which is performed at lower temperatures around 800°C. The precise temperature used depends on the chemical reaction chosen to produce the deposition material. LPCVD provides superior step coverage to both evaporation and sputtering, especially for metallisation layers, so it is favoured for formation of multi-layer interconnect. PECVD also has a faster deposition rate and produces improved uniformity of the layer thickness and composition than ATCVD.

PECVD is a complex process and is only used when low temperature processing is a necessity. The gaseous reactions occur in a plasma at around 150° to 350°C and silicon nitride and silicon dioxide can be deposited in this manner. Due to the complexity of this process it is very hard to control and provide reproducibility across wafer batches.

In order to provide accurate control mechanisms throughout deposition steps a combination of on-line monitoring of film growth and post-processing film thickness assessment must be used. Techniques such as mass spectrometry, elipsometry and partial pressure measurement of individual gas species may be used to monitor or infer the film thickness and deposition rate. Non-distructive testing of film thicknesses at the completion of the processing step use the reflective/refractive properties of the materials to provide data. Electrical four-point probe measurements and other destructive methods may be applied to test wafers to gain further information.

CVD techniques have a substantially higher capital cost associated with their equipment requirements when compared to evaporation or sputtering. However, as device geometries shrink beyond one micron, the required quality and throughput of device fabrication is placing a greater emphasis on the use of CVD techniques for layer deposition. For MOS technologies this means that in the future there will be an increase in the use of LPCVD and PECVD.

2.2.5 Lithography

Lithography is a multi-process step which enables circuit designs to be transferred from a software database onto the surface of the wafer. Each iteration of this step defines a feature, such as the gate or contacts, within the particular material layer once it has been grown or deposited. The main processes within a lithographic step are :

(i) Spin coating the wafer with a film of resist.

(ii) Soft bake of the resist (c. 100°C for 30 min).

(iii) Exposure of the resist to the image pattern.

(iv) Development of resist pattern to remove unwanted areas.

(v) Hard bake resist (c. 130°C for 30 min).

(vi) Perform desired action on patterned wafer. (e.g. source/drain implant or etching an oxide or nitride layer).

(vii) Strip remaining resist from wafer surface.

Resists can be classed as either *positive* or *negative*. The former becomes more soluble in developer after exposure whilst the latter becomes less soluble. Due to the nature of the chemical reactions that occur when a resist is exposed, positive resist enable higher resolutions to be attained [15] and are therefore more commonly used.

Many methods for wafer exposure have been developed for transferring the mask image onto the wafer surface, including optical, electron and ion-beam and X-ray techniques. Technological and economic factors will influence the actual choice, though the primary requirement is that the method must be capable of achieving the registration tolerance and resolution demanded by the particular product at an economic yield. Visible or ultra-violet radiation is most frequently used to expose the wafers through a glass/chromium or glass/emulsion mask. This technology is capable of resolving images down to the wavelengths used (c. 0.4μm) which is more than adequate for current feature sizes in commercial production facilities.

The significant parameters in assessing a printer are resolution, layer-to-layer alignment, throughput, depth of focus and cost. Within optical lithography, a choice of contact, proximity or projection printing techniques is available. Contact printing uses a 1:1 mask which is pressed against the resist during exposure and offers a high throughput for a relatively low capital cost. This is offset by the defects induced in both the mask and the wafer by the contact which significantly reduces the yield if masks are not frequently replaced or cleaned. Proximity printing provides a separation of around 25μm between the mask and the wafer which increases the mask life but reduces the resolution available due to Fresnel diffraction effects. Projection printing improves both the resolution and the mask life by projecting

the image onto the wafer through a system of lenses rather than using a shadow casting technique. This may be performed by a whole-wafer 1:1 projection system or by a direct-step-on-wafer (DSW) system which use a step-and-repeat technique to provide a 5X or 10X reduction of the image reticule onto the wafer surface at each die site. DSW offers very good resolution and superior layer-layer alignment than contact, proximity or whole-wafer projection systems, although it suffers from a lower throughput rate and higher cost. These drawbacks are justified for small geometry processing by the returned increase in yield.

Lithographic process monitoring is primarily associated with the measurement of linewidths in the developed resist which provides resolution and uniformity information. Optical microscopy or the use of Van der Pauw structures provide optical or electrical methods of attaining the relevant linewidth data. The remaining parameters such as baking times and temperatures and resist spin speeds are generally specified by the materials and equipment used and therefore stay constant once they have been established for a particular combination. Exposure and develop times are typically generated empirically by performing a matrix of tests over a range of probable values.

As process specifications incorporate tighter tolerances on layer-layer alignment, linewidth control and critical dimensions DSW printing will become the predominant optical technique. Their high capital cost and relatively low throughput is offset by the better processing quality and the resulting improvement in yield. Further developments of this technique, in the form of 'flash-on-the-fly' DSW printing, is progressing with the aim of improving wafer throughput. Electron, X-ray [16] and ion lithography all offer higher resolutions than optical methods and so will become important, once new technologies demand feature sizes less than approximately 0.4μm, the best optical systems can provide. Until this limitation is reached, these methods will remain peripheral to mainstream processing techniques.

2.2.6 Etching

After a pattern has been defined by a lithographic step in photoresist on the wafer surface (see §2.2.5), it must be transferred to the material layer itself. This requires the removal of the unwanted regions of the layer (i.e. those not covered by resist in the case of a positive resist). This is most commonly performed by an etch step. During the course of fabricating an MOS device many such lithographic and etch steps are performed to selectively remove areas of deposited or grown layers. Therefore techniques must exist to etch all the materials used throughout the fabrication process. For example, silicon dioxide must be removed to expose the silicon surface prior to an ion implantation or pre-deposition step and metallisation areas need to be removed to leave the desired interconnect structure. There are a number of methods available and the choice is dependant on the required feature size and profile as well as the material concerned.

Etching techniques can be broken down into two classes of process, namely *wet* and *dry*. Wet etching is the most fundamental technique used and involves immersing the prepared wafer in a solution of the appropriate chemicals for a predetermined time. Most liquid etches attack the layer surface in all directions at an equal rate and such an etch is said to be *isotropic*. Providing the feature linewidth and spacing is greater than the respective height then the unwanted horizontal feature narrowing can be compensated for in the mask dimensions once the error magnitude has been established. Wet etch performance may be controlled by variation of the chemical concentration, temperature, time and application of agitation. The choice of chemical and conditions depends on factors such as etch rate, selectivity, resist adhesion as well as the material to be removed. Wet etching also suffers from poor uniformity across wafers and although this can be improved by agitation, it typically requires the routine use of over-etch to ensure complete clearance of the relevant material. The simplistic nature of the process results in a high throughput which means that it is well suited to production environments. However, on reaching VLSI geometries where feature heights are comparable to their linewidths and spacing, mask compensation is no longer applicable to counter

the isotropic nature of the etch. The routine over-etches also demand a high selectivity with respect to both the mask and the substrate to ensure neither are subjected to unacceptable damage. For these reasons, anisotropic etching techniques are a prerequisite in VLSI device processing.

Dry etching techniques are based on the ability of a plasma discharge to fragment and ionise a relatively inert input gas, producing species which are highly reactive in both gas and solid phases. When these species combine with any exposed surfaces volatile products are formed, which are subsequently vented out of the reaction chamber. Careful choice of the plasma composition enables etch processes to be defined for all materials commonly found in semiconductor wafer fabrication. Highly directional motion of the reactive species can be induced by the use of r.f. power to excite the plasma, thus producing very anisotropic etches. Consequently, dry etching techniques are most appropriate for defining critical VLSI device features such as MOS gates.

There are two classes of dry etchers, termed plasma or reactive ion etchers, which differ in the polarity of the electrode that supports the wafers immediately below the plasma. In Reactive Ion Etching (RIE), the wafers are held on the r.f.-driven electrode. RIE is performed with lower operating pressures and higher energy ions than plasma etching and consequently has a higher etch rate making it preferential in production environments. However, it is found that within dry etching there is a trade-off between the selectivity and anisotropic performance [17]. Selectivity arises from the chemical reactions occurring and favours the higher pressure ambient of plasma etchers. On the other hand, the presence of higher energy ions within the low pressure RIE ambient increases the degree of anisotropic etching, but also reduces the selectivity by the resulting physical sputtering. By sharing the r.f. power between both electrodes, a blend of etch characteristics can be produced for specific process requirements and such a machine is known as a "flexible" etcher [18].

For both classes of etcher, the primary control parameters are those of r.f. frequency and power, input and output gas flow rate and the chamber pressure. The mechanisms involved in dry etching are very complex and as yet are not fully understood, which makes the etch rate difficult to control. As with all process steps it is vital to maintain good intra- and inter-wafer uniformity. Typically, dry etching is performed on a batch basis with the chamber being vented to atmospheric pressure for the load/unload operations. This reduces the overall throughput capability and also permits the adsorption of contaminants which may alter the etch rates as well as introducing particle defects onto the wafer surface. As a result, single wafer dry etching, where the wafers are automatically loaded and unloaded through a load-lock, is often used [19]. This provides a cleaner reaction ambient and, with only intra-wafer uniformity being optimised, should provide enhanced etch performance.

Wet etch rates are not generally monitored, the chemical immersion time of the wafers being empirically determined with visual confirmation of etch completion. However, in dry etching end-point detection by the use of optical spectroscopy or mass spectrometry can be implemented to monitor and control the progress of the etch. These techniques do not guarantee complete and uniform removal of all the required material and so a minimal over-etch period is usually included.

Within VLSI device fabrication, dry etching techniques are superseding wet etching with the exception of large feature processing. The necessity for improving yield and throughput whilst maintaining consistent inter-wafer etching advocates the use of single-wafer techniques. This is further supported by the ability to automate wafer movement and provide in-situ process monitoring.

2.3 Process Control and Design

The production of semiconductor ICs is an inherently serial process, with several hundred individual steps being performed in order to arrive at the desired circuit implementation in the silicon wafer. Each of these steps, as discussed in §2.2.1 to §2.2.6, can significantly influence other steps in the fabrication sequence. For example, any step involving a raised thermal ambient will cause redistribution

of any existing dopants within the silicon structure. Likewise, the presence of an oxide or nitride region will prevent the ingress of dopant atoms to areas shielded in this way. There are also more subtle effects such as the existence of oxide regions which promote enhanced diffusion of dopant atoms in the silicon beneath.

Before any step can become part of manufacturing process, careful consideration must be given to its control parameters so that the resulting actions fit in with the *complete* process schedule. A corresponding set of monitor parameters also needs to be defined, so that post-processing verification of the step performance can be made by test measurements on the final product and/or test wafers. The data returned from these measurements will be used to monitor the processing quality and provide process control feedback.

This simple control methodology enables deviations from process specifications to signal the need for an investigation into the current process parameters. In the short term, this can indicate where a temporary alteration to processing conditions should provide corrective action for future batches and on a longer term basis, it points to where more robust and permanent modifications need to be made. However, this procedure can be very difficult to implement, firstly in that it may not be easy to pin-point the source of variation and secondly, re-tuning the process can be a highly time consuming exercise. In either case, the time and yield lost during the investigation and subsequent rectification will represent a large financial loss.

In a production , as opposed to a research, environment where there are many products and different batches within a product line to be tracked, the volume of engineering data generated by a facility is vast. The distribution and analysis of this data, if performed efficiently, can provide substantial benefits to the overall production management. Equally, it can be catastrophic if inefficiency or misuse persist. Computer Aided Manufacturing (CAM) software packages such as COMETS(WORKSTREAM)® and PROMIS®, have been specifically designed for the manufacturing industry and are widely used to provide extensive data management services throughout the entire facility for engineers and management

alike. Although such systems enable process engineers to track, monitor and control processing, they only provide one half of the ultimate requirement for ensuring that accurate processing is achieved consistently. The other half can be obtained by investigating *a priori*, the effect which any excursions from the specified parameter levels will have on the final device characteristics. In this way, an inherently stable process can be designed which will be less sensitive to parametric variations and therefore return a significantly higher yield.

This process design concept requires that the variation and sensitivity of each output parameter to changes in the input conditions must, as a minimum be qualified and preferably be quantified. Over the past two decades a wealth of process models have been developed which describe the physical changes that occur during each of the major process steps discussed in this chapter. These software models are usually linked together within a single package to provide the means to simulate an entire fabrication process or just those steps of interest for a given problem. Chapter 3 discusses some of the models available, their merits and drawbacks and their growing importance in current and future semiconductor processing. Obviously, if the effect of input parameter variations is to be quantified, a number of these simulations will have to be performed over the desired range of possible conditions. Although some process parameters only affect a single device characteristic, more commonly they influence numerous characteristics and also interact with each other in complex ways. Therefore a rigourous and structured analysis of the simulation results is essential in order to extract an informative and quantitative set of relations between the process parameters and the device characteristics. The application of experimental design and analysis techniques to this field is covered in detail within chapter 6. The combination of process simulation and experimental design provides a very powerful and vital route to achieving a *structured* process design in which potential control problems are indentified and hopefully avoided before manufacture.

As device features diminish in size and processing tolerances tighten there is a greater need to fully understand the interaction of process steps with each other and to consider many of the physical characteristics which were of little consequence at larger geometries. Coupled with the increasing trend towards the use of Application Specific ICs (ASICs) and the emergence of low volume IC production, structured process design and control has never been more important

2.4 Summary

In this chapter, on overview of silicon wafer fabrication has been given in the form of an outline of the basic steps employed within a typical process. These steps, described in terms of how they are performed and their function, have been presented to emphasise their control and monitoring requirements. The concepts of process control and process design have been introduced and their importance demonstrated, especially with demands for ever-increasing yield and reliability whilst scaling device geometries into the submicon range.

References

1. J. S. Kilby, "The Invention of the Integrated Circuit", IEEE Trans. Electron Devices, ED-23, p648, 1976.
2. S. J. Hillenius, "VLSI Process Integration", VLSI Technology 2/e, ed S.M. Sze, McGraw-Hill, 1988.
3. D. Burkman, "Optimising the Cleaning Procedure for Silicon Wafers Prior to High Temperature Operations", Semiconductor Int., 4, 103, 1981.
4. T. O. Sedgwick, "Rapid Thermal Processing: How well is it doing and where is it going?", Mat. Res. Soc. Symp. Proc., Vol. 92, 1987, pp3-12.
5. A.C. Adams, "Dielectric and Polysilicon Film Deposition", VLSI Technology 2/e, edited by S.M. Sze, McGraw-Hill, 1988.
6. M.D. Giles, "Ion Implantation", VLSI Technology 2/e, ed S.M. Sze, McGraw-Hill, 1988, p355.
7. R.B. Fair, "Impurity Doping Processes in Silicon", ed F.F.Y.Yang, North-Holland Press, Amsterdam, 1981.
8. R.G. Mazur, D.H. Dickey, "Spreading Resistance Technique for Resistivity Measurements on Silicon", J. Electrochem. Soc., Vol 113, 1966, p255
9. J.B. Bindell, "Analytical Techniques", VLSI Technology 2/e, "ed S.M. Sze, McGraw-Hill, 1988
10. E.H. Nichollian, J.R. Brews, "MOS (Metal Oxide Semiconductor) Physics and Technology", John Wiley and Sons, New York, 1982
11. L.J. Van der Pauw, "A Method of Measuring Specific Resistivity and Hall Effect of Discs of Arbitrary Shape", Philips Res. Repts., Vol 13, 1958, pp1-9
12. R.B. Fair, "Diffusion and Ion Implantation", Semiconductor Materials and Process Technology Handbook for VLSI and ULSI, ed G.E. McGuire, Noyes Publications, New Jersey, 1988
13. J.A. Thornton, "Physical Vapor Deposition", Semiconductor Materials and Process Technology Handbook for VLSI and ULSI, ed G.E. McGuire, Noyes Publications, New Jersey, 1988.

14. K.E. Bean, "Chemical Vapor Deposition of Silicon and Its Compounds", Semiconductor Materials and Process Technology Handbook for VLSI and ULSI, ed G.E. McGuire, Noyes Publications, New Jersey, 1988.

15. R.K.Watts, "Lithography", VLSI Technology 2/e, ed S.M. Sze, McGraw-Hill, 1988.

16. J. Lingau, R. Dammel, J. Theis, "Recent Trends in X-Ray Resists - Part 1", Solid State Technology, Vol 32, pp105-112, Sept. 1989.

17. R.J. Schutz, "Reactive Plasma Etching", VLSI Technology 2/e, ed S.M. Sze, McGraw-Hill, 1988.

18. L.M. Emphrath, "Reactive Ion Etching for VLSI", IEEE Trans. Elec. Dev., Vol 28, pp1315-1319, 1981.

19. W.C. Dautremont-Smith, R.A. Gottscho, R.J. Schutz, "Plasma Processing : Mechanisms and Applications", Semiconductor Materials and Process Technology Handbook for VLSI and ULSI, ed G.E. McGuire, Noyes Publications, New Jersey, 1988.

Any given program, when running, is obsolete.

Any given program costs more and takes longer.

If a program is useful, it will have to be changed.

If a program is useless, it will have to be documented.

Any given program will expand to fill the available memory.

The value of a program is proportional to the weight of its output.

Program complexity grows until it exceeds the capability of the person who must maintain it.

Chapter 3

Process Simulation

3.1 Introduction

The fabrication of silicon ICs involves an inherently complex set of operations, with a typical process comprising several hundred individual steps. In each new generation of IC, device geometries have been shrunk in both planar and vertical dimensions in order to meet the continual demand for greater circuit performance and packing density. With the reduction in feature sizes to micron and submicron dimensions, the development of new technologies and fabrication processes has become increasingly more difficult, time consuming and expensive.

The use of computer aided design (CAD) tools has emerged as a powerful way in which to assist in the development of the technologies required to approach these levels of circuit complexity. Such tools not only cover circuit layout, but also process, device and circuit simulation. Together these can provide a more or less complete simulation system for the development of semiconductor products. A large number of numerical and analytical models have been developed over the past two decades to simulate each of the steps within a fabrication process. These models are generally implemented as a set of modular subprograms which are controlled by a supervisory shell program. This software provides the facility to calculate the impurity distributions and structural features of a fabricated device from a sequence of process steps with their respective control parameters.

Computer based modelling of semiconductor fabrication was first introduced in the 1960's to attempt to predict the process dependencies associated with the fabrication of the high speed double-diffused bipolar technology [1]. However, it was not until MOS technologies began to dominate the market during the late 1970's and early 1980's that the importance of rapid development in process modelling and simulation was fully perceived and acknowledged.

The most commonly used simulators are one dimensional, modeling a 1-D section of the device structure normal to the wafer surface, such as depicted in figure 3.1. One dimensional models provide sufficient accuracy and information for large geometry processes, where the minimum feature size exceeds 3μm. These models have been continually researched and improved since the introduction of process modelling and thus are much more 'mature' than their two dimensional counterparts.

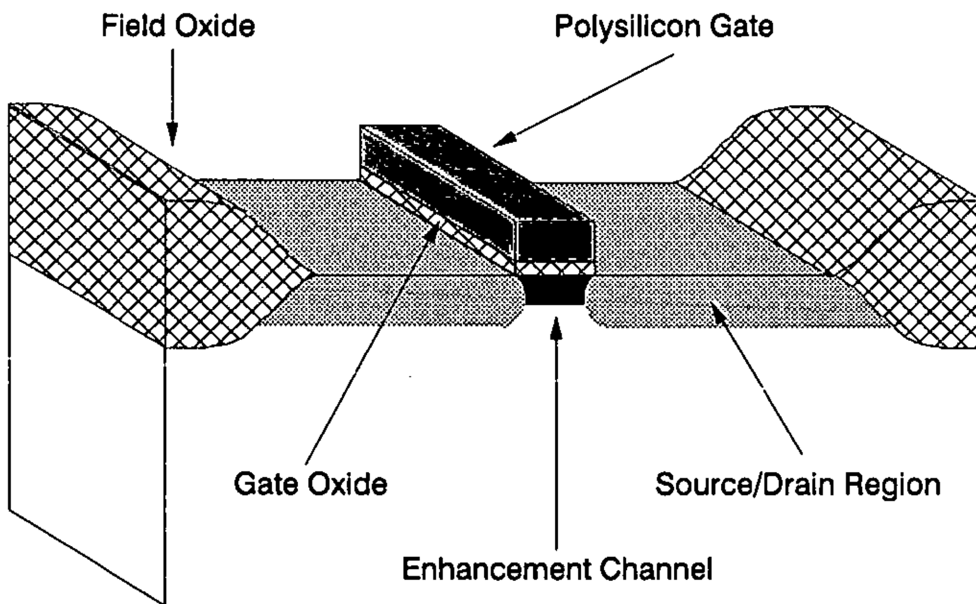


Figure 3.1. Section through an nMOS enhancement mode transistor.

However, once feature sizes begin to drop below the 3μm level, it becomes important to model the two dimensional effects that occur during the wafer processing, such as lateral dopant diffusion under field oxides or gate regions. The models used within 2-D simulators are less comprehensive than found in a 1-D system, but the solution times are still much longer (typically by a factor of 70-500 [2]). It is also anticipated that for accurate and constructive simulation of submicron geometries, the use of three dimensional modelling will be required [3],[4].

One major application of process simulation has been in the development of new fabrication processes. Traditionally, this was achieved by an iterative schedule of process modification, experimental fabrication and parametric test until the desired criteria were attained. With each cycle taking in the order of 6 to 8 weeks, this was a time consuming procedure. Economic pressures coupled with the increased demands being placed by the Application Specific IC (ASIC) market have led to a requirement for more rapid process development and design turn-round times. The need to simulate processes has become essential in low volume production if economic yields are to be achieved from the first batch of production wafers. Simulation not only provides a faster and cheaper development cycle, but it also enables process engineers to 'see' inside the device and observer features which are not directly visible in the physical structure, such as doping profiles. This feature can assist engineers in finding the cause of an anomalous device characteristic.

Another use which has emerged more recently is in the statistical control of VLSI fabrication processes. In its simplest form, simulators such as FABRICS [5] are used to assess the sensitivity of a particular process to the inherent variations found in the input parameters within a manufacturing environment. This methodology has also been expanded to incorporate this information into Computer-Aided Manufacturing (CAM) systems to provide additional on-line control functions [6]. This concept has also been illustrated with the incorporation of SUPREM II and COMETS™ [7] to provide *look-ahead* and *what-if* functions for process development and maintenance [8]. By offering the possibility of corrective processing to wafers which have been diagnosed as misprocessed at any point in the fabrication schedule, an opportunity is presented to improve the overall yield of the process.

Alternatively, a combination of process simulation and classical experimental design techniques can be used to observe the sensitivity of a given process to parameter variations and optimise that process for a particular set of performance goals. In this manner, any problems with the process can be ironed out before it

ever commences fabrication and an acceptable yield is assured for every batch of wafers so processed. The powerful nature of this technique is only now being realised with the considerable increase in design turn-round and economic pressures facing the semiconductor industry today.

3.2 One Dimensional Simulation

Simulation, by its very nature, can not *exactly* model *all* the events that are found in the real world. For the case of process simulation, a series of conceptual models have been developed which approximate the physics associated with each of the process steps described in the previous chapter. From these conceptual models arise a corresponding set of mathematical models which provide a description of the physical processes that are occurring within the semiconductor structure. These mathematical models fall into one of two categories, namely numerical or analytical. The latter can provide a rapid solution which is accurate only within a range of values for which it has been verified, whereas the former provides a more comprehensive approach but does consume a far greater computational resource. This issue is of such importance to the future of process simulation that it will be discussed in detail in §3.4. Such modelling techniques introduce limitations and hence errors, in the results attained from the simulations. However, if the models are applied judiciously, it is still possible to obtain quantitatively meaningful results.

Each mathematical model, be it numeric or analytic, contains a variable element in the form of adjustable coefficients so that it can be calibrated to a particular set of experimental data. Thus if it is important for a particular application that accurate quantitative results are obtained, then the process in question must be fully characterised to enable the relevant coefficients to be changed. This process characterisation comprises measurement or estimation of the model parameters. However for many applications, such as sensitivity analysis or trend analysis [9], an exact solution is not necessary and thus the default parameter values will provide the required data.

The largest simplification that can be made is to assess only the vertical dimension for the study of layer formation and dopant distribution. This is done by modelling a single one-dimensional section normal to the wafer surface at a time. In order to simulate the effect of the entire fabrication process over a whole device, a number of such sections through the major regions of the device will need to be simulated. As an example, consider figure 3.2 which illustrates a cross-section of an nMOS invertor structure with the significant regions requiring simulations highlighted.

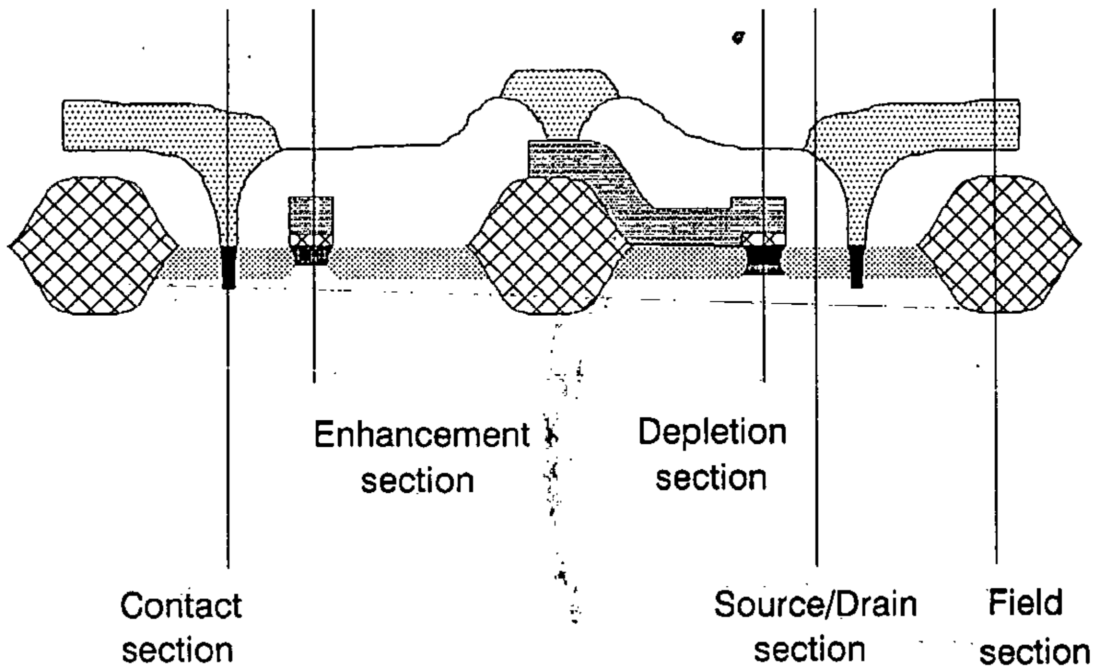


Figure 3.2. Cross-section of nMOS showing 1-D simulation regions.

The consequences of this simplification are that the models possess a much reduced parameter set and require considerably less computation. It is desirable for the models to be general-purpose and therefore theoretically physical based models rather than empirical models are favoured. Although this incurs substantially more computation, in the case of analytic models, it is too easy to operate outside the calibrated range of empirical models and thus incur large errors.

It is useful to point out a few of the general features of process simulators before examining one package in detail. In order to simulate any structure the following basic information is required:

- Initial wafer material type, orientation, and doping level.
- Definition of the semiconductor structure by means of a grid to provide a discretised simulation space in which the solution of the numerical models is performed.
- Specification of the necessary process steps in the order that they appear in the relevant fabrication schedule.

Each simulator has its own control language in which the structural initialisation and subsequent processing information is defined, as well as specification of the output format. An excerpt from a typical input file is shown in figure 3.3 and the link between the input syntax and basic equipment control parameters that would be found on an actual process runsheet, can be noted.

```
.
.
COMMENT         Grow pad oxide
DIFFUSION       TEMPERATURE=1050   THICKNESS=0.05   DRYO2

COMMENT         Deposit nitride masking layer
DEPOSIT         NITRIDE   THICKNESS=0.2   DX=0.02   SPACES=10

COMMENT         Grow field oxide under high pressure
DIFFUSION       TEMPERATURE=1000  TIME=30  STEAM  PRESSURE=5

COMMENT         Etch oxide and nitride layers
ETCH            OXIDE
ETCH            NITRIDE
ETCH            OXIDE
.
.
```

Figure 3.3 SUPREM III input statements taken from an MOS device simulation. Structural initialisation data has been omitted for clarity.

The output from the simulators provides physical, electrical and structural information pertaining to the device at the chosen point in the processing sequence. Such information includes:

- layer thicknesses and charge densities
- sheet resistivities and threshold voltages (where appropriate)
- graphical plots of the structure and doping profiles from the simulation space. This information may also be exported directly into certain device simulators [10]

The first process simulator to be widely used by industry and academia alike, was SUPREM II [11] and is capable of modelling both bipolar and MOS processes in one dimension. This simulator was developed at Stanford University, California, in the mid 1970's and came as the first of a family of process simulators. The models it incorporates are mature and well understood which, coupled with public domain source code, has continued its widespread use in both commercial and academic environments. These models use a mixture of physical and empirical solutions for ion implantation, oxidation, diffusion, epitaxy and etching. Default models for the distribution of boron, phosphorous and arsenic are included, but their use only pertains to silicon and silicon dioxide layers which is a severe limitation in the light of current technologies.

SUPREM III [12] was released in the early 1980's to replace SUPREM II and has substantially greater capabilities. The most important of which must be that simulated structures may contain up to ten material layers selected from silicon, silicon dioxide, polysilicon, silicon nitride and photoresist or user-defined materials. This feature, coupled with a revised set of physical models for the major process steps discussed in the previous chapter, has enabled SUPREM III to model most current process technologies. Table 3.1 provides a brief comparison of SUPREM II and SUPREM III.

SUPREM II	SUPREM III
2 layers	10 layers
default material for implant & diffusion	default material for implant & diffusion
silicon silicon dioxide	silicon silicon dioxide polysilicon silicon nitride aluminium
no additional material types	additional materials may be defined
3 dopant types	4 dopant types
default dopants antimony arsenic boron phosphorous	default dopants antimony arsenic boron/BF_2 phosphorous
oxidation of silicon	oxidation of silicon polysilicon silicon nitride

Table 3.1. SUPREM II versus SUPREM III

The ensuing sections provide a brief discussion of the scope and limitations of SUPREM III's one-dimensional models and reflecting the relationship between its control language format and typical equipment control parameters for each process.

3.2.1 Oxidation

The rate of SiO_2 growth has traditionally been described by the linear-parabolic Deal and Grove equation [13], the basic form of which is given in equation 3.1. Within SUPREM III, this has been further developed to account for the oxide growth on both silicon and polysilicon [14].

$$\frac{X_o^2 - X_i^2}{B} + \frac{X_o - X_i}{B/A} = t \qquad (3.1)$$

where :-

X_o	=	oxide thickness at time t
X_i	=	initial oxide thickness (if any) at t = 0
B	=	parabolic growth rate
B/A	=	linear growth rate
t	=	time

The parabolic and linear growth rates are given in their general form by equations 3.2 and 3.3 respectively. When solved with SUPREM III, terms for wet, dry or HCl ambients, pressure, substrate orientation and rapid initial growth are also included [15].

$$B = 2.D.C^*/N_1 \qquad (3.2)$$

$$B/A = \frac{C^*/N_1}{(1/k_s) + (1/n)} \qquad (3.3)$$

where :-

D	=	Oxidant diffusion coefficient in oxide
C^*	=	equilibrium oxidising species concentration
N_1	=	number of oxidant molecules per unit volume
k_s	=	oxide / ambient interface reaction rate constant
n	=	Si / SiO_2 interface reaction rate constant

All the factors which influence the growth of an oxide are modelled physically with the exception of HCl which is empirically implemented since no quantitative

physical model yet exists. Due to the almost entirely physical basis of the oxidation models, they are capable of simulating many different process conditions, ranging from short gate oxide growths to long isolation oxide formations.

In order to invoke an oxidation step from within SUPREM III, a DIFFUSION statement is used, whose parameters must include an oxidising ambient such as pure oxygen, pyrogenic steam or pure steam, as well as a suitable time and temperature. Figure 3.4 provides an example of a typical DIFFUSION statement in which the gas flows into the furnace have been directly specified. This option is most useful when relating a simulation to an actual set of manufacturing conditions.

```
.
.
DIFFUSION     TIME=30  TEMPERATURE=1000 F.O2=2 F.H2=2 F.N2=2 F.H2O=1
.
.
```

Figure 3.4 SUPREM III input statements for an oxidation in which the ambient gas flows have been specified as ratios.

The ability to simulate processing condition used in practice, is further enhanced by the option to supply additional parameters to indicate the growth of doped oxides and elevation or reduction in furnace pressure from the default atmospheric ambient. Both temperature and pressure may also be ramped at a user-specified rate.

3.2.2 Diffusion

One of the prime tasks of a process simulator is the generation of impurity doping profiles. Since great importance is placed on the resulting profiles, it is vital that the models associated with their evaluation are highly accurate. In the case of SUPREM III, diffusion models are present for the redistribution of antimony, arsenic, boron and phosphorous within polysilicon, nitride, silicon and silicon dioxide layers. For each of the listed impurities there is a set of physical models which are solved numerically.

Dopant redistribution occurs during every heat step used in the fabrication process, including oxidation, deposition and annealing. Thus the diffusion models within SUPREM III are required during the simulation of many different processes under a variety of ambients, temperatures and times.

Simulation of diffusion processes must therefore involve the solution of the complete one-dimensional continuity equation:

$$\frac{\partial C}{\partial t} = \frac{\partial}{\partial X}\left(D\frac{\partial C}{\partial X}\right) \pm \frac{q}{kT}\frac{\partial}{\partial X}\left(D\hat{C}\ \frac{\partial \phi}{\partial X}\right) \qquad (3.4)$$

where :-

D	=	effective diffusivity
C	=	total impurity concentration
$\hat{C}$	=	electrically charged impurity concentration

The potential ϕ is:

$$\phi = \frac{kT}{q}\ln\frac{n}{n_i} \qquad (3.5)$$

where n and n_i are the electron and intrinsic carrier concentrations respectively, at the diffusion temperature. There are four major diffusion mechanisms, namely vacancy-assisted, interstitial-assisted, interstitialcy and crowdion, which can contribute to the re-distribution of dopants in a semiconductor [16]. Within SUPREM III, equation 3.4 models the diffusion by the vacancy-assisted mechanism under non-oxidising conditions. In addition, account is also taken of high phosphorous concentrations, the presence of oxidising silicon or polysilicon surfaces, and transient diffusivity enhancement which may result from implant damage, for example. For multi-layer structures, dopant transport and segregation across material interfaces is also included in the relevant diffusion models. A more detailed discussion of the models used for diffusion in SUPREM III may be found in [17].

Explicit simulation of dopant diffusion through a semiconductor structure is performed using the DIFFUSION statement from the SUPREM III input language. As can be seen in figure 3.5, the processing conditions requiring specification are

those of temperature, time, and ambient. This includes the oxidising ambients as described in section 3.2.1 or an inert ambient such as nitrogen. As with oxidation, the diffusion rate will be affected by any temperature ramping that is present during the process step, but in contrast, there is no dependency on the pressure coefficient.

```
.
.
DIFFUSION     TEMPERATURE=1160   TIME=45   INERT
.
.
```

Figure 3.5 SUPREM III input statements for an inert diffusion.

It is most unusual under conventional manufacturing techniques simply to perform a diffusion without an associated oxidation or growth of a layer, and thus the models are generally called automatically during the simulation of these other process steps. However, the ability to explicitly simulate the consequences of the thermal component of a step, without its associated growth or oxidation, can be useful during process development and fault-finding.

3.2.3 Ion Implantation

Traditionally, initial dopant concentrations within a semiconductor structure have been obtained by either pre-deposition or ion implantation. Pre-deposition can be modelled as a conventional diffusion step with Dirichlet boundary conditions [18]. However, the fabrication of small geometry devices requires a far greater control of doping profiles than is available from a pre-deposition step and ion implantation also permits a wider range of impurity ions to be introduced into the structure. In addition, the fabrication of shallow or low concentration doped regions such as might be required in the formation of small geometry source and drain regions, with sufficient efficiency and reproducibility is only attainable by ion implantation. Dopants can also be implanted through multiple layers of a structure. Thus there is an almost universal use of ion implantation for this process step.

It is very important that the outcome of this step is accurately predicted, because the implanted profiles form the initial conditions for all subsequent diffusion processes. This operation is modelled by either an analytical or numerical technique depending on the trade-off between speed and accuracy that is acceptable to the end-user.

SUPREM II took the analytical approach and utilised a series of look-up tables based on the established LSS theory [19] to produce predicted range distributions for the implanted ions. On this basis of the LSS theory, the simplest resulting distribution is a symmetrical Gaussian curve with experimental data providing the projected range, R_p, and standard deviation, s, for a variety of ion-target combinations. Asymmetric profiles are found experimentally for many ions and are obtained by using the 3rd central moment for range information [20]. These distributions can be represented by 2-sided Gaussian profiles which are characterised by two standard deviations, s_1 and s_2 for arsenic, antimony and phosphorous. Boron requires the use of a modified Pearson distribution with an exponential tail added in order to account for the channelling effects. The major limitation of the basic LSS theory is that it assumes a homogeneous target, and hence is not applicable to the simulation of multi-layer structures. It has been reported [21] that the assumption of a homogeneous target has been removed, but this is not the case for the currently available simulators which incorporate this modelling technique. Additionally, LSS theory is unable to model recoil effects such as secondary implantation. The general analytical calculations represent an impurity distribution I(x) as shown in equation 3.5

$$I(x) = N \cdot [f(x) + f_t(x)] \qquad (3.6)$$

where :-

N	=	dopant concentration / dose from implanter
f(x)	=	normalised distribution (Gaussian / Pearson)
$f_t(x)$	=	exponential tail added to f(x) in silicon

Numerical modeling methods are based either on the Boltzman Transport Equation (BTE) or on Monte-Carlo (MC) techniques. The primary method incorporated in most of the available simulators at the present is based on the solution of the Boltzman transport equation from work by Cristel et al [22]. In this model, the scattering of the ions is described by changes in the statistical momentum distribution function. BTE *can* model recoil implantation effects and damage mechanisms, but like the analytical models, is based on an amorphous target and therefore can not explicitly model channelling effects.

Monte-Carlo methods on the other hand are based on the simulation of individual particles and their collisions in the target structure. By summing the nuclear and electronic stopping events in a large sample of ions (preferably 1000 or more), the range parameters for both primary and secondary recoils and any associated damage can be obtained. This methodology has the advantage that any ion/target combination can be modelled due to the entirely physical approach. Models for both amorphous and crystalline targets are available [23], thus allowing channelling effects to be included. Monte-Carlo simulations are highly computational when a sufficient number of samples are included to provide the necessary accuracy for small geometry devices.

SUPREM III offers revised analytical models incorporating Pearson IV/V/VI distributions [24] depending on the conditions pertaining to the step. It also has the numerical BTE method included, though in order to reduce the computational overhead the recoil and damage models have been omitted [25]. Implantations are invoked by the IMPLANT statement and require information on the dopant species, concentration/dose of the implanted ions, their acceleration energy and the solution method to be employed. Although any dopant may be used for the numerical solution, range characteristics only exist for arsenic, antimony, boron and phosphorous when utilising the analytic solution. Figure 3.6 provides an example of a typical IMPLANT statement from SUPREM III and indicated the close links between the simulation and fabrication parameter specifications

```
.
.
IMPLANT     BORON  DOSE=1E13   ENERGY=50   2-GAUSSIAN
.
.
```

Figure 3.6 SUPREM III input statements for a boron IMPLANT using a two-sided gaussian distribution function.

Analytical techniques are extremely efficient, but the simulation of small geometry devices requires the additional accuracy provided by the numerical BTE or MC techniques and has to be paid for in terms of the additional computational overheads.

3.2.4 Deposition

Modelling of deposition processes within most one-dimensional simulators is highly simplified, with the most common approach being to consider the step as the addition of an extra layer to the top of the current structure. Such a procedure may be justified by the fact that one-dimensional simulation does not include any detailed topographical information.

SUPREM III contains default models for the deposition of silicon, silicon dioxide, silicon nitride, polysilicon aluminium and photoresist. With the exception of polysilicon, there are no analytic or numerical models included to explicitly model the deposition processes, which enables a wide range of techniques to be nominally modelled by SUPREM III. Thus processes such as CVD, sputtering, resist spin-on and pre-deposition, as well as the presence of polysilicon gates or metal interconnect may be included in a simulation through the DEPOSITION statement. In order for this generalised method to operate, it is necessary to specify the desired end thickness or growth rate and time for the particular material layer. For low temperature steps, this simply results in the specific material being added to the current structure. In the case of high temperature steps, the diffusion models are also triggered in order to account for the motion of any dopants already present in the structure. The deposited layers may also be specified to contain impurity elements

such as antimony, arsenic, boron and phosphorous. For low temperature steps, this will result in a uniform dopant concentration throughout the new layer and a pre-deposition process is modelled by a high temperature step, where the specified dopant concentration will exist in the ambient gas.

The increasing role of polysilicon in IC circuits as a gate material and an interconnect has lead to the inclusion of a series of models for the growth of polysilicon films in SUPREM III [26]. In addition to the previously mentioned parameters requiring specification, either the ambient pressure or an initial grain size must be included. Segregation at the material interfaces, carrier trapping and grain growth are all included in these models.

Figures 3.7 shows examples of three different DEPOSITION statements from SUPREM III illustrating a low temperature deposition, doped silicon and a polysilicon deposition respectively.

```
.
.
DEPOSIT    NITRIDE    THICKNESS=0.2
.
DEPOSIT    SILICON    <100>  THICKNESS=0.1  RESISTIVITY  BORON=5
.
DEPOSIT    POLYSILICON    THICKNESS=0.5  TEMPERATURE=650  PRESSURE=1
.
.
```

Figure 3.7. SUPREM III input statements for nitride, doped silicon and polysilicon DEPOSITIONS.

As has been demonstrated, SUPREM III possesses a vastly simplified approach to deposition processes and the simulation parameters bear only a limited relevance to their counterparts in the fabrication cycle. More complex and comprehensive modelling is required when considering two-dimensional analysis and is available within most 2-D simulators as well as through separate programs such as DEPICT-1 [27], DEPICT-2 [28] and SAMPLE [29].

3.2.5 Etch

Etching processes within one-dimensional simulation are, like the deposition processes, highly simplified and in effect can be viewed as the direct inverse of deposition with no explicit treatment of the separate wet, dry or even ion beam milling etch techniques found in fabrication. Within SUPREM III, an ETCH step enables the removal of all, or part, of a specified material from the top of the current structure. Default models for the etching of silicon, silicon dioxide, silicon nitride, polysilicon, aluminium and resist are included and the presence of dopants within the layer has no effect.

Since there are no physical or empirical models associated with the simulation of an etch, it is necessary to know the outcome of a particular fabrication etch process in advance if it is to be modelled. Once this 'calibration' has been achieved, SUPREM III is then capable of simulating any etch process that might be performed in a fabrication process. As with deposition, both low and high temperature steps can be modelled, with the diffusion models being solved for any impurities present in the structure during a high temperature step.

As an example of the ETCH statement, figure 3.8 illustrates a simple oxide removal from an MOS gate structure, with figures 3.9 and 3.10 showing the structure before and after the etch.

```
.
.
ETCH      OXIDE  THICKNESS=0.2
.
.
```

Figure 3.8 SUPREM III input statements showing the removal of an oxide layer.

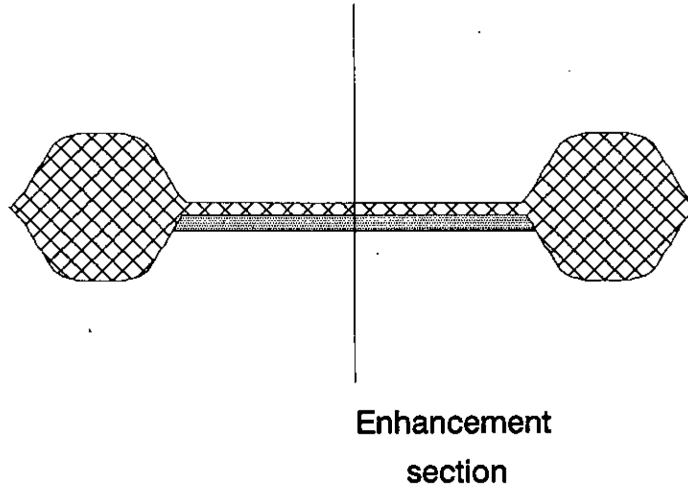


Figure 3.9. MOS gate region in Suprem III prior to ETCH statement.

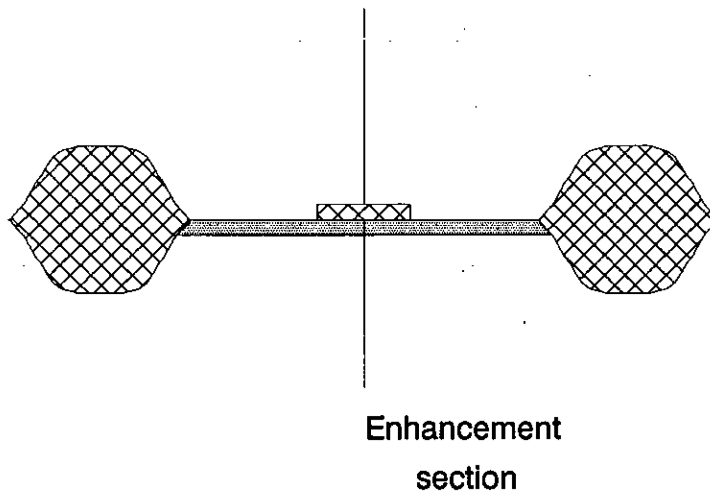


Figure 3.10. MOS gate region in Suprem III after ETCH statement.

If effective use is to be made of the ETCH and DEPOSITION steps within SUPREM III, it is vital to make sure that they are well characterised in advance, otherwise significant errors will arise. This becomes of particular importance when, for example, the effects of over- or under-etching are being investigated. In such

circumstances it might be more beneficial to perform a two-dimensional analysis using one of the dedicated topographical simulators, such as DEPICT-1/2, to provide the detail required.

3.2.6 Electrical

An important use of process simulation is to provide process and device engineers with the electrical performance of devices fabricated using a specific sequence of process steps. As has been seen in the preceding sections (3.2.1 through to 3.2.5), process simulation calculates the physical structure and its associated doping profiles for a given set of fabrication steps. These can then be analysed to provide basic electrical information such as

- electron / hole concentrations
- layer conductivity / resistivity
- net charges within a layer
- threshold voltage for MOS device

For a more accurate and detailed analysis, such as would be required for device optimisation, dedicated device simulators exist which provide 1-D [30] and 2-D [31] data. These programs may be interfaced directly to process simulators such as SUPREM III or SUPRA allowing the transfer of doping profiles and device structure. As with all the simulation programs discussed so far, it is important to calibrate the electrical simulations to a given fabrication process if quantitative process analysis is to be performed.

The electrical parameters obtained from SUPREM III are calculated from a numerical solution to Poisson's equation as shown in equation 3.7 under a series of bias conditions.

$$\frac{\partial}{\partial y}\left\{\epsilon(y)\frac{\partial\psi(y)}{\partial y}\right\} = -q\{p - n + N_D^+ - N_A^-\} \qquad \text{semiconductor} \tag{3.7}$$

$$= 0 \qquad \text{insulator}$$

where :-

ϵ	=	dielectric constant of material
ψ	=	potential
p	=	hole concentration
n	=	electron concentration
N_D^+	=	electrically active donor ion concentration
N_A^-	=	electrically active acceptor ion concentration

In SUPREM III, it is the ELECTRICAL statement which indicates the start of a solution to Poisson's equation. If V_T is to be calculated then a series of bias points will need to be specified since the threshold voltage is calculated by extrapolating the conductance versus bias curve tangentially to zero from the point of maximum slope. By default, layers are biased to zero volts, but can be individually biassed by use of the BIAS statement. In addition, the presence of any surface charge on the substrate can be specified by the QSS statement. An END.ELECTRICAL statement signifies the end of the solution to Poisson's equation. Figure 3.11 shows the electrical statements required to calculate V_T for an MOS transistor structure.

```
.
.
ELECTRICAL  STEPS=18   VTH.ELEC   LAYER=1
BIAS                  LAYER=3    V=-0.2   DV=0.1   ABSCISSA
QSS                   LAYER=1    CONC=5E10
END.ELECTRICAL
.
.
```

Figure 3.11 SUPREM III input statements to perform a threshold voltage calculation.

As can be seen, these commands bear little relation to any real on- or off-line threshold voltage measurement technique, but they can be used to assist in the initial calibration of the simulator to a particular process.

UNIV. EDIN. LIB.

3.3 Two Dimensional Simulation

As device geometries scale down below 1.5 - 2.0μm, two-dimensional modelling of impurity distributions and surface topology become critical in assessing device performance.

The basic operation of MOS device involves the field effects imposed by electrodes on doped silicon regions. Analysis of these devices requires a minimum of 2-D and preferably 3-D [32] solutions to Poisson's equation (see §3.2.6) if accurate modelling of their performance and sensitivity to fabrication technology is to be assessed. Most variations in fabrication processing result in subtle 2-D effects which enforce the need for 2-D process modelling to provide sufficient resolution for these imperfections to be reflected in the structural and doping information.

A number of 2-D simulators such as SUPRA [33], SUPREM-4 [34], COMPOSITE [35], BICEPS [36], RECIPE [37] and ROMANS-II [38] have therefore been developed with the intention of providing general purpose small geometry process simulation capabilities. All of these have their respective limitations, some of which will be highlighted in the course of this section, making it important to ensure that the correct simulator is used for a given modelling problem.

The major difficulty associated with the development of 2-D process models is in the consideration of the moving boundary conditions for diffusion during thermal oxidations. There are several methods available to surmount this difficulty, though none is without compromise. By using an analytic solution to the impurity diffusion during oxidation, the issue of spatial discretisation can be avoided and calculations can be achieved quickly. This rapid response is obtained at the expense of flexibility, in that analytic models are generally only valid for low concentration diffusions. Alternatively, finite element and finite difference techniques, with or without moving boundaries may be employed.

The problem associated with the numerical solutions is that of producing a grid system around which the semiconductor structure can be defined and the required equations solved. One way to solve this problem is to discretise the physical simulation region at each time step and then solve the resulting nonlinear equa-

tions. In order to cope with the fact that one of the boundaries is moving, a new spatial discretisation and an associated interpolation from the old discretisation to the new one is needed at each time step. Another approach is to transform the diffusion equations and their respective boundary conditions from a physical co-ordinate system to a co-ordinate system where the moving boundary appears stationary in the time domain. In this way, the solution region is simplified at the expense of making the underlying diffusion equations more complicated.

SUPRA established itself as one of the first commercially available two-dimensional process simulators. Many of the models are based on those of SUPREM III and allow the simulation of ion implantation, silicon oxidation, inert drive-ins and low temperature deposition and etching. Although the structure may comprise up to ten layers from six material types, impurity distribution is only calculated in silicon and silicon dioxide layers. Thermal oxidation must be solved analytically, and diffusions in doped inert ambients must be solved numerically, but the remaining models may use either method. However, one severe constraint is that numerical solutions may be performed after analytic, but *not* visa versa, which seriously limits the scope for simulating entire processes.

SUPREM-IV [39] is a still more recent two-dimensional simulator which includes an advanced set of numerical and analytic models. A numerical thermal oxidation model is now included. This eliminates many of the constraints applied to SUPRA, though it does dramatically alter the overall computational intensity. Impurities are now simulated in all layers, though the total is now limited to four from silicon, silicon dioxide, nitride and polysilicon.

In contrast to these two packages is COMPOSITE. This possess a completely different development background from the Stanford/TMA family of simulators, coming from the Fraunhofer-Institut für Festkörpertechnologie in Munich. The models included are two-dimensional extensions of those found in SUPREM III, however physical modelling of deposition, etching and lithography steps are also incorporated, giving a genuine ability to model an entire MOS fabrication process.

Neither SUPRA or SUPREM-IV have any physically based topographical capabilities, though SUPRA does have a bidirectional interface with DEPICT / SAMPLE to provide these facilities.

All of the simulators mentioned during the course of this section produce two-dimensional structural and impurity concentration data. However unlike their one-dimensional counterparts, a dedicated device simulator such as PISCES-2B or CANDE [40] is required to provide the resulting electrical characteristics.

By comparison with one-dimensional analysis, two-dimensional process simulation is far more complex to use efficiently and successfully. A knowledge of the inherent limitations of each simulator is required to prevent erroneous results. More crucially, a careful choice of grid structures is essential for the numerical modes so that sufficient detail can be achieved in the active parts of the device. It should also be noted that, compared to a one-dimensional simulator, less accurate vertical doping profiles will be produced given the reduced number of grid points in the vertical dimension and the use of less mature and comprehensive models. For example, SUPRA may use up to 100 nodes in a given dimension, whereas SUPREM III could use up to 500 in the vertical dimension. Nevertheless, the requirement for two-dimensional simulation can not be regarded as anything other than essential, if useful qualitative and quantitative analysis of small geometry devices is to be achieved.

3.4 Computational Requirements of Process Simulation

The application and importance of both one and two-dimensional process simulation in designing, developing and debugging VLSI fabrication processes has been highlighted in the preceding sections of this chapter. However, before any conclusions can be drawn on the overall benefit of process simulation, it is necessary that the computing resources demanded by the relevant software be presented and assessed.

In the past, these software packages wrere typically run on multi-user mainframe machines, such as the VAX™ 11/750 or on a minicomputer, such as one of the MicroVAX™ series of machines. In their day, these machines were very popular,

but were not cheap to buy or maintain. Even with efficient virtual memory management systems, the number of users that could concurrently access these large programs concurrently, without the time spent swapping tasks dominating, is quite low (typically 2-6 depending of the software). At present, personal workstations, such as the SUN 3™, 4™, or IPX™ range of machines, are capable of providing computing powers in the order of 10-30 times that of their predecessors, for a fraction of the costs. However, even these technological advances are not sufficient to meet the demands being placed on simulation machines due the increase in the utilisation of process simulation and by the move towards 2-D and 3-D simulation that is required to model present and future processes.

This information highlights the problems associated with this hardware arrangement. It does not lead to an efficient process simulation environment in which each process / device engineer who *could* benefit from access to the software would *actually* be able to use it. The problem is further compounded by the fact that as device geometries scale down, not only is there a greater need for more simulation but additionally, each simulation becomes more computationally intensive. In the extreme, the use of three-dimensional modelling, which has been proposed for submicron processes, would render conventional simulation procedures unusable. Therefore, a fresh approach is required to provide a suitable computing environment which allows relatively cost-effective access to the necessary software by all the relevant engineers.

There are two aspects which could lead to the significant improvements which are needed. Firstly, an *economic* source of superior computing power, which is preferably available in a desktop format, would deliver a more immediate solution, though emphasis has to be placed on the 'economic' constraint if global access to the engineering community is to be met. This proviso rules out the alternative range of computing platforms in the CRAY™ style of supercomputing platforms which are extremely expensive to both purchase and maintain. Secondly, on a more long term basis, the development of improved numerical algorithm's within the

solution of the physical models would produce a lower loading on both existing and future computational platforms. This route must not be ignored, but it is unlikely to produce significant advances in the short term.

Although SUPREM II is only a one-dimensional process simulator which presents a relatively low load on a processor, the availability of source code has allowed execution on a variety of computing platforms to be compared. Table 3.2 presents a series of CPU times for a typical CMOS gate region simulation on a number of processing elements, along with an estimate of their respective costs. These results do provide a clear indicator as to where a possible solution may exist.

Hardware Platform	CPU Times (secs)	Approx. Costs (£) (1990)	Remarks
VAX 11/750	4460	55000	plus maintenance(~10%)
SUN 3	2070	6000	plus maintenance(~10%)
PC-AT 80286	6170	2500	'AT' Compatible
PC-AT + IMST414	5580	2700	INMOS Transputer - with no floating point unit
PC-AT + IMST800	312	3100	INMOS Transputer - with floating point unit

Table 3.2. Comparison

3.5 Summary

This chapter has presented a brief synopsis of process simulation within the semiconductor manufacturing environment. The main physical and electrical models as contained in a particular one-dimensional simulator, SUPREM III, have been described along with examples of their respective input data formats

This chapter has presented a brief review of process simulation and its role within the semiconductor fabrication environment. The merits and limitations of both one- and two-dimensional simulation are then considered in the light of cur-

rent processes and their future trends. SUPREM III is then examined in detail as an example of a typical one-dimensional simulator with a discussion of its main physical and electrical models and their respective input and output data formats. Although two-dimensional simulation is vital in order to model the lateral effects that become significant in small geometry device processing, it is shown to lack in the accuracy obtainable from one-dimensional modelling and require considerably greater computational resources. The issue of computational overheads is then expanded upon and shown to severely limit access to this software. With this in mind, a comparison of various hardware platforms, in terms of both cost and performance, is used to focus on a possible simulation environment which would enable its full potential to be exploited.

References

1. S.M. Hu, S. Schmidt, "Interactions in Sequential Diffusion Processes in Semiconductors", J. Appl. Phys., Vol 39, pp4272-4283, 1968

2. P.H. Singer, "Process Modeling and Simulation", Semiconductor International, Vol 10, No. 2, p78, February 1987.

3. W. Fichtner, L. Nagel, R. Penumalli, W. Petersen, J. D'Arcy, "The Impact of Supercomputers on IC Technology Development and Design", Proc. IEEE., Vol 72, No. 1, p96, 1984.

4. A.M. Mazzone, G. Rocca, "Three-Dimensional Monte-Carlo Simulations - Part 1: Implanted Profiles for Dopants in Submicron Devices", IEEE. Trans. CAD, Vol 3, No. 1, pp64-71, 1984.

5. S.R. Nassif, A.J. Strojwas, S.W. Director, "FABRICS II: A Statistically Based IC Fabrication Process Simulator", IEEE Trans. Computer-Aided Design, Vol.CAD-3, No.1, January 1984, pp40-45.

6. P.K. Mozumder, C.R. Shyamsundar, A.J. Strojwas, "Statistical Control of VLSI Fabrication Processes: A Framework", IEEE Trans. Semiconductor Manufacturing, Vol.1, No.2, May 1987, pp62-71.

7. "Introduction to COMETS", COMETS User Manuals, Version 5, Consilium Inc., Mountain View, CA, USA, December 1988.

8. A.J. MacDonald, "", Ph.D. Thesis, EMF, Edinburgh University, 1989.

9. P. H. Singer, "Process Modeling and Simulation", Semiconductor International, No.2, Vol. 10, February 1987, p79.

10. R.J. Sokel, D.B. MacMillen, "Practical Integration of Process, Device and Circuit Simulation", IEEE Trans. Elect. Dev., Vol.32, No.10, October 1985, pp2111.

11. D.A. Antoniadis, R.W. Dutton, "Models for Computer Simulation of Complete IC Fabrication Process", IEEE Trans. Elect. Dev., Vol.26, No.4, April 1979, pp490-500.

12. C.P. Ho, J.D. Plummer, S.E. Hanson, R.W. Dutton, "VLSI Process Modeling - SUPREM III", IEEE Trans. Elect. Dev., Vol.30, No.11, November 1983, pp1438-1453.

13. B.E. Deal, A.S. Grove, "General Relationship for the Thermal Oxidation of Silicon", J. Appl. Phys., Vol.36, 1965, pp3770-3778.

14. L. Mei, R.W. Dutton, "A Process Simulation Model for Multi-layer Structures Involving Polycrystaline Silicon", IEEE Trans. Elect. Dev. Vol.29, No.11, November 1982, p1732.

15. C.P. Ho, J.D. Plummer, S.E. Hanson, R.W. Dutton, "VLSI Process Modeling - SUPREM III", IEEE Trans. Elect. Dev., Vol.30, No.11, November 1983, p1439.

16. J.C.C. Tsai, "Diffusion", VLSI Technology 1/e, ed. S.M. Sze, McGraw-Hill, 1983, p171.

17. C.P. Ho, J.D. Plummer, S.E. Hanson, R.W. Dutton, "VLSI Process Modeling - SUPREM III", IEEE Trans. Elect. Dev., Vol.30, No.11, November 1983, p1442.

18. P. Pichler, W. Jüngling, S. Selberherr, E. Guerrero, H.W. Pötzl, "Simulation of Critical IC-Fabrication Steps", IEEE Trans. Computer-Aided Design, Vol.CAD-4, No.4, October 1985, p384.

19. J. Lindhard, M. Scharff, H.E. Schiott, "Range concepts and Heavy Ion Ranges", Mat. -Fys. Medd. Vid. Selsk. Vol.33, No.14, 1963, pp1-42.

20. J. Gibbons, S. Mylroie, "", Appl. Phys. Lett., Vol.22, June 1973, p568.

21. A.M. Mazzone, G. Rocca, "Three-Dimensional Monte-Carlo Simulations - Part 1: Implanted Profiles for Dopants in Submicron Device", IEEE Trans. Computer-Aided Design, Vol.CAD-3, No.1, January 1984, p64.

22. L.A. Christel, J.F. Gibbons, S. Mylroie, "Application of the Boltzmann Transport Equation to Ion Range and Damage Distributions in Multi-layer Targets", J. Appl. Phys. Vol.51, December 1980, pp6172-6182.

23. A.M. Mazzone, G. Rocca, "Three-Dimensional Monte-Carlo Simulations - Part 1: Implanted Profiles for Dopants in Submicron Device", IEEE Trans. Computer-Aided Design, Vol.CAD-3, No.1, January 1984, p65.

24. "SUPREM II Users Manual", Technology Modeling Associates Inc., Palo Alto, CA, USA, December 1988, p 3-6.

25. C.P. Ho, J.D. Plummer, S.E. Hanson, R.W. Dutton, "VLSI Process Modeling - SUPREM III", IEEE Trans. Elect. Dev., Vol.30, No.11, November 1983, pp1447.

26. L. Mei, R.W. Dutton, "A Process Simulation Model for Multi-layer Structures Involving Polycrystaline Silicon", IEEE Trans. Elect. Dev., Vol.29, No.11, November 1982, pp1726-1734.

27. "DEPICT-1 Users Manual", Technology Modeling Associates Inc., Palo Alto, CA, USA, June 1987.

28. "DEPICT-2 Users Manual", Technology Modeling Associates Inc., Palo Alto, CA, USA, January 1988.

29. W. Oldham, S. Nandagokar, A. Neureuther, M.O. Toole, "A General Simulator for VLSI Technology and Etching Processes", IEEE, Trans. Elect. Dev., Vol.28, 1981, pp1455-1459.

30. Y.P. Yu, R.W. Dutton, "SEDAN-3, Users Manual", Integrated Circuits Laboratories, Stanford University, Stanford, CA, USA, 1985.

31. "PISCES-2B Users Manual", Technology Modeling Associates Inc., Palo Alto, CA, USA, March 1988.

32. W. Fichtner, L.W. Nagel, B.R. Penumalli, W.P. Petersen, J.L. D'Arcy, "The Impact of Supercomputers on IC Technology Development and Design", Proc. IEEE, Vol.72, No.1, January 1984, p106.

33. D. Chin, M.R. Kump, H. Lee, R.W. Dutton, "Process Design using Two-Dimensional Process and Device Simulators", IEEE Trans. Elect. Dev., Vol.29, No.2, February 1982, pp336-340.

34. "SUPREM-4 Users Manual", Technology Modeling Associates Inc., Palo Alto, CA, USA, 1989.

35. J. Lorenz, J. Pelka, H. Ryssel, A. Sachs, A. Seidl, M. Svoboda, "COMPOSITE - A Complete Modeling Program of Silicon Technology", IEEE Trans. Elect. Dev., Vol.32, No.10, October 1985, pp1977-1986.

36. B.R. Penumaali, "A Comprehensive Two-Dimensional VLSI Process Simulation Program, BICEPS", IEEE Trans. Elect. Dev., Vol.30, No.9, September 1983, pp986-992.

37. G.E. Smith, A.J. Steckl, "RECIPE - A Two-Dimensional VLSI Process Modeling Program", IEEE Trans. Elect. Dev., Vol.32, No.2, February 1982, pp216-221.

38. C.D. Maldonado, "ROMANS II, A Two-Dimensional Process Simulator for Modeling and Simulation in the Design of VLSI Devices", Appl. Phys., Vol.A31, No.3, 1983, p119-138.

39. "SUPREM-IV Users Manual", Technology Modeling Associates Inc., Palo Alto, CA, USA, July 1989.

40. "CANDE Users Manual". Technology Modeling Associates Inc., Palo Alto, CA, USA, July 1988.

All great discoveries are made by mistake.

Chapter 4

Parallel Computational Systems

4.1 Introduction

The von Neumann model for computer architectures as shown in figure 4.1 or a variation there-on, has remained the dominant serial micro-architecture for many decades. Although computational theory proposed that this model could provide the means to solve most existing numerical problems, it has been the technology required to realise this in hardware that has proved to be the fundamental limiting factor over much of its life.

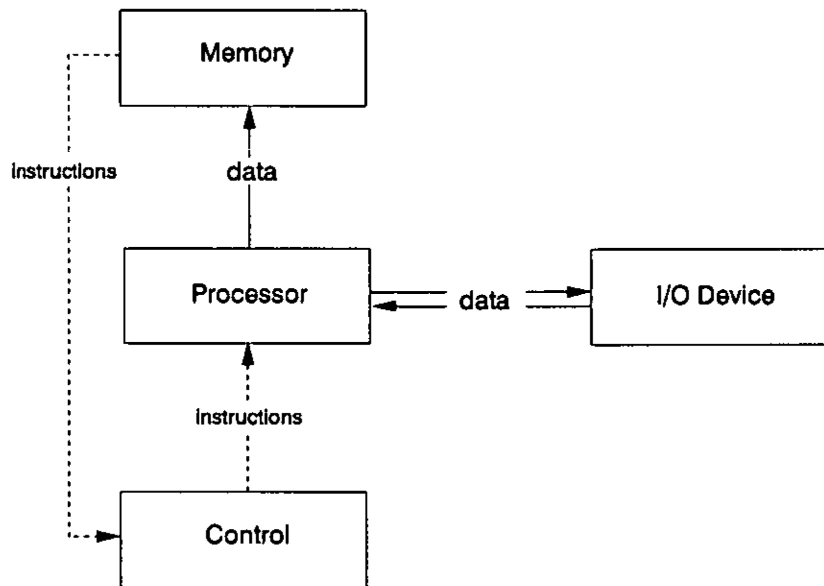


Figure 4.1. The von Neumann computer architecture.

However, continued progress in fabrication, device and circuit design techniques has led to greatly enhanced CPU speeds, instruction set, memory sizes and access times. In this way, VLSI technology has lifted many of the hardware limitations which were originally so restrictive on computing achievements and has enabled greater potential to be extracted from this architecture. Consequently,

families of devices have emerged which offer a wide range of performance and provide a simple path for upgrading machine capability. A prime example of this is the Intel 80x86 microprocessor family (8086 - 80486) which powers the IBM PC series of computers.

Because of the basic architectural consistency throughout this continual development, end users have been able to place relatively little importance in architectural detail and a high degree of software compatibility and portability between hardware platforms has resulted. This has been accompanied by the creation of a vast pool of knowledge and expertise in both hardware and software. A huge investment has been made over this period into the development of many immense software packages for use by academic and commercial establishments alike - a factor which has continually influenced and hindered the course of alternative developments.

Unfortunately, although advancing technologies have altered or removed many of the constraints which influenced the von Neumann design since its inception, it is becoming increasingly difficult to obtain yet more performance from this time-honoured model. It is the architecture itself which is now the limiting factor due to the various communication and processing bottlenecks that are associated with it and the time has come for new approaches to be researched and applied to the computational problems being faced today and in the future.

The knowledge that parallelism could provide a means to improve computer performance has been around since before the introduction of the first electronic computers. However, it is only more recently that the technology with which to implement such concepts has become available. This progressive technology has led to a repeal of Grosch's law [1], which stated that the best cost-performance ratio would always be obtained from a powerful uniprocessor. No longer is it true that a single large processor will necessarily outperform a number of smaller processors, given the same capital cost. VLSI fabrication technology already enables the entire functionality of complex microprocessors to be realised on a single chip. Similarly, multiple processing elements, albeit of a less elaborate nature, have also been

produced in single chip formats. If the cost-effectiveness associated with volume IC fabrication is to be harnessed, then parallelism must be exploited on an extensive scale. Consequently, the construction of parallel computers from a large number of relatively small, cooperating processing elements, provides an attractive alternative to the high budgets associated with the serial supercomputers of today. Such levels of system integration are already being achieved and have resulted in the introduction of affordable multiple processor computing platforms. Although much work remains to be done with respect to the processing technology, a significant proportion of the current technological research is altering its emphasis towards that of processor interconnection. Experience has provided the evidence that this factor is highly influential in overall system performance and suitability to particular applications.

However, although parallelism appears to be a natural concept when observing the way humans tackle problems, traditional computational skills and strategies have moulded the conceptual approach to this field into a highly sequential manner. With this in mind, a parallel programmer has a vast array of new concepts to grasp and additional responsibilities when developing such software. These commitments relate to such issues as the distribution of tasks amongst the available processors and the communications between tasks and processors. Ideally, it should be the function of software tools to assist in determining *how* a program will run on the hardware and permit the designer to describe *what* the program will accomplish. To date, such tools have been virtually nonexistent due to insufficient underlying theory and only now are the first generation of suitably comprehensive development environments becoming available.

Parallel programming is non-trivial and requires the development of many new algorithms and protocols to realise its potential benefits. The porting of existing serial software onto parallel platforms is typically time consuming and highly application dependent, if maximum performance is to be extracted. At present, few compilation facilities exist which can successfully automate this process whilst retaining a sufficient performance advantage over the serial version. Commercial

software development in this field has lagged behind that of the hardware, especially in the provision of complete parallel programming environments which support both program development and its application thereafter. This has left the academic community floundering in its attempt to gain the much needed support required to prove the genuine benefits afforded by this technology and presents an all too familiar catch-22 situation.

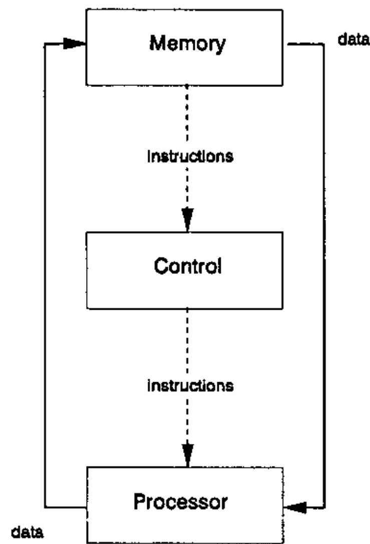
The field of parallel processing has seen many players come and go throughout its relatively short existence as each has attempted to produce competitive and marketable systems. This sadly reflects the complexity and cost of the resources associated with the developing efficient and affordable parallel computing facilities. It also highlights the failure of the industry as a whole to demonstrate and convince the open market of the real benefits that can be provided by this technology.

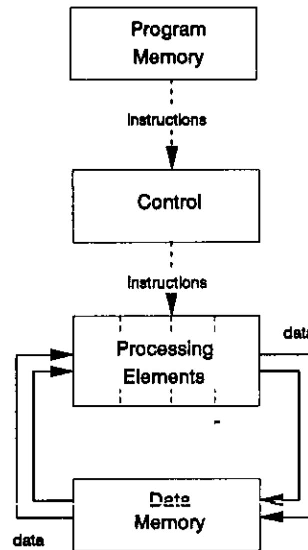
This chapter intends to provide a brief overview of parallel computing architectures, charting their progress from von Neumann's serial model through to the present multiple processor architectures. Then, as an example of one of these and as an introduction to the computing technology associated with this work, a more detailed description of the Inmos transputer will be presented. This section will cover both hardware and software aspects of the transputer technology and then proceed to highlight a few of the factors which have hindered its widespread application to parallel problems.

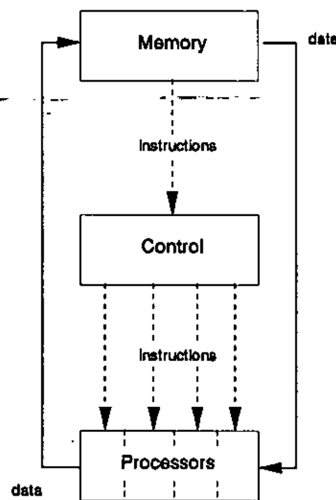
4.2 Parallel Computer Architectures

Many attempts have been made to provide a thorough classification of parallel architectures, though the two most commonly referred to are those by Flynn [2] and Shore [3]. The former based his evaluation on how a particular machine related its instructions to the data that they process, whilst the latter considered the structural organisation of the architecture itself. Terminology associated with both of these classifications has become very much part of the general computer science language used throughout the field. Flynn's taxonomy identifies four distinct classes of processing machines which are listed below and illustrated schematically in figure 4.2.

- SISD   Single Instruction - Single Data
- SIMD   Single Instruction - Multiple Data
- MISD   Multiple Instruction - Single Data
- MIMD   Multiple Instruction - Multiple Data







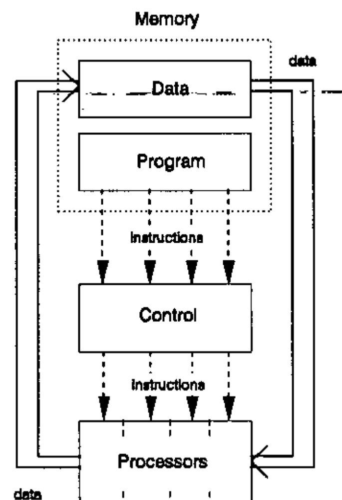


Figure 4.2. SISD, SIMD, MISD and MIMD Architectures.

However, before going on to examine each of the aforementioned architectural classes, it is important to decide how individual systems can be objectively assessed in relation to one and other. Such a review will obviously include a measure of sheer computational performance, but must also consider more wide-ranging issues relating to its implementation and operational requirements. Processor and multiple processor performance is generally quoted in terms of either MIPS (millions of instructions per second) or MFLOPS (millions of floating point operations per second) when considering scientific calculations. The latter is usually highlighted by manufacturers if a dedicated floating point unit (FPU) is included, be it as part of a chipset, as with the Intel 80x86 and 80x87 chip sets, or included on-chip, as in the case of the T800 transputer series of devices.

Comparison of these performance figures can be highly misleading in that the relationship between the instruction execution rate and the resulting arithmetic performance is dependent on both the function and the processor. This then becomes additionally complicated when both CISC (complex instruction set computers) and RISC (reduced instruction set computers) machine architectures are being compared since a RISC processor can be expected to execute more instructions for a given function than its CISC counterpart. The rationale behind the RISC concept is one of providing high performance support for programs written in high level languages [4] and reducing system interrupt latency. One further problem with these figures is that they relate to a *peak* performance, which can only be realised under highly specific or even theoretical conditions. Bearing this in mind, it is therefore far more constructive to combine this peak performance data with results from a number of representative benchmark calculations. There are a number of these *standard* benchmarks, each tailored to highlight a particular computational process. In this way, a genuine and attainable measure of the processor performance can be obtained and it becomes possible to match processing systems to their prospective applications.

4.2.1 SISD

This is the conventional serial von Neumann computer architecture in which there is a single stream of instructions which are performed on a single data set. As was mentioned earlier, a nominal degree of parallelism has been introduced into this model and thus it is worth briefly highlighting the evolutionary course of the serial computer concept and architecture. The earliest designs worked on a bit-serial approach using a single processor to perform all the computations on the required operands, bit at a time. This later gave way to word-serial processing units, which operated on multiple bit words, though still maintained the serial time-sequencing of the instructions. The final development saw the introduction of pipelined processor architectures where individual arithmetic processes were implemented in separate specialised hardware units. Thus, as each data set passed from one process to the next, the hardware process would receive a new data set in its place and hence a number of data sets rippled through the processing units at one time. In this manner, multiple data sets could be simultaneously operated upon by different processes.

This class includes machines ranging in performance from the ubiquitous desktop P.C. through the familiar SUN workstations to the IBM and DEC mini-computers and finally to the very fast serial supercomputers. Examples of these supercomputers are the CRAY-x series of machines from Cray Research and the ETA GF-10 from the now defunct Engineering Technology Associates. Both of these processor families offer performance in the order of GFLOPs by immersing the processing units in liquid nitrogen.

A great advantage with these concepts is that no modifications to existing user software were called upon, making them very attractive propositions. However, if explicit knowledge of task independence prevails, then this may be used to induce further performance by *vectorising* regions of the code for execution on a vector supercomputer. Unfortunately, the running costs of these *supercomputer* pro-

cessors is extremely high due to their power consumption and liquid nitrogen cooling requirements. Consequently they are generally found only within large commercial organisation whose budgets can justify and support these running costs.

4.2.2 SIMD

This architecture follows the simplest approach to constructing a massively parallel computer by connecting a large number of identical processing elements (PEs) together in a multidimensional array, with a single host processor controlling the network. Instructions are then broadcast simultaneously to all of the PEs which then execute these on their locally stored data sets. Typically, the network geometry is both fixed and regular in its nature and this can result in long inter-processor communication paths. For example, in a 64 by 64 two-dimensional array, there can be up to 61 intermediate PEs between the source and destination. The commonest geometries are either a taurus, as shown in figure 4.3, or a hypercube, as shown in figure 4.4. Almasi et al[5] provide a fuller discussion of other geometries which are application to both SIMD and MIMD machines.

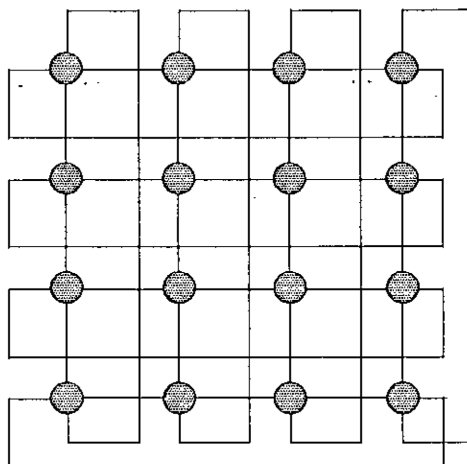


Figure 4.3. SIMD Taurus Geometry.

One very important aspect to consider in large multiprocessor computers is that of *redundancy*. In the basic geometries illustrated so far, if *any* processing elements fail then the entire computer becomes inoperable, which is not at all

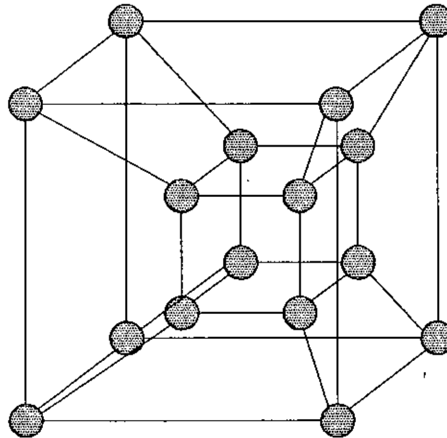


Figure 4.4. SIMD 4-D Hypercube Geometry.

desirable. Most commercial machines therefore tend to include an extra row(s) or column(s) of PEs which can be switched in by software to replace a faulty device and hence provide a more robust operational environment.

The most familiar example of such a machine is the ICL DAP (Distributed Array Processor). The first generation of these machines comprised a 64 x 64 array of PEs constructed from MSI components, each with 4 Kbits of RAM, and performed bit-serial arithmetic at 4MHz. Both the host and PEs are programmed in FORTRAN, albeit with a special version used on the PEs, and the resulting performance is in the order of 200Mflops.

Advanced Memory Technology (AMT) now produce an LSI/VLSI version of the DAP which incorporates a 32 x 32 array of 12MHz PEs hosted by a MicroVAX or Sun workstation. Unlike its predecessor, the AMT DAP incorporates a duplicate set of PEs for redundancy. These sets operate simultaneously until a fault is detected, when the functioning set continues and the faulty set is taken off line.

Other examples of well known SIMD machines include University College of London's CLIP (Cellular Logic Image Processor), Thinking Machines' Connection Machine[6] and Goodyear's Massively Parallel Processor (MPP)[7]. Most of these machines are built as one-off's and are used for specialised tasks within academic

or commercial research and development environments. Most of the previously mentioned machines are programmed in sub- or super-sets of one of the high level serial languages (i.e. C, FORTRAN or PASCAL).

4.2.3 MISD

According to Flynn's taxonomy, machines of this type permit multiple instructions to be executed on every data set. It might therefore appear that this class is somewhat academic since there are in fact no genuine examples of it. However, it may be considered in terms of a *macro* pipelined architecture where each processor executes a different set of instructions on one data set which would presumably be stored in global memory.

4.2.4 MIMD

MIMD computers are an evolutionary step forward from the first generation SISD machines and contain several *independent* processors. These processors are usually identical and each executes its own program, which may or may not be the same for all the processors. In order to contrast this with the parallelism exhibited by SIMD machines, the term *concurrency* is often used with respect to MIMD computers.

MIMD architectures may be classified according to many characteristics, but the two most important and distinctive features are those listed below :-

- The number and power of the processors used.
- The processor to memory relationship and connectivity.

The first classification separates machines with a small number of very powerful processors from those which contain a larger number of relatively smaller processors. Of these, the former have generally evolved from existing serial computers and have the advantage that they can continue to support much of their original software with little or no modification - a highly significant feature for many vendors and potential customers alike. The Cray XMP and YMP series of machines are good examples of this technology and consist of 2, 4 or 8 vector processors combined in a single machine.

The second classification concerns the use of distributed (local) or shared (global) memory amongst the processors and leads to the terms, *loosely-coupled* and *tightly-coupled* in describing MIMD architectures. In a tightly-coupled machine, the available memory is shared by all the processors, with the exception of limited local caches in specific instances. This requires the use of very fast, complex switching networks or a high bandwidth bus to form all of the processor to memory connections. Machines which incorporate physically shared memory posses one major disadvantage in particular in that they can not be scaled up indefinitely, since there will be an ever increasing contention amongst the processors in attempting to access the memory [8].

The loosely-coupled architecture provides an alternative approach by distributing the memory among the processors and thereby reducing the required memory bandwidth. The problem is now one of accessing memory locations on remote nodes and requires a suitable connection strategy. The use of a single bus or switch only leads to the communications bottleneck discussed previously. Similarly, since the total number of possible connections rises with the square of the number of processors, it is equally unfeasible to achieve 100% connectivity. Needless to say, the commonest solution involves the connection of each processor to a subset of the remainder within the system. The general *hypercube* architecture, as pioneered by CalTech [9], is one of the most frequently used interconnection topologies in current generation machines. The basis of a d-dimensional hypercube is that each processor has d connections to its neighbouring processors as can be seen in figure 4.5 which illustrates the simplest members of the series.

The Intel iPSC family of machines use a hypercube architecture as their basis and consist of up to 7-dimensional cubes of 80286/7 or 80386/7 processor sets. The original iPSC/1 used a *store-and-forward* routing strategy which left much to be desired and this was replaced by a hybrid circuit switching and packet switching system in their iPSC/2[10]. This offer a ten times improvement in message latency over the iPSC/1.

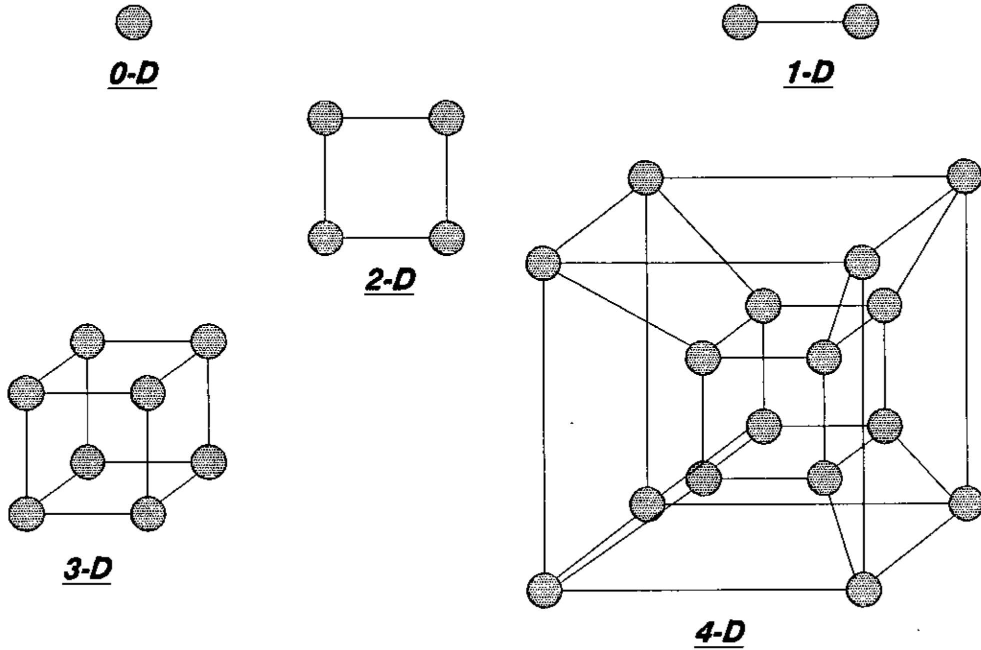


Figure 4.5. The first five members of the hypercube series.

An alternative to this concept of fixed topologies has been taken by Meiko and Parsytec in connecting the processors through switching chips which allows variable topologies to be constructed under software control. When this is combined with an adaptive routing software package, such as TINY[11] which is discussed later in §4.3.3, then application codes can be designed independently of the underlying processor topology. This in turn permits topological variations to be constructed and tested for a particular application.

The range of MIMD machines available at the present time is too large to be successfully highlighted within the scope of this work. Hockney [12] provides a comprehensive survey of these machines and an equally full and more recent survey of SISD, SIMD and MIMD machines is given by Trew and Wilson [13].

4.3 The Inmos Transputer

4.3.1 Architecture and Concepts

The term *transputer* refers to a unique family of VLSI microprocessor devices produced by Inmos Ltd, all of which adhere to a common architecture. The transputer hardware and its associated programming language, OCCAM, were both designed specifically to address the issue of parallel processing and alleviate many of the constraints imposed by conventional systems. The transputer can be considered as the first genuine single chip microcomputer since processor, memory and communication links are all integrated into a single device. As can be seen from figure 4.6 which illustrates the general transputer architecture, each product also contains an application specific interface which enables it to be customised for a particular purpose.

At the present there are three main members to the first generation of transputer products and the main features of these are highlighted in table 4.1. As will be discussed later in this chapter, a second generation transputer has been launched [14] by INMOS but this will not be available until early 1992.

	T212	T414	T800	*T9000*
Processor	16 bit	32 bit	32 bit	*32 bit*
Floating point support	n/a	μcode	FPU	*FPU*
On-chip RAM	2 Kbyte	2 Kbyte	4 Kbyte	*16 Kbyte*
Max Clock Freq.	20 MHz	20 MHz	30 MHz	*50 MHz*
MIPS (pk/sustain)	20/10	20/10	30/15	*200/70*
MFLOPS (pk/sustain)	n/a	n/a	2.9/2.25	*25/15*

Table 4.1. Summary of basic transputer features.

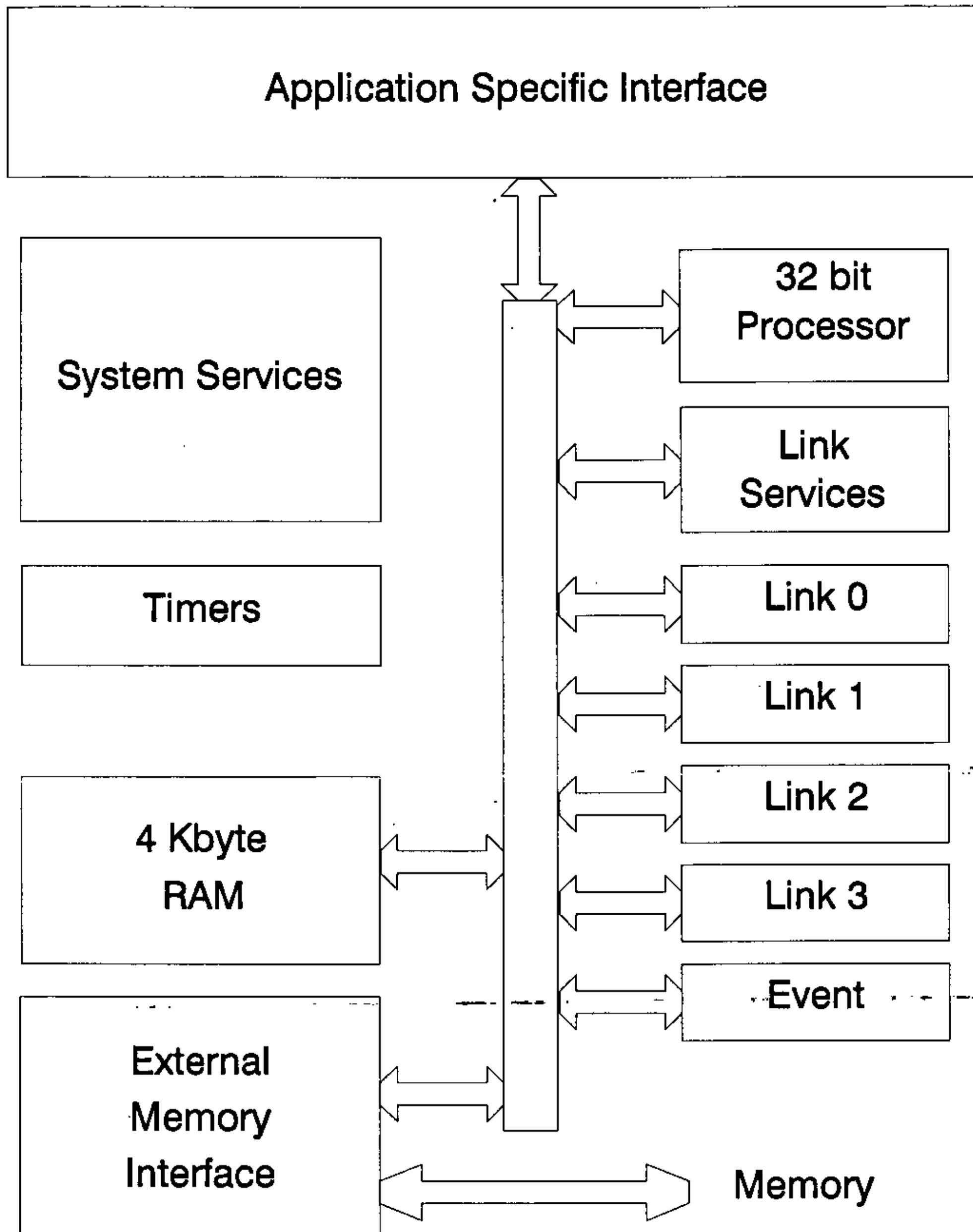


Figure 4.6. The general transputer architecture.

The processor itself follows a RISC-like philosophy by using a small core of very fast, simple instructions to boost performance whilst minimising the CPU complexity. In general, this approach results in a larger code for a given task than would have been required by a conventional CISC architecture. However, in the case of the transputer this has been overcome by the use of a single-byte basic

instruction format. This can be extended to multi-byte instructions where necessary, with each byte retaining the same format [15]. Not only does this improve the effectiveness of the instruction fetching mechanism, but it also means that code can be run on both the 16 and 32 bit transputers without any recompilation.

Unlike conventional RISC machines, the transputer instruction set is not hard-wired, but is instead contained in microcode so that application specific functionality can be simply included with no loss of performance. It is through this approach that the floating point and graphics capabilities of the T800 have been offered. The transputer also benefits from the RISC philosophy in terms of very fast process switching [16] due to the small number of registers involved, namely three stack and three control [17]. The processes being switched may be low level or complete high level user processes and in this manner transputers can execute multiple processes on a single processor, which provides an economic route to system development. In the support of high level languages, procedure calls are dealt with in a similar manner, with each procedure being allocated its own work-space.

The reduced complexity of the CPU which is afforded by the RISC-like philosophy means that only 25% of the silicon area on the T414 is occupied by the CPU, which enables sufficient memory and the I/O links to be included on the same device. The inclusion of on-chip memory permits considerable architectural freedom when designing multiple processor systems. It may also be sufficient to store the entire code required by many embedded applications and thus remove the need for any external memory. The second important feature of transputers is that of their serial links which provide inter-processor communication paths. There are up to four bidirectional links on each transputer which operate asynchronously at speeds up to 20 Mbit/s. In real terms this means that it is possible to transfer up to 1.5 Mbytes of *useful* data per second on each of the two wires which constitute a link [18]. The link drivers are directly implemented in silicon and operate

autonomously. The sending and receiving processes remain unscheduled until both are ready, at which point communication is established and data will be transferred in parallel with CPU (and FPU where applicable) operations.

The point to point communication links enable direct connections to be made to two (T212) or four (T414/T800) other machines or components, facilitating the construction of processor networks with arbitrary topology and size. Figure 4.7 illustrates how these simple links enable arbitrary network configurations to be easily constructed.

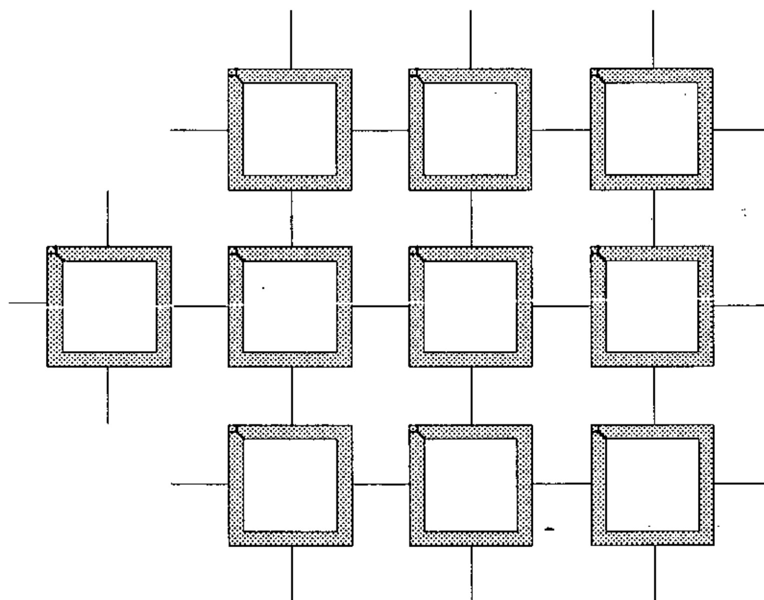


Figure 4.7. Construction of an arbitrary network configuration.

The serial nature of the links simplifies the physical connections to a twisted pair of wires between basic plugs and sockets, rather than the complex backplane technology associated with bus-based systems. The wire links also enable processors to be distributed between a number of cabinets and/or act as a local area network.

In contrast to multiprocessor buses, the asynchronicity of these links eliminates contention for the communication mechanism, regardless of the network size. Additionally, both memory and communications bandwidths scale directly with the number of transputers in the system and neither saturate with network

expansion [19]. This last feature, coupled with the process-oriented nature of the original operating language, OCCAM, permits systems to be reconfigured independently from their application software and thus simplify the task of finding an optimum configuration for a given application. Hence transputer systems offer truly scalable performance as well as a controlled performance degradation in the event of processor failures - a feature of great importance in complex real-time systems. Such flexibility in terms of system development, expansion capability and fault tolerant redundancy makes transputer based parallel systems an attractive option for a broad spectrum of applications. These range from high performance embedded controllers through real-time graphics and simulation to application specific accelerators. It is this final category which broadly describes their role within the work which will be presented in the following chapters.

4.3.2 Parallel Algorithms

There are many advantages to be gained from fragmenting an application into a set of concurrent modules which may be executed on a multiple processor computing system. However, as will be highlighted by the work presented in Chapter 5, parallelising an existing application is generally a non-trivial task. The extent of these difficulties and the resulting parallel performance is very dependent on the application concerned and how it was originally structured. Ideally, programs should run N times faster when executed on N processors compared to their execution a single processor. But in reality the speedup will range anywhere from a poor $\log N$, through the common $N/\log N$ response and on to N itself [20]. Unfortunately it is possible in some instances to realise a speedup of less than 1 as a consequence of poor programming, which is exacerbated on a parallel machine. Finally, in a few cases, speedups in excess of N have also been reported [21], but these are the exception rather than the rule.

Although the choice of algorithm is important in determining the true speedup, it is only one of many factors which may influence the program running time. As was originally noted by Amdahl [22], the compute time for an application may be divided into *parallel* and *serial* portions. The resultant speedup will be

asymptotically limited by the latter, no matter how efficient the former can be made. For example, consider a particular task which requires a total of 100 operations to be performed, of which 80 can be done in parallel but the remaining 20 require serial execution. Then the maximum speedup attainable by using 80 or even 100 processors, compared to a single processor, is 5 (limited by the 20/100 serial operations) and not 80 (the parallel content).

The crux of the matter is not in writing or re-writing the application, but in how it is mapped on to the chosen hardware architecture. Sadly, parallel computers are not particularly forgiving and the difference in performance between programs which have been written well or not, is far greater than would be obtained from a serial architecture. This can be taken to the extent that in an extreme case, a multiprocessor version may run more slowly than the original serial program. A successful and efficient solution requires the ability to *think parallel*. This entails taking the distribution of both data, computation and inter-processor communications into account. The idea being to match the program/algorithm parallelism to that of the target architecture, which if achieved successfully, will positively influence the overall system performance[23][24].

Programs typically exhibit many levels of parallelism and the term *granularity* is often used to indicate the level of independent computation that each processor can perform in relation to its communication needs. A *course-grained* application is one which may be divided into logical parts involving relatively long independent processing sequences with little synchronisation or communication. In contrast to this, few instructions are executed between inter-processor communications in a *fine-grained* application. The concept of granularity may be shown schematically by figure 4.8 which illustrates how a program may be split up into subtasks, using a cubic representation.

However this split is achieved, it is vital that the overall system efficiency be maximised by balancing the loads placed on each processor. This is to ensure that they are performing *useful* work as much of the time as is possible and are not idle

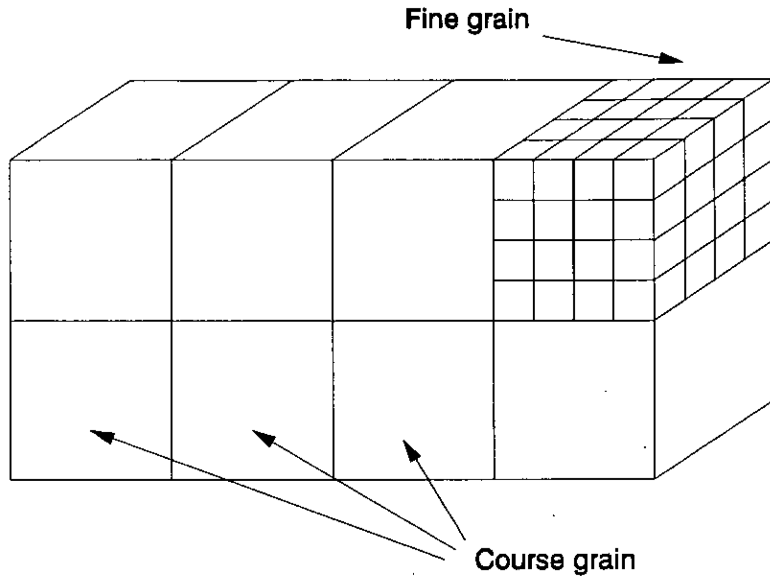


Figure 4.8. Course- and Fine-grained task granularity

whilst waiting for other processors to catch up. Hence as can be seen from equations 4.1 and 4.2, there is a close tie between the efficiency of an entire parallel system and the resulting speedup.

$$\text{Efficiency} = \frac{t_1}{N \cdot t_N} \tag{4.1}$$

$$\text{Speedup} = \frac{t_1}{t_N} \tag{4.2}$$

Load balancing is a complex issue in its own right and discussions have been presented on the matter [25]. But, in simple terms, loads can be more effectively balanced for a large number of small tasks [26], where there are many more tasks than processors. Unfortunately, this conflicts with the drive for a higher computation to communication ratio, which requires a reduced number of larger tasks, in order to improve the system efficiency. Although communication technology has seen many improvements of recent, many fine-grained applications still become *communication bound* and thus fail to attain their potential performance.

The resulting trade-off is yet one more of the challenges which must be faced when high performance parallel applications are being developed. Until such times as *intelligent* compilers are capable of efficiently mapping applications to particular architectures automatically, it will be left for the developer to exploit existing parallelism rather than attempt to introduce more. This parallelism can usually be classified under one of the following headings :-

- Geometric parallelism
- Algorithmic parallelism
- Event parallelism

Geometric parallelism is said to exist when the problem space, be it linear, quadratic or cubic in nature, is divided up with each region being assigned to an individual processor. Computations pertaining to the data within each region may then proceed concurrently until inter-processor communication is needed for the exchange of boundary values. In order to minimise the overhead of these communications, it is favourable for the target architecture to match that of the problem space concerned. Thus a linear problem is suited to a chain of processors and a quadratic problem space is best mapped onto a 2-dimensional processor array and so on. This form of parallelism is frequently associated with image processing problems or simulations of physical phenomena, such as those described in chapter 3.

The second form of parallelism to be exploited is that of algorithmic parallelism. In this case, the algorithm itself is decomposed into a number of functions, each of which is assigned to a different processor. The processors are then connected together, typically in the form of a pipeline, through which all of the data will flow in turn. Thus at any one moment, each processor will be working on a separate data set and the degree of parallelism which can be obtained overall will depend on the number of sub tasks available. This approach is well suited to production or process simulations, but is considered to be difficult to implement. Unless the load balancing of each function is carefully tuned, bottlenecks will arise which will limit the system response as a whole. In addition, the optimum processor configuration for one

problem is unlikely to be appropriate for another, making for an inflexible system. This last problem can be overcome by incorporating dynamically reconfigurable links between processors, so that the topology can be altered under software control.

The final form of parallelism is that of event parallelism and this is frequently the simplest and most efficient approach for exploiting the benefits of parallel processing. Providing that the original problem can be broken down into a large number of *independent* tasks, then these can be *farmed out* to a number of *slave* processors for computation under the control of a single *master* processor. Such a configuration is usually referred to as a *task farm*. In its simplest form, each slave executes the same serial code on its local data set. The familiar ray-tracing problems are an example of this, where each processor is given a small patch of the overall image to compute at any time. However, as will be seen in chapter 7, the incorporation of a more sophisticated operating system permits different programs to be run across the task farm. Whichever level of complexity is deemed suitable for a particular application, the benefit of event parallelism comes from the increase in *throughput* rather than an increase in the *performance* realised by one instance of the program. This is in contrast to the response of the two other cases previously discussed. This distinguishing feature often indicates the suitability of one particular algorithm over another for a given application. For example, a task farm is the obvious solution for a large batch of independent simulations or calculations, but is totally unsuitable for reducing the execution time of a single intensive application.

4.3.3 Parallel Software - Languages and Tools

In one respect, parallel computers are no different from conventional serial machines in that by themselves they do little more than convert electrical energy into heat somewhat inefficiently! What transforms them into highly valuable assets is the software that runs on them and this is one area in particular where the state of serial and parallel products differs substantially. To date, powerful and robust programming environments for the parallel software engineer have been notable, primarily by their absence. This strongly contrasts their serial counterparts for

which there is an abundance of products serving most computing fields. By examining the nature and needs of todays computer users, the reason for this state of affairs becomes apparent. The computing community can be coarsely broken down into three groups of users whose needs may be summarised as follows.

- Basic data processing services, such as word processing.
- Scientific 'number crunching' in research and development.
- General application codes for individual user communities.

Under this classification, users from the first of these groups actually don't even want to use computers *per se*, but wish access to the facilities they provide. Their major concern is in the response time of the application packages, which can usually be satisfied by the use of serial computing platforms, and hence will not be considered further at this point. It should also be noted that this group accounts for approximately 90% of the total user community, which helps place the bargaining power of the remaining two groups into perspective.

The second group provided many of the original users in the parallel processing community and was centred around the stage of academic and commercial research into fields such as aerodynamics or quantum physics. Although scientific programmers appreciate good software tools, the overall goal of maximising performance has in the past frequently overruled such luxuries As a result of this and the relatively small community size, these users have created a very limited demand for the development of comprehensive parallel programming environments.

It is from the final group that the second generation of parallel processing users is emerging and whose needs require particular consideration. The application programming community is sufficiently large and loquacious that its impression and success with parallel processing is probably *the* key issue in opening up a volume market in this product area. It has therefore become the responsibility of parallel software and hardware manufacturers alike to provide a satisfactory route for these programmers to take advantage of the attractive price/performance offered by parallel computers. For the companies concerned the cost of failure is high and the writing is very much on the wall for all to see with the demise of past

competitors, such as ETA Systems, Multiflow and Myrias Research. The parallel processing arena is too large to assess in its entirety in terms of the software available within the scope of this chapter and so the remainder of this section will concentrate on the evolution of software tools for the INMOS transputer since its arrival on the market.

Originally, the primary language for the development of transputer based parallel systems was intended to be OCCAM [27], which was designed along with the hardware itself. INMOS provided their Transputer Development System (TDS) [28] as an integrated environment for the programming of transputer networks under OCCAM. In its initial release, it comprised of a powerful editor, compiler, file manager and runtime facility which was available for both IBM and NEC PC hosts. Although this was a good tool, it sadly lacked any debugging facilities or support for *foreign* languages which limited its capability. The provision of a network debugger was included in subsequent releases of the TDS. The process-oriented nature of the language is directly reflected in the transputer architecture and this enables efficient design and execution of parallel systems written in this manner [29]. However, as much as these benefits were seen as being of great importance to the design of high performance parallel systems, OCCAM has never gained an impressive level of popularity amongst the parallel computing community, especially outside the realm of academia. The reasons for this are two-fold. Firstly there was a general reluctance amongst programmers to learn another language, albeit very akin to a conventional high level language with the addition of constructs with which to express parallelism. Secondly, vast resources of software already existed which would never be re-coded in OCCAM, regardless of the possible gain in performance. Thus, transputers may also be programmed in most conventional high level languages, such as C, Fortran and Pascal. Ada, which can also be considered as another concurrent programming language [30], was supported in order to attract military orders on both sides of the Atlantic. In addition, individual processes may be coded in the optimum language for its given role and these pro-

cesses may then be linked together under an OCCAM harness to produce the overall system. This multi-lingual capability is depicted in figure 4.9 and substantially facilitates the parallel implementation of existing codes.

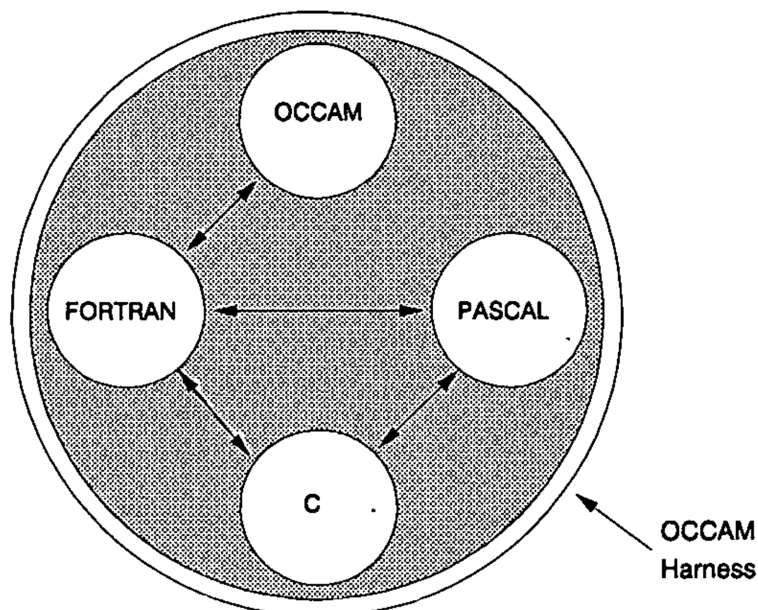


Figure 4.9. The use of multiple languages within a transputer system.

The OCCAM support required for these foreign languages necessitates the use of the INMOS OCCAM Toolset [31], rather than the TDS, which provides a much simplified operating environment with fewer facilities to assist with task of investigating development problems.

The use of these products, whether it be in a single or multiple language system, still requires explicit control of all inter-process communications by the programmer. Because of the fact that there are only four physical links for inter-processor communications, this either limits the interconnection strategy or requires the design of custom multiplexors to increase the number of virtual channels using each link. Both of these options are undesirable as they compromise the ability to alter the partitioning of processes across the network without recoding the communications. As a result of these deficiencies, a number of communications

harnesses have been developed for inclusion within high level codes, examples of which are CSTOOLS[32], FORTNET[33] and TINY[34], which is discussed in detail during chapter 7. These harnesses enable any process to communicate directly with any other process in the system, regardless of where either actually resides and are a valuable addition to the conventional language constructs. Consequently any repartitioning of processes can be achieved within the definition of the system configuration and no recoding becomes necessary. However, even with the inclusion of these improved facilities, the use of these tools is most suited to porting existing applications to a transputer environment or developing new applications of limited complexity. In addition, they are intended primarily for use in single user systems and inherently support the OCCAM model of concurrency[35]. Although this model is generally the most applicable for typical parallel developments, there are alternative models, such as that of *tuple space* as supported by Linda[36], which may be better suited for particular application fields such as transaction processing and database operations.

Because there are a number of hosts which support transputer products (i.e. PC compatible, Sun 3, Apple MacIntosh, MicroVAX), compatibility has limited the facilities which can be provided on all the systems. As a consequence, a number of host specific tools have arisen which can aid developers in their particular environment. These include performance estimation[37], run-time graphical representations[38] or multi-window debugging[39]. An early step in the right direction was the Windows File Server (WFS) from Nexis Technology Ltd[40]. This provided windowed support for the 3L compilers, the INMOS TDS and Toolset by providing a version of the transputer file server running under Microsoft Windows. The development of such tools is highly commendable, but unfortunately does nothing to direct effort into the provision of a unified development environment which is supported by multiple host platforms. However the need for such facilities was confirmed with the introduction of multi-user transputer based systems, where the parallel resources within a single machine could be shared by a number of users at any given time. A static allocation of resources is supported by the Meiko range of computing surfaces, such as installed in the Edinburgh Parallel Computing

Centre[41], whereas a dynamic strategy is taken by the Parsytec range of machines[42]. The former scheme partitions the available resources into a number of fixed *domains*, which appear to the user as a self-contained machine. These may only be altered in size by reconfiguring the physical links of the relevant processors and then rebooting the machine. This is obviously an inefficient and rather limiting method of partitioning the resources. The dynamic approach permits new users to access any unused resources and then releases these again when the session is terminated. Over a period of time, this generally permits a far high degree of processor utilisation by a larger number of users. Both of the above systems are designed around the manufacturers own hardware units and the resulting systems usually employ a large number of processors. An alternative to this is to host an independent set of transputers operating under Helios[43]. This provides a multi-user, multitasking environment with a UNIX-like user interface. It is a genuinely distributed operating system based on the client-server model, communicating with host systems via servers running on a variety of operating systems, including MS-DOS, SunOS, UNIX and AppleOS. Helios supports dynamic processor allocation to multiple users under the control of a special network server and thus a possible system configuration might take the form of that depicted in figure 4.10.

Helios supports a wide range of programming languages (C, FORTRAN, Pascal, Modula II, STRAND), all of which can call each other and system libraries directly[44]. The parallelism of programs written in these languages is specified using the Helios Component Distribution Language (CDL) which defines the required communication topology. The software is then automatically distributed across the physical processor network on a dynamic basis. This means that applications can be run on variable size, variable configuration networks with no code amendments, which is an invaluably powerful feature. Multi language, source level debugger, macro assembler and performance monitoring tools are all available to assist with application development. Graphics support is provided with both X Windows and Windows 3.0[45] interfaces.

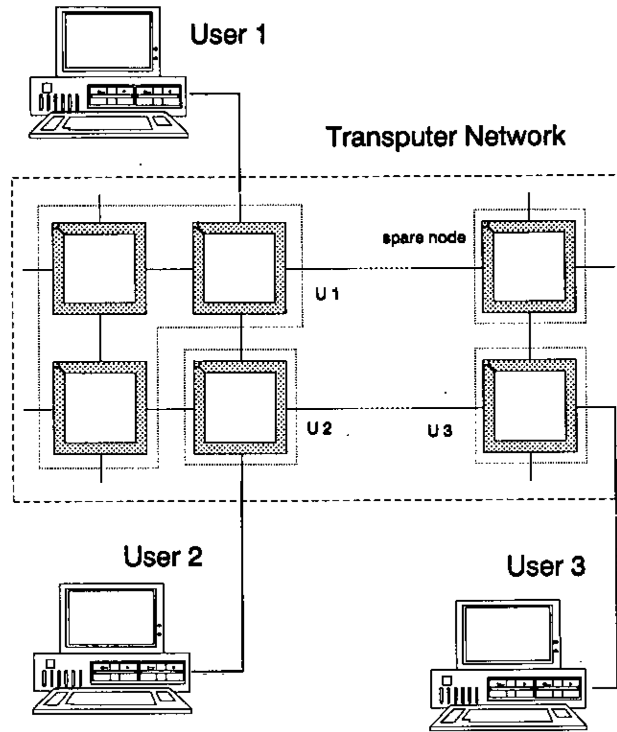


Figure 4.10. Multi-user transputer system running Helios.

An ethernet support package which includes a TCP/IP server enables transputer systems running Helios to be integrated into other computer networks. This facility means that users no longer require a direct connection to the transputer system and has many implications for potential system applications. Hence system configurations may take the general form shown in figure 4.11 with the transputers appearing as another shared system resource.

This remote, networked processing capability has substantial appeal for off-loading computationally intensive tasks from conventional machines, as might be the case in simulation oriented environments. One such application for this approach is discussed in chapter 7 which describes how the combination of experimental design techniques and suitable simulation software leads to improved semiconductor fabrication processes. Another suitable environment is in the

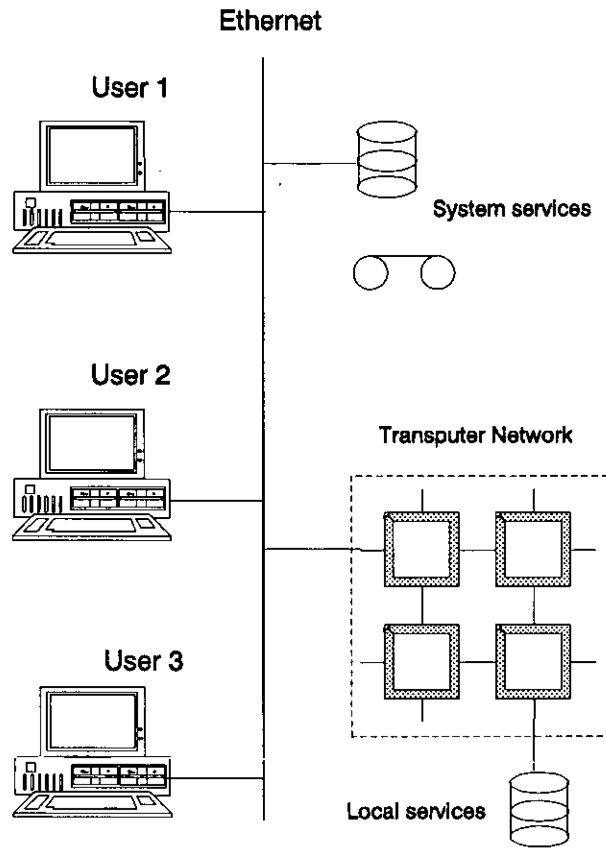


Figure 4.11. Ethernet based multi-user transputer system.

addition of simulation facilities to Computer Aided Manufacturing CAM systems to provide a *what if* capability in the event of mis-processed products being found [46].

The Express parallel programming environment from the ParaSoft Corporation[47] is another tool which has gained support, particularly through its portability to a number of parallel computer platforms. At present, implementations are available for most transputer based systems, NCube's hypercube, Intel's iPSC hypercube, Meiko's Computing Surface, Sun-3/4/386i networks and even some shared-memory machines such as the Cray Y-MP. The basic Express system provides a comprehensive message-passing communications harness which is made available through an extensive set of FORTRAN and C libraries. Program

decomposition and optimisation tools and low level graphics support for PC or X Windows are also provided with the system, all of which are controlled through a menu-driven front-end on the host. A number of additional tools, such as source level debuggers, a simple automatic C and FORTRAN paralleliser and profiling utilities are supported by ParaSoft which produces a comprehensive programming environment.

The IDRIS parallel operating system[48], as developed by Parsys for their SuperNode (SN) series of transputer based machines, provides a multi-user, multitasking operating system. However, unlike Helios, it has specifically been developed for use on the SN series of machines. This is a UNIX compatible operating system that conforms to the IEEE POSIX standards and as such, enables many commercial applications to be ported to a parallel operating environment. Although it employs a distributed client-server model, many system calls can be serviced locally due to the placement of an IDRIS kernel on every processor. This increases performance and reduces the loading on inter-processor communications. Because the kernel is compact and requires no memory management facilities[49], it is ideally suited to a transputer based implementation. User programs may be written using the 3L compilers or the INMOS TDS, with an automated process placement similar to Helios. X Windows and TCP/IP ethernet support are also provided as standard under IDRIS enabling complete integration of the resource into existing facilities. As a consequence of the distributed UNIX environment, Parsys have been able to provide a suitable platform for the porting of Oracle's distributed database system. The significance of this development is far greater than the obvious benefits available to Oracle users, in that it is probably the first port of a genuinely commercial package to a transputer based parallel processing system. The IDRIS operating system offers the same potential upgrade route to many other commercial packages. If even a small number of additional ports are successfully achieved this could finally create the critical mass required to encourage large-scale use of parallel processing in the commercial sector.

There are many alternative software products available for transputer systems, both in terms of operating systems and application development tools, to those discussed in this section. However, it does illustrate the trends and limitations associated with the evolution of this product area. In conclusion, advances in this area lag far behind that of the hardware and this has seriously inhibited the levels of progress and acceptability of parallel computing systems to date.

4.3.4 The Future

INMOS first announced their *transputer* concept in 1984 - the computer on a chip - and the T414 appeared on the market two years later, to be followed by the T212 and T800 members of the family. The micro-architecture of these devices, as shown in §4.3.1, was distinctly unconventional when compared the existing microprocessors and was met scornful criticism in general. But time is often a better judge of these issues and five years on, the transputer concept has matured into a most credible force with sales making it the top 32-bit RISC microprocessor of 1989[50]. Much of this credit is due to its unique architecture which has enabled transputers to take a substantial share in both the system CPU and embedded processor markets.

However, not to be seen to be standing still, the next generation transputer, codenamed the H1, was announced in September 1989[51] and viewed as the first member in a new family of transputer devices. INMOS and their new owners, SGS-Thomson, were convinced that parallel and multiprocessing would be the future for microprocessors and that to meet this their next generation transputer would be designed to support more general purpose parallel computing applications. Formal announcement of the new transputer, the T9000, was given in April 1991 along with the release of its technical specifications. These had every intention of continuing the success of the transputer concept well into the 1990's and fight off the challenge posed by Intel's i860 processor which would lead it in terms of raw performance and time to market. The T9000 is probably the most deterministic microprocessor designed to date[52], especially with respect to its new communications system. When considering embedded applications in particular, deter-

minism is often of much greater importance than sheer computing performance which should enable transputers to maintain their edge over the competition, as it has done in the past.

The architecture of the T9000, as shown in figure 4.12, contains many new features, compared to that of the T800, which provide the additional functionality and performance that the new device promises and yet maintain binary compatibility with existing products.

At the heart of the T9000 is what INMOS describe as a pipelined superscalar processor, whose performance peaks at 200MIPS and 25MFLOPS whilst sustaining in excess of 70MIPS and 15MFLOPS[53]. The five stage pipeline operates on a 50MHz clock and can execute groups of up to eight instructions at one time. In this way the T9000 running at 50MHz can execute T805 binary code approximately 10 times faster than on the original 20MHz T805. The 16Kbytes of on-chip RAM is programmable as RAM, cache or as 8Kbytes of each. For many embedded applications, be they single or multiprocessor systems, the full RAM option may be sufficient so as to require no external memory and greatly simplify the overall design. It is the totally new three layer metal sub-micron cMOS process which permits the large on-chip memory capacity as well as the high clock speed.

It is however, the completely new communications system employed by the T9000 which will be regarded as the most important improvement over its predecessors. Although the existing limit of four physical communications links is retained, these now operate at 100Mbit/s (c.f. 20Mbit/s) and in conjunction with the Virtual Channel Processor (VCP) enable channels to exist between two processes located on *any* transputer. A typical interconnection problem, as depicted in figure 4.13, shows the need for three channels to pass over the single link between the two transputers in order to provide the desired inter-process communications.

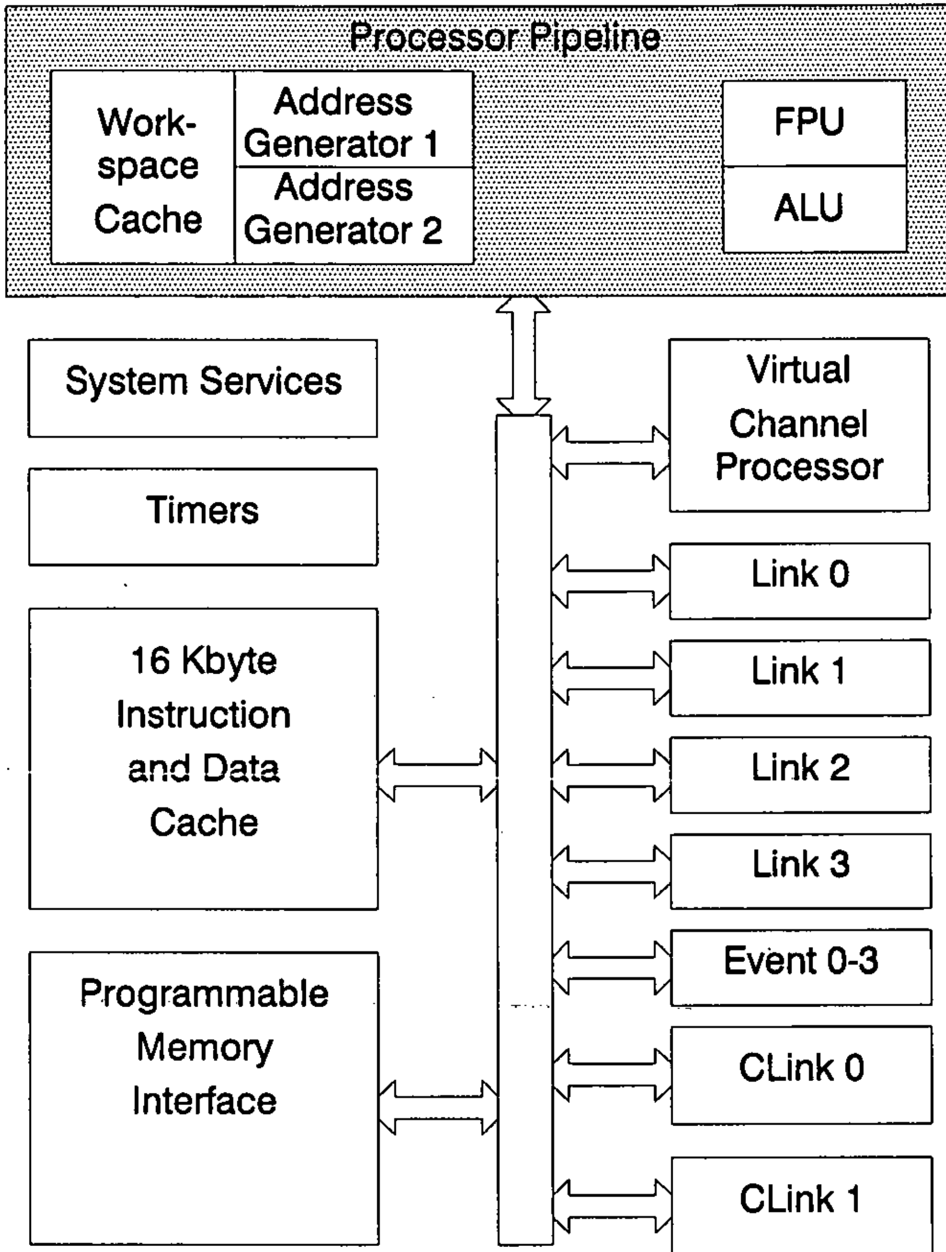


Figure 4.12. T9000 system level architecture.

Existing transputer systems overcome this problem by the use of software communications harnesses supported either as part of the operating environment, as with Helios, or as an explicit utility such as TINY. Figure 4.14 illustrates one solution using TINY and shows the routing processes that run concurrently with the user processes on each network processor.

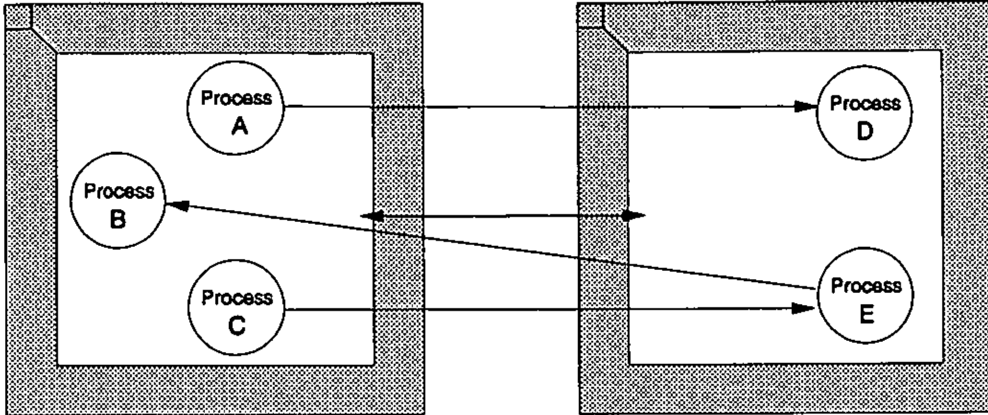


Figure 4.13. A typical interconnection problem.

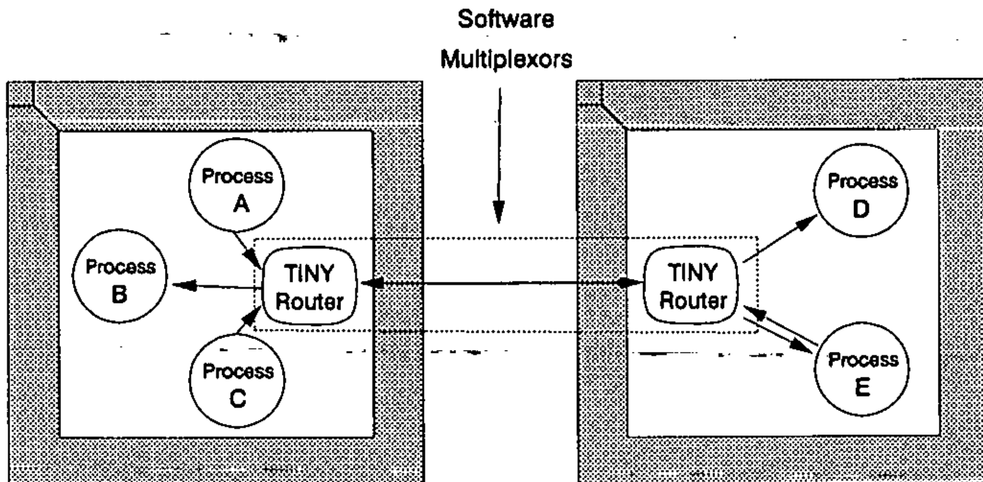


Figure 4.14. The use of TINY to provide virtual communication channels.

The T9000 solution adds the necessary multiplexing capability to permit any number of processes to access each link and transparently share the physical link. This is achieved through a dedicated hardware process which runs concurrently with the CPU and supports a large number of virtual channels on each link. These virtual channels behave in the same manner as the soft channels which form the inter-process links on a single transputer and thus the T9000 has effectively

removed any differentiation between software and hardware channels. Therefore at a schematic level, as shown in figure 4.15, the T9000 solution to the above problem looks very similar to that delivered by TINY.

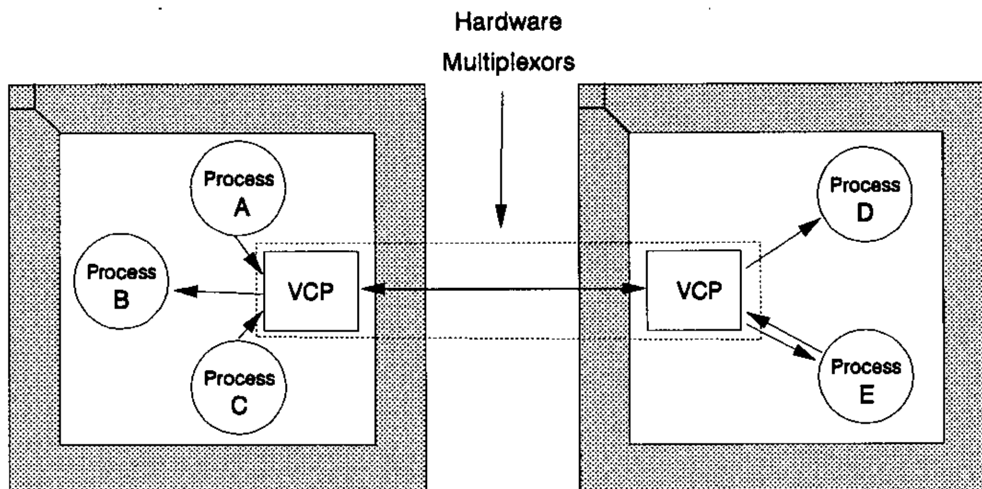


Figure 4.15. T9000 hardware based communication system

However the hardware provision of virtual channels relieves the need for software routing harnesses such as TINY, thus improving communications performance whilst reducing the code required on each processor. A special purpose routing chip, the C104, permits large arbitrary and hierarchical networks to be built. Message routing through the network is determined using an *interval labelling* algorithm[54] which claims to provide an inherently deadlock free communications strategy.

From an operational aspect, the T9000 will retain its support for all of the existing transputer languages and toolsets by virtue of the instruction set compatibility. A range of system software in terms of operating systems and real-time kernels will also be supported, particularly reflecting the needs of the embedded systems market. These include VRTX32 from Ready Systems and a distributed UNIX environment from Chorus[55].

Even though a complete technical specification will only be documented and made available with the release of T9000 silicon in 1992, it should be more than apparent from existing data that the combination of processing power, multiprocessing capabilities, standard and custom software support will take the T9000 successfully into the 1990's. But as with all new products, the key to its good fortune, lies not only in its capability and cost, but equally in the marketing policies adopted to sell it. To this end INMOS will have to work very hard to stave off the competition.

4.4 Summary

For all that parallelism is inherent in so many problems, it is not always easy to visualise, especially for a newcomer to the field, and it often requires a fresh train of thought to be applied before it can be successfully harnessed. The difference in performance that can be observed between an efficiently and inefficiently coded parallel program is generally far greater than for a sequential program. A fact which is all too often borne out with personal experience. At present, the development of parallel processing hardware is far in advance of its associated software and until there is a concerted effort in producing portable development tools and operating environments which abstract the programmer from the specific architectural details, then the true power of this technology will fail to be realised.

This chapter has provided a brief overview of various type of vector, parallel and multiprocessor computing facilities as they exist at the present time. It has also looked at the hardware and software development for one parallel processing machine in particular, the INMOS transputer, and suggests how this relates to the current status of the parallel applications market. Finally, a description of the next generation transputer, the T9000, has been given with an assessment of how this device will broaden the use of parallel and multiprocessing facilities outwith its traditional home ground of academic and industrial research. The next chapter focuses on a specific application of the INMOS transputer and illustrates how the benefits and drawbacks presented above manifest themselves in a commercial problem.

References

1. K. Hwang and F. A. Briggs, "Computer Architectures and Parallel Processing", McGraw-Hill, New York and London, 1984.
2. M. Flynn, "Some Computer Organisations and their Effectiveness", IEEE Trans. Comput., Vol. C-21, No. 9, 1972, pp948-960.
3. J. E. Shore, "Second thoughts on parallel processing", Comput. Elect. Eng., No. 1, 1973, pp95-109.
4. P. Chow and J. Hennessy, "RISC architectures", Computer Architecture - Concepts and Systems, Ed. V. M. Milutinovic, North-Holland, 1988, pp48-83.
5. G. S. Almasi, A. Gottlieb, "Highly Parallel Computing", The Benjamen/Cummings Publishing Co., 1989, pp279-300.
6. Thinking Machines Corporation, "Connection Machine CM-200 Technical Summary", June, 1991.
7. J. L. Potter, "The Massively Parallel Processor", MIT Press, 1985.
8. "G. Wilson, "An Introduction to Parallel Processing", Past, Present Parallel, eds A. Trew and G. Wilson, Springer-Verlag, 1991, p8.
9. C. L. Seitz, "The Cosmic Cube", Comm. ACM, Vol. 28, No. 1, 1985, pp23-23.
10. Intel Scientific Computers, "The INTEL iPSC/2 System - Product Information", August 1987, p14.
11. L. J. Clarke and G. V. Wilson, "TINY: An Efficient Routing Harness for the Inmos Transputer", Edinburgh Parallel Computing Centre Technical Report, EPCC-TR90-03, 1990.
12. R. W. Hockney, "MIMD computing in the USE 1984", Parallel Computing, Vol 2, 1985, pp119-136.
13. A. Trew and G. Wilson, "Past Present, Parallel", Springer-Verlag, 1991.
14. INMOS Ltd, "The T9000 Transputer", Product Seminar, June, 1991.
15. R. Dettmer, "The lean machine", Electronics and Power, August, 1985, p554.
16. "The Transputer Instruction Set - A Compiler Writers Guide", Inmos Ltd, Prentice Hall, 1988, p83.

17. "Transputer Reference Manual", Inmos Ltd, Prentice Hall, 1988, p29.

18. "Transputer Reference Manual", Inmos Ltd, Prentice Hall, 1988, p5.

19. "Transputer Architecture : Reference Manual", Inmos Ltd, Bristol, July 1987, p7.

20. G. S. Almasi, "Overview of Parallel Processing", Parallel Computing, No. 2, 1985, p201.

21. S. G. Akl, "The Design and Analysis of Parallel Algorithms", Prentice Hall, 1989, pp21-30.

22. G. Amdahl, "The validity of the single processor approach to achieving large scale computing compatibility", AFIPS Joint Comput. Conf, No. 30, 1967, pp483-485.

23. A. Gibbons, W. Rytter, "Efficient Parallel Algorithms", Cambridge University Press, 1990, pp88-141.

24. I. Plander, "Parallel Processors and Multicomputer Systems", Algorithms, Software and Hardware of Parallel Computers, Eds, J. Miklosko, V. E. Kotov, Springer-Verlag, 1984, pp273-320.

25. V. Sarkar, "Partitioning and Scheduling Parallel Programs for Multiprocessors", Pitman Press, 1989, pp101-146.

26. C. D. Howe, B. Moxton, "How to Program Parallel Processors", IEEE Spectrum, September 1987, p41.

27. INMOS Ltd, "OCCAM 2 Reference Manual", Prentice Hall, 1988.

28. INMOS Ltd, "Transputer Development System", Prentice Hall, 1988.

29. "Transputer Reference Manual", Inmos Ltd, Prentice Hall, 1988, p8.

30. A. Burns, "Programming in OCCAM 2", Addison-Wesley, 1988, pp150-163.

31. INMOS Ltd, "D705 OCCAM 2 Toolset", 1988.

32. Meiko Ltd, "CSTools for MeikOS", April 1991.

33. R. J. Allan, L. Heck, "Fortnet : a parallel FORTRAN harness for porting application codes to transputer arrays", Proc. Applications of Transputers 1, IOS Press, August 1989, pp82-89.

34. L. J. Clarke, G. V. Wilson, "TINY: An Efficient Routing Harness for the INMOS Transputer", Edinburgh Parallel Computing Centre Tech. rep, EPCC-TR90-04, 1990.

35. INMOS Ltd, "Transputer Architecture - Reference Manual", Bristol, June, 1987.

36. G. V. Wilson, "Linda-like Systems and their Implementation", Edinburgh Parallel Computing Centre Technical Report, EPCC-TR91-13, 1991.

37. A. Sharma, S. Kumar, "CODE : An OCCAM Development Environment", Parallelogram, October 1990, pp14-15.

38. F. Mourlin, E. Cournarie, "A Graphical Environment for OCCAM Programming", Proc. Applications of Transputers 1, IOS Press, August 1989, pp252-261.

39. TBUG Users Manual, 3L Ltd, 1989.

40. N. Spicer, "Windows File Server", Parallelogram, September 1989, pp18-19.

41. D. J. Wallace, "Supercomputing with Transputers", Supercomputer, No. 36, 1990, pp120-132.

42. Parsytec Ltd, "Parallel Products", 1990, p16.

43. "The Helios Operating System", Perihelion Software Ltd, Prentice Hall, 1989.

44. M. Charnley-Fisher, "Helios Clinic", Parallelogram, June 1990, p24.

45. D. Hart-Davis, "Helios : Are You Being Served?", Parallelogram, April 1991, pp15-18.

46. A. J. MacDonald, "Integrating CAM and Process Simulation to Enhance the Analysis and Control of IC Fabrication", Ph.D. Thesis, Electrical Engineering Department, Edinburgh University, 1989.

47. R. Allan, "Distributing the Load", Parallelogram, August 1991, p18-19.

48. J. Boffey, "Idris Version 5.0", Parallelogram, August 1991, pp16-17.

49. Parsys Ltd, "IDRIS Operating System - Product Profile", August 1989, p2.

50. M. Hatch, "Complete System on Silicon", Electronics Weekly, 19 September 1990, p20.

51. N. Spicer, "Late Developer", Personal Computer World, November 1989, pp182-186.

52. R. Causey, "The Design of the T9000 Transputer", Electronics Weekly, 24 April, 1991, p14.

53. INMOS Ltd, "The T9000 Transputer - Product Overview Manual", Bristol, 1991, p55.

54. C. Dyson, "Inmos H1 Architecture Revealed", New Electronics on Campus, Spring 1991, p8.

55. G. Wilson, "Little Chips Muscle in on the Big Game", The Independent, 27 May 1991, p12.

No experiment is ever a complete failure - it can always serve as a negative example.

Chapter 5

Parallel Implementation of Process Simulation Code

5.1 Introduction

The requirement for and application of process simulation to the development and manufacture of semiconductor products has been highlighted in previous chapters. However, as was illustrated in chapter 3, the CPU usage and/or elapsed execution times for a typical modelling task provides a substantial disincentive to perform a rigourous simulation schedule for a given problem. With this in mind, there is good reason to find a suitable alternative computing strategy and environment which will enable the full potential to be gained from process simulation techniques.

The relatively regular structure of process simulations suggested that the use of either vector or parallel techniques might be profitable. It was therefore decided to investigate the suitability of the cheap parallel performance available in transputer based systems for the computational requirements of process simulation.

The software chosen to be investigated during the course of this work was SUPREM II from Stanford University. A one-dimensional simulator was chosen because this provided a less complex problem to partition and it was invisaged that the methodologies used could be generalised to two-dimensions at a later date. Although this program has limitations in the light of current fabrication technologies, such as the ability to model only silicon and silicon dioxide layers, this does not affect the basic concepts which are of importance to producing a parallel version of the code. The models included in the software are mature and relatively well understood and are solved numerically using finite difference techniques, in common with most of the other one-dimensional simulators available. Finally, the easy access to source code (which is public domain) made SUPREM II a favoured

choice, in contrast to SUPREM 3 where the source code is only available under licence from the vendor at an exorbitant cost over an above that of the executable code.

In order to obtain superior performance from the new version of SUPREM II without disproportionate investment of resources, a few basic issues must be investigated before redeveloping the software. The prime task was to determine the most computationally intensive sections of the algorithms, since these would gain the greatest benefit from parallel execution. Another important issue was to assess how much of the original FORTRAN would need to be re-written or changed into OCCAM to achieve an operational package. The OCCAM compiler produces more efficient code than the FORTRAN compiler, in terms of execution time, by a factor of approximately two. However, it would have been wasteful of time to re-write excessive quantities of SUPREM II, especially given the inherently serial nature of so many of the operations. The final major issue to be resolved was that of partitioning the relevant parallel modules over an arbitrary network of transputers to ensure simple and efficient inter-process communications and enable the network to be simply expanded if so required.

This chapter addresses all of these points with respect to SUPREM II. It then reports on the results obtained from the particular solution implemented and discusses these in the light of the knowledge gained during the re-development of the software. Finally some ideas on how a future version could be improved are also provided.

5.2 One-Dimensional Simulation

The continuous physical processes modelled by SUPREM II are approximated numerically using finite difference techniques. Each layer of the physical structure is divided into a series of cells which are associated with the nodes on a one-dimensional grid. Information is then stored about the impurity concentrations for each dopant type associated with a cell in the structure. Every cell in the interior

of a layer is centred about a single node on the grid. The cells at the ends of a layer have one cell boundary at the end node and the other boundary exists halfway between the end node and the adjacent interior node as shown in figure 5.1.

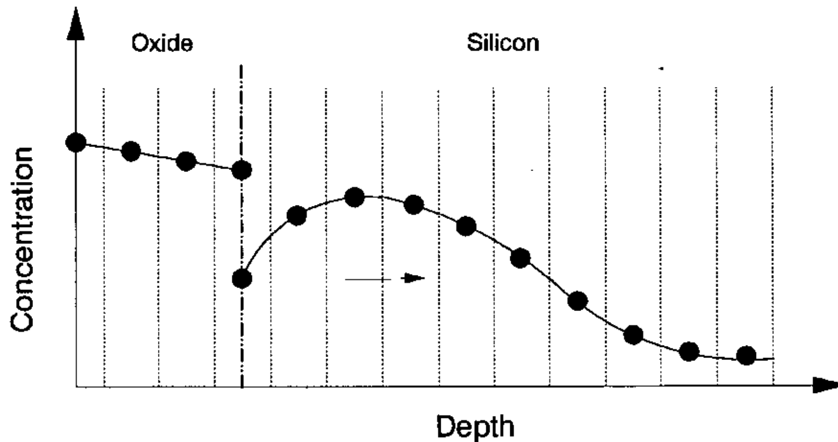


Figure 5.1. Cell/Node structure within SUPREM II.

The silicon wafer is represented in SUPREM II by up to 400 nodes divided into three regions, each of which may possess a different grid structure. One region is allocated for any oxide which may be present and two silicon regions may be defined. This permits a fine grid to be placed close to the silicon - silicon dioxide interface for an accurate representation of steep or narrow impurity profiles, whilst a course grid can be retained for the bulk, so using fewer grid points in total. Thus the overall grid structure takes the form of that shown in figure 5.2 below.

This grid structure can and does change during the course of a simulation with the addition or removal of nodes from the oxide region or the movement of the silicon/silicon dioxide interface as an oxide region is grown. Additionally, the extent of the simulation region may be altered as simulations proceed and the relevant grid spacings altered by the user as well as by the program.

The other important issue which will influence what modifications must be made to the code, is that of the data-structures. All the information pertinent to the structure during the simulation must be retained and remain accessible by all

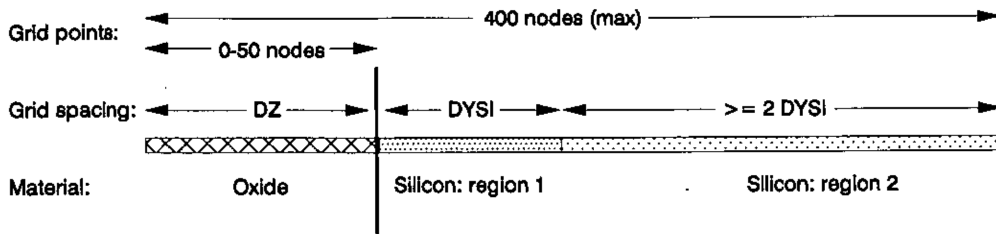


Figure 5.2. Grid structure within SUPREM II.

the relevant processors. The SUPREM II data structure is divided into five main blocks, of which four are actively used during program execution and the remaining one was reserved for an optimisation routine which did not materialise before SUPREM II was superseded by SUPREM III. These are held as *common blocks* within the FORTRAN enabling global access to the data contained within them by all subroutines. Whilst this is an established and well-proven method of data storage within the sequential programming community, it does present difficulties when applied to parallel programming architectures. Those which possess no shared system memory, as found in a standard transputer network, cannot implement this style of data storage efficiently and alternative methodologies are therefore necessary. At the present point in the discussion, the data structures will be introduced in their sequential format and any modifications will be presented at a later stage.

The first data block contains six individual arrays, each of 450 real words. Four of these arrays use the first 400 points to store up to four individual dopant concentrations in each cell and the remaining 50 points contain global information pertaining to the respective element. The fifth array is a 'scratch' space for general calculations. The last array contains the spatial information for all the data points showing the physical distance between adjacent nodes on the grid which uses 400 points and the last 50 points contain physical parameter pertaining to the current step, such as temperatures and times.

The second data block contains three arrays of type logical, integer and real respectively, which describe grid, substrate and I/O details. The third data block holds all the model parameters, which are initialised to SUPREM II's default values on program start-up but can be subsequently revised during program execution. The final data block consists of four 410 word arrays which are used as general working arrays by any routines needing such data storage throughout program execution.

In order to partition these data blocks, it is important to establish what information may be required by each routine throughout a particular simulation. Additionally, it will be necessary to ascertain which data segments are required by the entire simulation region, i.e. *global*, and which are only relevant to a given physical region, i.e. *local*. As a starting point for this latter exercise, it is beneficial to visualise the data structures and figure 5.3 displays each of the four main data blocks, indicating the global and local segments within them.

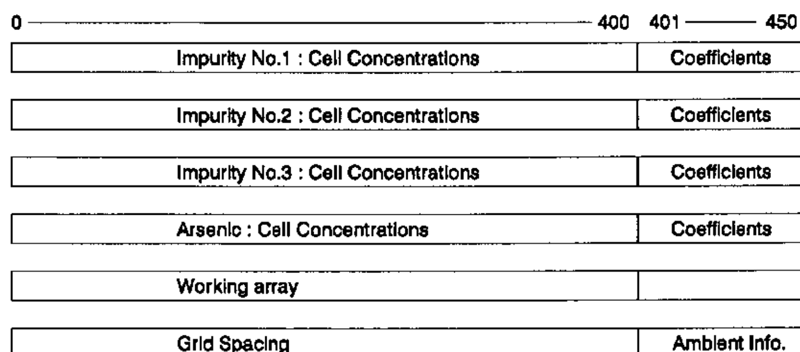


Figure 5.3. SUPREM II Common Block layout.

5.3 Partitioning SUPREM II code

As has been mentioned, many operations within process simulation are inherently sequential, such as input/output routines, and therefore cannot be made to benefit from parallel execution. Although such routines could be re-written in OCCAM to take advantage of the superior compiler performance, the time required to achieve this is not justified by the additional performance attained. On this basis

it is necessary to assess each of the major program models in terms of their computational requirements and of the suitability of the algorithms to parallel execution. From this, it will then be possible to partition the program up into sequential and parallel segments and subsequently decide how to distribute the parallel segments over a transputer network.

Ion implanted impurity distributions are described by either Gaussian or Pearson distribution functions. These distributions are described by the first two or three central moments which are the projected range, R_p, and one or two standard deviations, s_1 and s_2, depending whether a one- or two-sided profile is being utilised. These moments are calculated from LSS theory and extracted at runtime from look-up tables within the code and hence the model is very quick to execute and requires little computation.

Thermal oxidation is described by the familiar Deal and Grove formula [1] given in chapter 3, with the linear and parabolic coefficients being determined for each time step during the total oxidising time. Since this process requires an elevated temperature, its evocation is from within the diffusion module whilst calculating the redistribution of any dopants already present within the structure. Although the coefficients include terms for many more process parameters than was originally specified, this still does not constitute a particularly computationally intensive operation.

Etching steps simply involve the removal of all or part of the existing oxide layer from the structure. If an elevated temperature is specified then the diffusion models are invoked as well to cover any dopant redistribution that may occur, otherwise there is no genuine computation associated with this step. There is only a low temperature oxide deposition step within SUPREM II, which merely adds the desired quantity of doped or undoped oxide to the top of the current structure. By nature of the enforced low temperature, there is no need to even consider dopant redistribution during the operation.

In terms of computational requirements, the diffusion model provides a contrast to those models already mentioned. The algorithm for this model will be expanded later in this section, but involves solving the one-dimensional continuity equation with time- and concentration-dependant diffusion coefficients for each time step, Ft, throughout the entire course of a high temperature step. Numerically, this is achieved by the formation of a large sparse matrix of diffusion coefficients from which the updated impurity concentrations can be calculated in terms of their current concentrations. This then becomes both a data and numerically intensive operation to perform. In addition, the diffusion models can be invoked during *any* high temperature steps during the process to be simulated, which means that the computational performance of this step may dominate that of the entire simulation.

The elapsed times for each of the major program models within SUPREM II are tabulated below and have been taken from a typical CMOS process simulation. Measurements are presented for simulations run on both T414 and T800 transputers (i.e. without and with floating point units (FPU) respectively). These substantiate the relative computational complexities of the individual models, as proposed by analysis of the underlying algorithms. It is also important to note the performance gained by use of the dedicated FPU hardware. This factor is obviously dependant on the data types used within a particular application, but with predominant use of real data in process simulation, it provides a significant performance increase from a marginal increase in hardware cost.

Program Model	T414 (secs)	T800 (secs)
Ion Implantation	0.5	0.2
30 min Diffusion - no oxidation	31.1	0.58
30 min Diffusion - oxidation	35.2	0.68
300 min Diffusion - no oxidation	364.2	18.7
300 min Diffusion - oxidation	417.5	21.45
1μm SiO_2 etch - low temperature	0.1	0.05
1μm SiO_2 etch - 280°C	0.1	0.06
1μm SiO_2 deposition	0.1	0.05

Table 5.1 SUPREM II execution times for a typical CMOS process on T414 and T800 transputers.

As can be seen, it is the diffusion algorithm which assumes the most computationally intensive part of SUPREM II. Given that the oxidation routines are also called from within this module, when applicable, it rapidly becomes apparent that this code segment would benefit greatly from some means of increased performance. Therefore it was decided to split the execution of the code at a high level such that the diffusion model and any associated routines would run in parallel and the remainder of the code would continue to be executed serially on a single processor. This means that at each occurrence of a diffusion step in the simulation input file, the necessary data-structures will have to be downloaded onto the transputer network, the calculations performed and then all the data-structures loaded back up on to the main processor.

The next step in partitioning the code is to assess each stage of the diffusion algorithm from a suitably high level and to consider the data requirements and likely frequency of execution. In order to assist with this task, it is best to view the algorithm in both a pseudo-code format and diagramatically from the point of the

routines involved and from where they are called. Figure 5.4 shows a brief outline of the diffusion algorithm in a pseudo-code form, whilst figure 5.5 displays the hierarchy of the software subroutines involved. An explanation as to the function of each of the subroutines is then presented.

```
WHILE  time.so.far < total.step.time
 calculate next time step - Ft
 IF oxidising ambient
      calculate additional oxide thickness
 calculate global diffusion parameters
 DO for each impurity present
      form diffusion coefficient matrix
      solve matrix system for new dopant concentrations
      update impurity concentrations database
 time.so.far = time.so.far + Ft
```

Figure 5.4

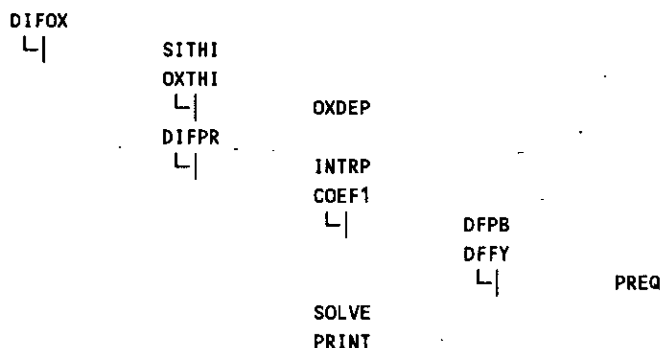


Figure 5.5

A brief summary of each of the subroutines mentioned above is presented to assist understanding of the partitioning of this code segment.

DIFOX Top level routine for oxidation/diffusion segment. Calculates the time step, Ft, for each iteration and calls *DIFPR* with the relevant data for all the impurities present.

SITHI Calculates thickness of unconsumed silicon.

OXTHI Calculates oxide growth for time step Ft.

OXDEP Calculates the linear and parabolic oxide growth rate constants.

DIFPR Controls diffusion calculation for each impurity in turn.

INTRP	Interpolates for dopant concentration values around moving oxide/silicon interface during oxide growth.
COEF1	Sets up diffusion coefficients for matrix system representing the discretised continuity equation.
DFPB	Prepares diffusion coefficients in presence of phosphorous.
DFFY	Prepares diffusion coefficients in absence of phosphorous.
PREQ	Modifies diffusivity due to effect of the clustering of impurity atoms.
SOLVE	Solves the matrix system for new dopant concentration values.
PRINT	Prints summary of matrix solution.

Within this set of subroutines, the only computationally intensive one is the matrix solver, *SOLVE*. This is instanced once for each impurity present during every time step, Ft, and accounts for the majority of the CPU time during a diffusion calculation. If *SOLVE* is the only routine to be converted to run in parallel, then the entire dopant concentration data-structure and any relevant global parameters would have to be transferred into and out of the transputer network up to four times per time step (since the maximum number of impurities is limited to four). Although this would require the least amount of code to be re-written in OCCAM, the inefficiency caused by the excess data communications would seriously impinge on any performance benefit achieved in the calculation time. Were it possible to use shared memory amongst the processor network, then this data transfer becomes superfluous and such a methodology would provide a quick and efficient test of SUPREM II's suitability for parallel execution. However, this is not the case and an alternative partitioning point must be found.

The next possible breakpoint that might prove more efficient, when viewing the hierarchy of subroutines displayed above, is to perform all the calculations for a given time-step before transferring the necessary data-structures. This would involve converting *DIFPR* and all its associated subroutines to run in parallel. However, a typical diffusion process performed under an oxidising ambient may

require around 100 time steps, Ft, in order to achieve the total step time, whilst constraining any errors occurring in the numerical solution. So although 4 data transfers can be eliminated per time step, this option does not eliminate the communication bound problem already being faced; thus a yet higher level of partitioning appears to be required.

The next and final choice remaining open at this point is to transfer all computation to a parallel mode as soon as a diffusion/oxidation process step is invoked. Under this scheme, only a single data transfer to and from the transputer network is required for the calculation, regardless of the number of impurities present or the number of time steps necessary. Therefore all code associated with *DIFOX* has to be re-written in OCCAM to facilitate the inter-process communications needed throughout its execution. Although basic communications extensions have been provided in the transputer release of FORTRAN, these do not provide the same comprehensive capabilities that are available from OCCAM.

Given that the breakpoint between sequential and parallel execution has been established, it is now necessary to determine how the physical problem can best be partitioned into an arbitrary number of *processes*. Then these processes must be configured to run on a number of actual processors with, as yet, undefined connectivity. There can be much to be gained in partitioning a problem into a particular number of *processes*, whether or not that number of *processors* is available because any subsequent expansion or contraction of the transputer network only requires altering the number of *processes* running on each *processor*. For the process simulation problem, the simplest partitioning scheme is to divide the semiconductor structure up into a particular number of abutting segments, each containing a proportion of the 400 nodes on the one-dimensional grid. The actual number of cells per segment is flexible, providing the total workload per processor ensures that a sufficiently high computation to communication ratio is maintained. The most basic and coarse partitioning is to divide the 400 nodes as evenly as possible over the number of available processors. This can be achieved statically, but a more

even distribution of tasks will result if it is done dynamically each time a diffusion step is performed. In this way any change in the number of nodes *actually* used will be taken into consideration.

Using this simple partitioning scheme, the data requirements for each processor can be surmised as follows. Firstly, every processor will require a copy of all the global parameters, common to all sections of the simulation and this will also need to be updated on all processors whenever a change is made on a particular machine. Secondly, each processor requires a local data set, pertaining to the particular section of the simulation which it is performing and also access to the boundary data, for its nearest neighbours. Where possible it is preferable for each machine to retain a copy of the data referring to the first one or two nodes on the adjacent machines, thus creating an overlap which can prevent the need for some inter-processor communications. Hence the data stored across the processor network can be thought of in terms of figure 5.6.

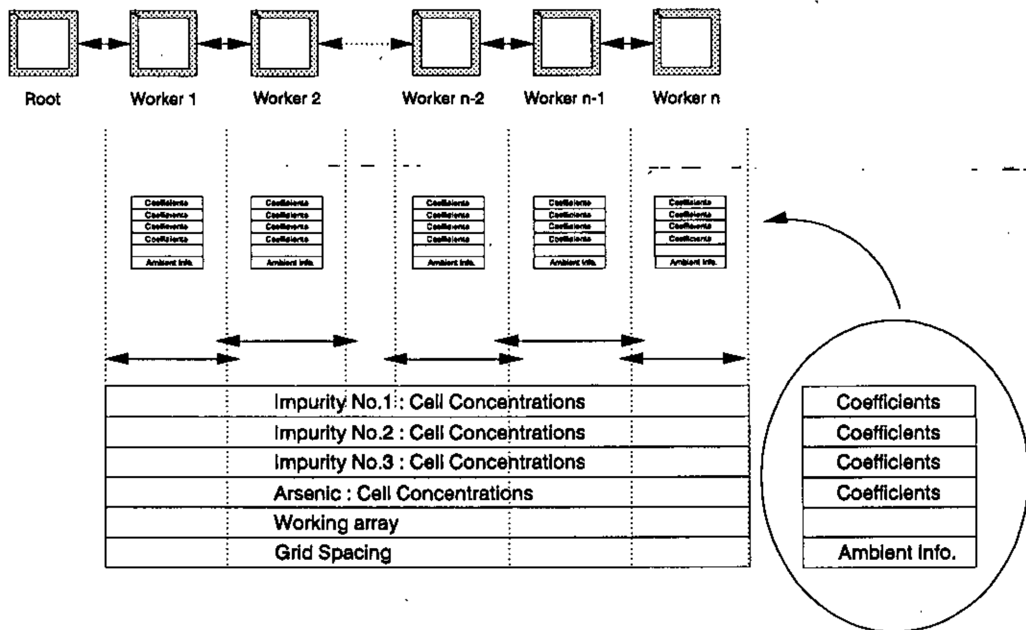


Figure 5.6. Partitioned SUPREM II data-structures.

The modifications required to the original software in order to achieve this partitioning come in four main areas. Firstly, there is a series of additional communication modules which transmit and receive the common data blocks between the single serially executing processor and the processor network operating in parallel. Secondly, there is another set of communication routines, this time dedicated to data exchange between the processors executing in parallel. Thirdly, there is the incorporation of additional logic within the scope of the original subroutines which ascertains whether a particular operation is relevant to the particular physical segment being modelled on that processor. For example, the calculation of oxide growth is only relevant to processors which contain the nodes from the surface down to the oxide/silicon interface and may be omitted on the remaining processors. The final area of change is in the inclusion of a parallel matrix solution algorithm to replace the gaussian elimination scheme presently employed by SUPREM II, since this possesses no inherent parallelism and would in fact run slower across a transputer network.

These modifications could be coded up in the FORTRAN of the original software, but it was decided to re-write all those routines which execute in parallel in OCCAM. This would not only improve their speed of execution, but also enable code debugging to be performed in an entirely OCCAM based environment. The problem associated with this task is that the translation of unstructured FORTRAN constructs, such as loosely used GOTO's, into the structured OCCAM syntax is not always that simple. Within OCCAM, this often requires the use of many conditional constructs in order to enable the program flow to be replicated. As was found under these circumstances, 15-20 lines of FORTRAN could easily result in the generation of 60-100 lines of OCCAM. Thus the final parallel code required 8500 lines of OCCAM to be written and debugged in order to implement the parallel version of SUPREM-II.

5.4 Parallel Matrix Solution Method

It has been shown that dopant re-distribution throughout the semiconductor structure may be modelled by the one-dimensional continuity equation. The numerical solution to this partial differential equation can be represented by the following basic matrix system where **A** represents the diffusion coefficients, **u**, the present impurity concentrations and **x**, the new concentrations after a given time period.

$$A \cdot \underline{x} = \underline{u} \tag{5.1}$$

The one-dimensional nature of the problem means that the impurity concentration within a given cell is only dependant on its nearest neighbours and hence equation 5.1 may be expanded into a sparse, tri-diagonal matrix system as indicated within figure 5.7.

$$\begin{pmatrix} a_1 & b_1 & & & & \\ c_2 & a_2 & b_2 & & & \\ & c_3 & a_3 & b_3 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & \cdot \\ & & & c_{n-1} & a_{n-1} & b_{n-1} \\ & & & & c_n & b_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \cdot \\ \cdot \\ x_{n-1} \\ x_n \end{pmatrix} = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \cdot \\ \cdot \\ u_{n-1} \\ u_n \end{pmatrix}$$

Figure 5.7. Tri-diagonal matrix system.

Many methods for solving these tri-diagonal matrix systems have been reported for both vector and parallel computers [2],[3],[4]. A partition method by Wang [5] designed for tridiagonal systems has also been extended by Meier [6] for general banded systems of equations. Wang's method was chosen for this work because of the similarity in its partitioning scheme with that proposed for the

data-structures in SUPREM II. A brief outline of the method will be presented below, though a fuller description and assessment of the computational requirements should be obtained from [5].

The basic algorithm involves five passes of the matrix system to eliminate the upper and lower diagonals and ultimately provide a solution in terms of the main diagonal set of elements. Operations can be independently carried out on each block in parallel, except for the calculation of the bordering nodes which requires inter-partition communications. Firstly, the matrix **A** is partitioned in to L sub-blocks of size, K, which need not necessarily be identically sized. As an example, a matrix **A** of order 12, will be partitioned into 3 blocks of 4 elements as shown in figure 5.8.

$$
\left(\begin{array}{cccccccccccc}
a_1 & b_1 & & & & & & & & & & \\
c_2 & a_2 & b_2 & & & & & & & & & \\
 & c_3 & a_3 & b_3 & & & & & & & & \\
 & & c_4 & a_4 & b_4 & & & & & & & \\
\hline
 & & & c_5 & a_5 & b_5 & & & & & & \\
 & & & & c_6 & a_6 & b_6 & & & & & \\
 & & & & & c_7 & a_7 & b_7 & & & & \\
 & & & & & & c_8 & a_8 & b_8 & & & \\
\hline
 & & & & & & & c_9 & a_9 & b_9 & & \\
 & & & & & & & & c_{10} & a_{10} & b_{10} & \\
 & & & & & & & & & c_{11} & a_{11} & b_{11} \\
 & & & & & & & & & & c_{12} & a_{12}
\end{array}\right)
\left(\begin{array}{c}
x_1 \\ x_2 \\ x_3 \\ x_4 \\ \hline x_5 \\ x_6 \\ x_7 \\ x_8 \\ \hline x_9 \\ x_{10} \\ x_{11} \\ x_{12}
\end{array}\right)
=
\left(\begin{array}{c}
u_1 \\ u_2 \\ u_3 \\ u_4 \\ \hline u_5 \\ u_6 \\ u_7 \\ u_8 \\ \hline u_9 \\ u_{10} \\ u_{11} \\ u_{12}
\end{array}\right)
$$

Figure 5.8. Partitioned block matrix.

The lower diagonal elements are then eliminated within each block except for a set of *fillins*, f_i.This leaves a more-or-less lower triangular matrix as shown in figure 5.9

The upper diagonal elements are now eliminated in parallel, except for an additional set of fillins, g_i, leaving the matrix in the form shown in figure 5.10.

$$
\begin{pmatrix}
a_1 & b_1 & & & & & & & & & & \\
 & a_2 & b_2 & & & & & & & & & \\
 & & a_3 & b_3 & & & & & & & & \\
 & & & a_4 & b_4 & & & & & & & \\
 & & & c_5 & a_5 & b_5 & & & & & & \\
 & & & f_6 & & a_6 & b_6 & & & & & \\
 & & & f_7 & & & a_7 & b_7 & & & & \\
 & & & f_8 & & & & a_8 & b_8 & & & \\
 & & & & & & & c_9 & a_9 & b_9 & & \\
 & & & & & & & f_{10} & & a_{10} & b_{10} & \\
 & & & & & & & f_{11} & & & a_{11} & b_{11} \\
 & & & & & & & f_{12} & & & & a_{12}
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{pmatrix}
=
\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \end{pmatrix}
$$

Figure 5.9. Lower diagonal matrix, with *fill-ins*.

$$
\begin{pmatrix}
a_1 & & & g_1 & & & & & & & & \\
 & a_2 & & g_2 & & & & & & & & \\
 & & a_3 & b_3 & & & & & & & & \\
 & & & a_4 & & & & g_4 & & & & \\
 & & & c_5 & a_5 & & & g_5 & & & & \\
 & & & f_6 & & a_6 & & g_6 & & & & \\
 & & & f_7 & & & a_7 & b_7 & & & & \\
 & & & f_8 & & & & a_8 & & & & g_8 \\
 & & & & & & & c_9 & a_9 & & & g_9 \\
 & & & & & & & f_{10} & & a_{10} & & g_{10} \\
 & & & & & & & f_{11} & & & a_{11} & b_{11} \\
 & & & & & & & f_{12} & & & & a_{12}
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{pmatrix}
=
\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \end{pmatrix}
$$

Figure 5.10. Main diagonal matrix, with *fill-ins, f,* and *g*.

Two further passes of this system provide the elimination of the fillins, f_i and g_i, leaving only a main diagonal of elements, as seen in figure 5.11, which permits x to be solved for in terms of u. This final step can obviously be achieved independently in each partition.

$$
\begin{pmatrix}
a_1 & & & & & & & & & & & \\
& a_2 & & & & & & & & & & \\
& & a_3 & & & & & & & & & \\
& & & a_4 & & & & & & & & \\
& & & & a_5 & & & & & & & \\
& & & & & a_6 & & & & & & \\
& & & & & & a_7 & & & & & \\
& & & & & & & a_8 & & & & \\
& & & & & & & & a_9 & & & \\
& & & & & & & & & a_{10} & & \\
& & & & & & & & & & a_{11} & \\
& & & & & & & & & & & a_{12}
\end{pmatrix}
\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \end{pmatrix}
=
\begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \\ u_8 \\ u_9 \\ u_{10} \\ u_{11} \\ u_{12} \end{pmatrix}
$$

Figure 5.11. Final diagonal matrix, with no *fill-ins*.

The simplest partition scheme for this method places one block of elements on each processor in the network, with each block/processor containing k rows from the above matrix system. The actual value of k will depend on the number of cells being used by SUPREM II at the time of the diffusion process and the number of processors being used.

5.5 Hardware Architectures

The one-dimensional nature of the simulation problem, coupled with the proposed method of partitioning the computation, would suggest that a one-dimensional network of processors could be a productive hardware architecture. Without becoming extremely convoluted, there are only two general ways of arranging the processors to provide a one-dimensional array of computational *worker* nodes. These are loosely termed the *spaceline* and *pipeline* architectures. These present a similar array of interconnected processors for performing simulation, but differ in their approach to the communication of the major data-structures between the worker nodes and the single *root* processor which performs the serial operations.

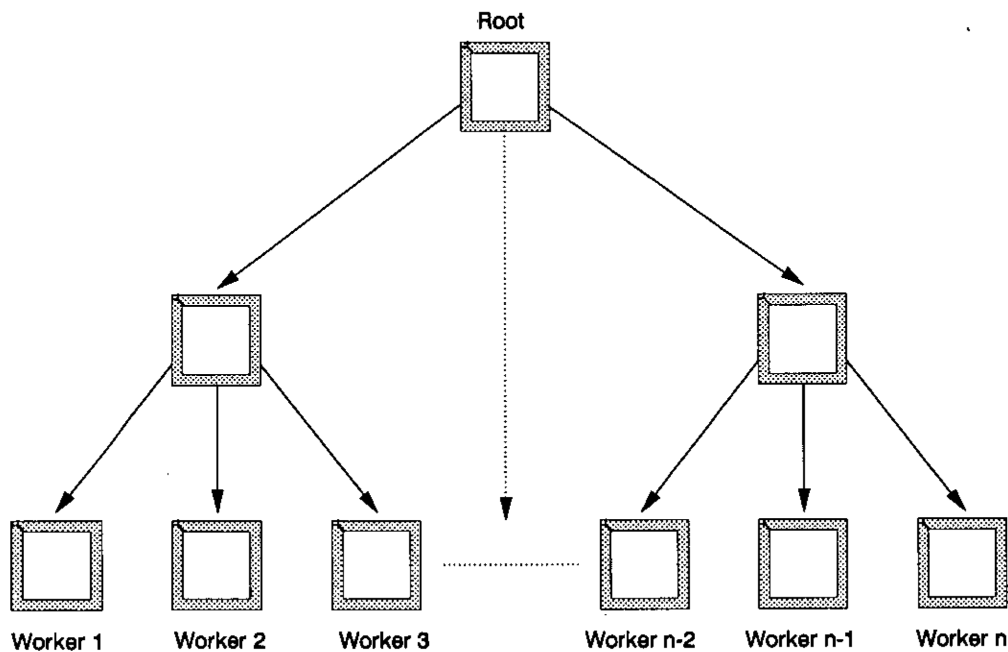


Figure 5.12. *Spaceline* transputer network architecture.

The spaceline architecture, as depicted in figure 5.12, takes the basic form of a tertiary tree structure as dictated by the four communication links available on each transputer. This produces a hierarchical network of processors linking the root to the *leaf*/worker nodes through a number of *branch* processors, with each level having a particular function to perform. There may be multiple levels of branch processors, whose sole task is to partition the data-structures and distribute them to the next level down. Similarly these processors also perform the reverse operation of combining the data partitions on their return to the root processor. Obviously this results in an extremely rapid load/unload of the worker processors, since the distribution is being performed in parallel across each level of the spaceline. The nearest neighbour communications are also very efficient, with direct links between every worker node. However, it does produce a somewhat inefficient use of processors, in that for a total of N processors, $(\frac{N}{3} + 1)$ are wasted in terms of the actual computation associated with the simulation. Furthermore, this architecture is extremely inflexible with respect to expansion because it requires transputers to

be added on in groups consisting of n processors, where n is a power of three. This last issue entirely defeats one of the major benefits of transputer based systems, which is that of simple re-configuration to run variable sized networks. Prolific amounts of complex logic are required to ensure that the spaceline architecture can be expanded or contracted with little or no user assistance.

In spite of these criticisms and because of pre-conceived worries over communication performance, this architecture was implemented initially using a total of nine processors (i.e. with six worker nodes for the simulation). Sufficient hardware existed for this size of network and it operated successfully. However, due to the number of nominally 'wasted' processors within the system, a larger network of nine worker nodes was not attainable with the available hardware. As will become evident in §5.6, it was important to investigate the network performance over a wider number of processors and so an alternative architecture had to be implemented which could make greater use of the total number of available processors. Therefore, the pipeline architecture as shown in figure 5.13 was also implemented.

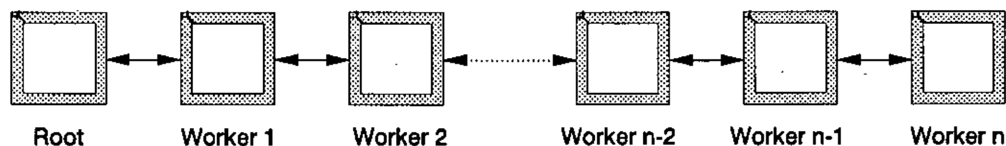


Figure 5.13. Pipeline transputer network architecture.

The major published drawback with this architecture is that it results in a slower load/unload of the transputer network which will eventually dominate overall systems performance as the pipeline length is increased . The actual length of the pipeline at which this occurs depends on the computational and communication requirements of the computation. However, in favour of the use of this system is the fact that for the spaceline architecture, the total load/unload time for the data-structures was only 60ms [7] in comparison to a computational time of 10mins or 10s on the T414 and T800 transputers respectively. This overhead can be

regarded as negligible, providing that the network size remains at this order of magnitude. The pipeline architecture has a far higher degree of processor utilisation in computational terms than the previously mentioned design, with at most one processor not being available as a worker node. In common with the spaceline system, nearest neighbour communications are efficient to implement.

Conceptually, each transputer has four bidirectional links with which to communicate to either other transputers or other peripheral devices. Therefore, depending on the access to all the transputer links, a duplicate communications path can be created to provide extra performance or redundancy in the event of a failure. When applied to the pipeline system, this means that two independent *physical* links with each neighbouring processor can be made, one to provide global network-wide data communications and the other to provide nearest neighbour communications, as shown in figure 5.14. This scheme greatly simplifies the protocols necessary to send and receive any data from around the network. The local data channel requires only a very basic protocol to inform the recipient node the data type and number of data items within a message. The global channel still requires this information, but in addition the recipient node number/address must be encoded with the data.

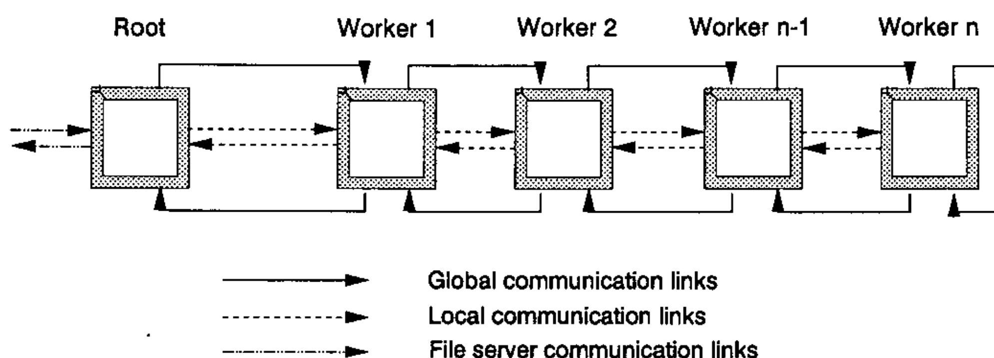


Figure 5.14. Global and local inter-transputer communication links.

The physical nature of these links is emphasised, because it is quite possible to provide a communications harness, which is transparent to the user, but that allows multiple *soft* links to be mapped onto the physical links. Communication harnesses such as *TINY* or *TITCH*, as developed on the Edinburgh Concurrent Supercomputer (ECS) [8],[9] enable applications to use many independent communication links between *processes,* whether they are mapped onto the same or different *processors*. Obviously if such harnesses are to be efficient and are not to degrade the computational performance of a network, they must be extremely carefully developed, in close association with the transputer hardware and low level system routines.

Due to the relatively simple nature of the communications demanded by the partitioning of SUPREM II, it was decided to use just the two physical channels between each processor along with the primitive protocols necessary. Initially, the network was constructed from a set of T414 transputers which made all four links available for the user to connect at will. Thereafter, the network was upgraded to run on T800 transputers in order to gain the benefit of the floating point unit. However, these came on a set of prefabricated BOO3™ boards, each of which contained four transputers and some external memory. The problem with these boards is that two of the four links on every transputer are hardwired together to form a ring system, as depicted in figure 5.15.

This constraint means that there were insufficient free links with which to form two independent communication channels between every node in the pipeline and so a single communication path had to be used. This also involved producing a substantially more complex protocol for the associated controllers, in order to combine all the necessary data traffic onto the one path. The effect was to reduce the communication performance in two ways. Firstly, the more comprehensive logic associated with the communication controllers ran slower. Secondly, the existence of only one physical data path meant that global and local communications could

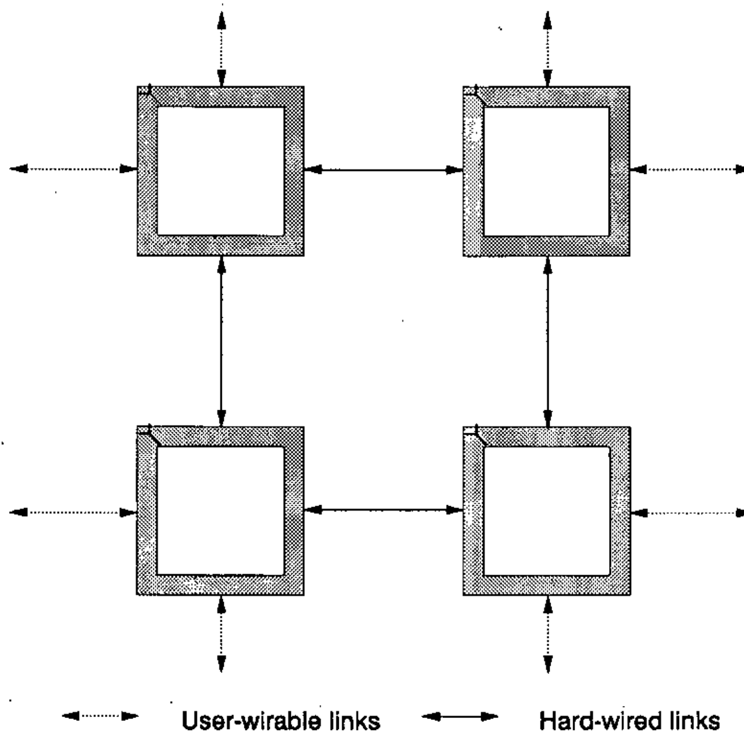


Figure 5.15. Schematic layout of BOO3™ transputer board.

no longer run in parallel over their previously independent links. The reduced communications performance was still insignificant compared to the computation times and so could be deemed acceptable.

5.6 Results

In assessing the resulting performance of the parallel version of SUPREM II, there were two main levels at which investigations were made. Firstly, the parallel matrix solver was tested without the remainder of the diffusion related subroutines being called. In order to provide the solver with a valid data set, this was loaded from the filing system where the relevant matrix coefficients had been stored from previous runs. This test would enable the increased performance over the modified gaussian elimination algorithm used by the sequential SUPREM II, to be observed. The second test case, was to analyse the performance of the complete diffusion algorithm from the point at which it was initially called in the sequential portion

of the simulation code. This would then indicate the total speed-up which had been realised by the parallel version of SUPREM II. The execution time of the original FORTRAN version of SUPREM II on a single transputer was taken as the control value on which all calculations were based. This provides a set of results which is then representative of the benefit gained solely from the change to parallel execution. Obviously, given the absolute performance of transputer hardware systems, as shown in chapter 4, the actual increase in execution times over many of the traditionally used hardware platforms will be even greater.

The next problem lies in finding a reliable method of measuring the performance of the transputer network. Unlike most multi-user operating systems such as UNIX™ or VMS™, there are no CPU monitoring functions inherently built into the system and an alternative method is required. OCCAM concerns itself with the time dimension in a far deeper manner than most conventional languages and provides an explicit *TIMER* object within its syntax[10]. This behaves like a normal communication channel except for its uni-directional nature. It can only provide an input to a variable, i.e. a user can not set a TIMER, but only read its value. Also, unlike normal channels, more than one process may input from a given TIMER, which is a necessity in order to provide time related information about a series of concurrent processes. By the use of modulo arithmetic, time differences can be measured with a resolution of 64μs on most implementations. Time delays can also be built in using the same construct, a feature which can be helpful in locating marginal communication problems.

The results in figure 5.16 show the performance of the parallel matrix solver described previously in §5.4. The matrix dimensions were varied to model a one-dimensional grid of lengths from 50 to 1000 nodes and the solution performed using between 3 and 8 transputers. The time for the serial gaussian based solution within SUPREM II for the same datasets has also been included to provide a reference benchmark.

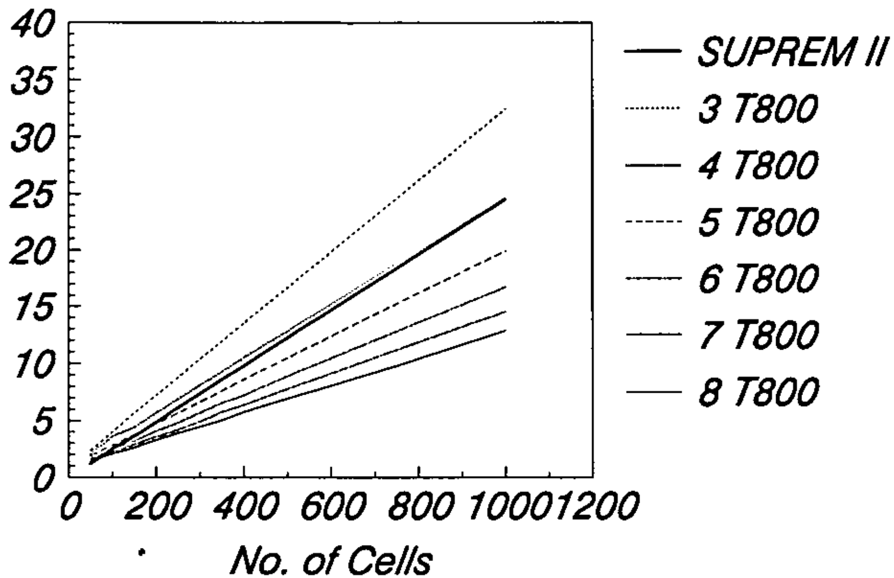


Figure 5.16. Parallel matrix solution results, showing how the execution time varies with the number of cells in the matrix for varying number of processors.

There are various aspects of this figure which deserve an explanation. Firstly, for a given number of transputers in the network, the solution time is linear with respect to the magnitude of the problem domain. The significance of this trend lies in the fact that if an imbalanced distribution of the nodes ever arises during a real simulation then the performance penalty will not be excessive. Secondly, and of greater importance from a general problem partitioning aspect, is the relevance of the relative performance of the serial algorithm. It must be noted that for the parallel algorithm implemented here, in excess of four transputers are required to provide an improved performance. This can be explained by the fact that there are many additional overheads concerned with the parallel algorithm. This can be in terms of the data communications to load and unload the network with the necessary data-structures and also those associated with globally updating any of the common parameters, throughout the course of the solution, which must be available across the network.

5.7 Discussion

This chapter has described the complete operation of taking an existing one-dimensional process simulation package, SUPREM II, and re-developing the code where applicable, to run with superior performance on a transputer network. This goal has been successfully achieved, as indicated by the results shown in §5.6, though it must be emphasised that this particular implementation should not be considered to be definitive or the most efficient. However, the prime motive behind the project brief was of feasibility rather than ultimate performance. In assessing the work accomplished and results attained, there are many points which arise that require clarification or expansion.

In general, problems which possess either algorithmic or geometric parallelism require a solution which is entirely application specific in order to become concurrent. As with any individual solution, this consumes vast amounts of resources in comparison to a standard 'off-the-shelf' solution which may require little or no alteration. The performance gained from the parallel version of SUPREM II required many man-months of software development. Even allowing for the inefficiencies caused by traversing the learning curve for the new software and hardware environments at the outset of the project, the superior execution time has been extremely costly to realise. To put the magnitude of the task into perspective, approximately 16 man-months were needed to produce to final operating version of the software used to realise the results outlined in §5.6. If 50% of this were discarded to allow for the acquisition of sufficient skill in using the transputer hardware and software development packages, this still leaves 8 man-months of 'proficient' software development which might appear excessive for the modification of the few subroutines discussed in §5.3.

As a result of electing to re-code the relevant diffusion related subroutines in OCCAM, it was necessary to understand the original FORTRAN code to a far greater depth than would otherwise have been required. In light of the experience gained during this work, a much reduced volume of re-coding would be undertaken for a similar project in the future, which would save a lot of time. The actual re-writing

of the FORTRAN was made particularly difficult due to its extremely unstructured and undocumented nature and hence failure to map elegantly into the structured OCCAM syntax. Coupled to this was the fact that OCCAM and FORTRAN store any multidimensional data-structures in conflicting formats which entailed a conversion whenever large arrays were being passed between the two languages. FORTRAN uses a *column major* format whereas OCCAM uses a *row major* format. Initially, because the code was being re-written, it was intended to alter any relevant array subscripts within the OCCAM version, thus saving the need for a conversion process. However, in order to efficiently partition the data-structures into segments [11] for each transputer, OCCAM required the alternative data storage scheme to be used, making data conversion a prerequisite.

The next major item which required attention was that of trouble-shooting and debugging a parallel system. The concurrent execution of processes provides an additional group of stumbling blocks, such as communication deadlocks etc, which can cause a set of parallel processes to hang. Apart from tracking down where the system has stopped, it then became vital to be able to differentiate between which process caused the stoppage and which were hanging because of that failed process. Obviously, such debugging facilities require direct access to the transputer hardware and a comprehensive symbolic map of the entire parallel system. From these it is then possible to build up a picture of each processor in the system and assess at what point in the code it has stopped. When this project commenced, the OCCAM development software environment needed to interface with other languages (i.e. FORTRAN in this case) had *no* debugging facility whatsoever, as this was still under development and not available for general release. This omission forced the use of very primitive incremental programming techniques in order to establish what processes were causing a particular hang up in the system. Although debugging facilities have now been included in the latest release of the OCCAM Toolset™, their nonexistence during the re-development of the SUPREM II software drastically elongated the project timescale.

The afore-mentioned constituted the main issues pertinent to the timespan of the software re-development. The other area which requires comment is that associated with further improving the performance of the concurrent system. The first section of the software which would benefit from further investigation is that of the parallel matrix solution algorithm. The direct block-based method implemented to date is less favoured in the literature than the family of iterative solution techniques based around the conjugate gradient method[12]. However, it was considered that it would be more worthwhile to pursue development of other applications of parallel techniques to process simulation rather than re-implement SUPREM II using these alternative algorithms. Commercially, SUPREM II had been overtaken by the introduction of SUPREM III and IV which provide much greater process modelling capabilities. Therefore any further development of the SUPREM II code would have primarily been an academic exercise and access to commercial versions of SUPREM III and IV was not possible at this time, which ruled out the possibility of continuing with these updated codes.

In producing the parallel version of SUPREM II, the largest problem was that surrounding the data transfers necessary to provide all the processors with a complete set of local and global information. This arises because the transputer architecture is based on local memory alone and relies on effective communication policies to ensure that all the processors contain valid data. Therefore, another area for future research would be an investigation into the benefits of using a shared memory multiprocessing architecture, such as the BBN Butterfly[13], to provide a suitable parallel computing surface for data intensive applications. Whichever architecture is chosen for the production of a parallel version of software similar to SUPREM II in terms of data and computational requirements, it can be seen that the task is very expensive in terms of the application specific development time. For many applications, which are only ever run on a limited number of datasets at a time, this is the only way in which parallel computing techniques can provide additional performance. However, if it can be foreseen that an application would be exercised on a large number of datasets, then an alternative parallel strategy may be employed. This is commonly referred to as the *task farm* approach. In brief

though, it involves running a number of *independent* jobs in parallel on the relevant computing hardware, with each job executing serially on an individual processor. In this fashion, assuming that there are a large number of jobs to be performed compared to the number of processors available, then a linear performance increase equal to the number of processors employed can be expected. The only overhead associated with this approach, is that involving a single master processor which is responsible for allocating jobs to each worker processor and collating the returned data. The application and implementation of a task farm will be discussed in detail in chapter 7.

References

1. B.E. Deal, A.S. Grove, "General Relationship for the Thermal Oxidation of Silicon", J. Appl. Phys., Vol.36, 1965, pp3770-3778.

2. C.H. Romine, J.M. Ortega, "Parallel Solution of Triangular Systems of Equations", Parallel Computing, Vol.6, 1988, pp109-114.

3. A.H. Sameh, D.J. Duck, "Parallel Direct Linear System Solvers - A Survey", Math. and Computation in Simulation, Vol.19, 1977, pp272-277.

4. G.A. Geist, M.T. Heath, "Matrix Factorisation on a Hypercube Processor", Proc. Hpercube Processors, SIAM Philadelphia, 1986, pp161-180.

5. H.H. Wang, "A Parallel Method for Tridiagonal Equations", ACM Trans. on Math. Software, Vol.7, No.2, 1981, pp170-183.

6. U, Meier, "A Parallel Partition Method for Solving Banded Systems of Linear Equations", Parallel Computing, Vol.2, 1985, pp33-43.

7. W. J. C. Alexander, A. J. Walton, J. M. Robertson and R. J. Holwill, "The Implementation of Process Simulation on Transputers for the Production of ASIC's", Microelectronics '89, Brisbane, Australia, July 1989.

8. "TITCH User Guide", Edinburgh Concurrent Supercomputer Project, ECSP-UG-2, 1988.

9. "TINY User Guide", Edinburgh Concurrent Supercomputer Project, ECSP-UG-9, 1989.

10. "Occam 2 Reference Manual", Inmos Ltd, Prentice Hall, 1988.

11. D. Pountain, D. May, "A Tutorial Introduction to OCCAM Programming", Inmos Limited.

12. D. O'Leary, "Parallel Implementation of the Block Conjugate Gradient Algorithm", Parallel Computing, Vol. 5, 1987, pp127-139.

13. A. Trew and G. Wilson, "Past, Present, Parallel", Springer-Verlag, 1991, pp64-75.

If enough data is collected, anything may be proven by statistics.

Chapter 6

Process Optimisation

6.1 Introduction - Design for Manufacturability

Throughout the evolution of the IC manufacturing industry the market has gradually altered from being supply-driven to demand-driven, largely as a result of the relatively small number of major companies involved globally, each attempting to retain or acquire a leading market position. Consequently, an extremely competitive edge to both the production and marketing philosophies has evolved, in terms of both price and quality. From a manufacturing point of view, this can be seen in the ever-increasing importance that has been placed on yield and turn-round times in order to increase the viability of a particular process design.

Typically, processes are presently designed round a set of nominal parameter values which have been found, either by experiment, simulation or a combination of both, to lead to the desired device specifications. The methodology commonly used to obtain these nominal values has already been discussed in chapter 3. However, during any actual fabrication cycle, variations in these process parameter values will be experienced due to for example, fluctuations in the operating conditions within the processing equipment. These variations obviously translate into variations in the resulting device parameters and therefore require to be investigated to ensure that the spread across the final product remains within the pre-defined, acceptable range of the process specification. In short this places a large responsibility upon the quality assurance aspects of the production facility and may result in an unnecessarily low yield because even functional devices may still fail to meet the actual product specifications. By briefly viewing the progress and changes in quality assurance policies within the IC manufacturing industry, possible means to improve this state of affairs become apparent.

The first attempts at quality assurance in the manufacturing process simply involved a post-fabrication inspection, with any faulty or under-spec samples being discarded. A slight improvement on this was made by bringing the point of

inspection forward, with the introduction of in-line testing. Now at any test point there was the option of scrapping poor samples or re-working marginal cases, which could at least reduce the volume of discarded products. However, rejection of product wafers is a costly procedure and such principles do not enhance the built-in production quality, but merely ensure that those samples available for the market meet their specification.

The next stage associated with quality assurance was to alter the emphasis from product inspection to process control, with the idea of minimising sources of manufacturing variations. This was done by assessing each process step in isolation and where necessary, attempting to reduce any variability by perhaps redesigning the step or purchasing new equipment with superior uniformity and control characteristics. This approach has many disadvantages associated with it. The task of actually pin-pointing the sources of variability is generally non-trivial, and once found, it may be difficult or impossible to fix by way of process re-design. The capital cost associated with replacing fabrication hardware is also extremely high. Lastly, it can be a time consuming activity, taking months or years to slowly tune a process for satisfactory yield, in which time large sums of money will have been lost.

Hence, there is a definite need for designing a *manufacturable* process in order that specifications may be met first time with acceptable yield levels. This is essential when considering the ASIC market, where low-volume, one-off production runs are commonly found and would be financially beneficial for all IC manufacturing environments, especially given the present economic climate within the semiconductor industry. This leads to the introduction of the term *design for manufacturability*, that is, designing a product in order to make it as insensitive to manufacturing variations as is possible. In complete contrast to all the previously mentioned techniques, this is performed during the design phase rather than during manufacture. By taking such an approach, quality is actually being built into the product and any possible production problems can hopefully be highlighted and ironed out before the first silicon is committed to fabrication. Assuming the goal of

minimising the design sensitivity to processing variations is met, then financial gains are obtained from the reduction in lost yield, time spent tuning the process and on expensive fabrication equipment.

Therefore, it is necessary to assess the response of any device characteristics which are deemed critical for a given situation, in terms of the respective input parameters. Once this characterisation in terms of the process parameters has been achieved, then their values may be optimised to the point where least sensitivity in the required device specifications is achieved. Now, small fluctuations in these parameter values throughout the fabrication process should not cause a significant loss of yield in the final product.

Conceptually, this appears a sound policy to follow, but it does require that the relationship between the device responses and the processing parameters used in fabrication must be determined. There are two possible routes to provide this information. Firstly by conducting a series of experimental fabrication runs under a variety of process conditions which reflect any likely variations found under normal fabrication conditions. This is an extremely costly and time consuming approach with each process run taking around four weeks to perform and costing many tens of thousands of dollars in terms of material and labour alone. Secondly, the same exhaustive set of experiments could be performed using some of the process simulators discussed previously. Although much quicker and cheaper than the experimental procedure, this still consumes many weeks of CPU time on typical computing platforms. As a result of this, the use of statistical experimental design techniques has been proposed in order to produce a suitable set of experimental/simulation points and perform the subsequent analysis of the results.

This chapter first introduces the ideas associated with experimental design techniques. By briefly discussing some of the classical approaches to defining the required experiments and three of the main design objectives used to analyse the data, the extremely powerful nature of this technique is demonstrated. Next an overview of one particular commercially available software package, RS/1™ (BBN Software Products Inc), is presented. This package provides a complete software

shell under which statistical experiments may be designed and subsequently analysed. The application of these techniques to IC process design and optimisation is then discussed and presented through using a simple case study. Finally, a few conclusions as to the implementation of these tools are presented. This provides a background to the experimental work described in the next chapter.

6.2 Experimental Design

Experimental design is a strategy for the effective management of experimentation. It is based around the use of a *planned* program of experiments which employ statistical techniques for their design and analysis [1],[2]. As a consequence, much more useful data can be extracted from a smaller number of runs than would normally be feasible if an unstructured approach were to be used. Traditional experimental techniques tended to employ unsystematic variations on the input parameter set, which were based on previous results. Such an approach can lead to conflicting results, especially when many factors are being simultaneously observed. Additionally, it is all too easy for many implicit assumptions to be subconsciously inferred throughout the procedure. In contrast, experimental design starts with a few basic informed assumptions and provides a complete set of experimental runs in advance, which will cover the entire range of interest [3]. This ensures that all possible conditions are investigated and maximises the chance of finding the *best* operating region for a particular set of goals.

Experimental design techniques provide many benefits which assist an engineer to reduce the product design time and cost, so reducing the overall lifetime costs of that particular process. The most apparent of these is that of enabling quality and robustness to be built into the process from the initial design stage. In achieving these goals, any factors which may prove important with respect to process control issues will be automatically highlighted and can therefore be given careful examination during the fabrication cycle. Additionally, the relative effects of variations in any of the processing parameters is known prior to fabrication and hence emphasis can be placed on the controlability and uniformity of the equipment pertinent to these factors.

There are six identifiable stages involved in an experimental design problem and these can be summarised as follows:

- Problem definition. The factors to be varied and responses to be observed must be established. In addition, the objective of the experimental analysis must be determined (see §6.2.1).
- Model specification. The type of model describes the general form of the equation (i.e. linear, quadratic, cubic etc.) which will be fitted to the experimental results during their analysis. The model type will affect the number of runs required and hence has to be determined in advance of the experimental work.
- Design Specification. Either a classical or computer-generated design must be selected in order to establish the actual experiments which are required to be performed (see §6.2.2). The choice is dependant on the experiment objective previously specified and will be one of the following :-

  Plackett-Burman

  Full Factorial

  Fractional Factorial

  Box-Behnken

  Box-Wilson
- Experimental work. The necessary experiments may be carried out in either a standard/structured or random order.

  Standard - Run order is organised by variation in factors. This approach produces a runsheet which is simpler to follow.

  Random - By randomising the run order, possible bias effects can be eliminated from the results and it also reduces the effects of factors which are not being explicitly considered in the experiment

- Analysis of the results. The model(s) must be fitted to the experimental data either directly or by the use of regressional techniques. Usually an iterative refinement process is then applied in order to confirm the model validity and quality-of-fit. On this basis, a requirement for additional model coefficients or the inclusion of extra factors may be highlighted.
- Interpretation of the results. Response surfaces and statistical results may be calculated and displayed graphically. Response values for factor settings within the operating region, but not experimentally determined, may then be predicted using the fitted models. This permits the optimum factor settings for a particular experimental goal to be established.

Although each of these stages can be considered individually whilst illustrating their role within the overall methodology, their functionality is only realised once they have been integrated into one complete system. This requires that a solid set of data structures and interfaces must be developed to fully support the operations involved. The following two sections provide a brief introduction to concepts associated with the design objectives and templates.

6.2.1 Design Objectives

There are three main design objectives which may be called upon to analyse the data from a particular set of experiments. These are Screening, Response Surface Methodology[4] and Taguchi[5][6]. Each objective enables a different set of information to be efficiently extracted from the experimental results. Consequently, the selection of design objective influences the choice of the template (see §6.2.2) used to allocate the experimental points within the region of interest.

Screening is specifically intended for the investigation of a large number of factors (e.g. more than 10), in order to find the most significant factors within a minimum number of runs. A Plackett-Burman design is normally used to produce the experimental points for this case and typically requires only one, two or three more runs than the number of factors to be screened [7]. Naturally, such a reduction

cannot be achieved without trade-off's. In this case the penalty for minimising the number of runs is an inability to estimate any possible interactions between factors. However, once a suitably reduced set of significant factors has been obtained, they can be more comprehensively investigated using one of the following alternative techniques.

Response Surface Methodology (RSM), is a technique for providing a more detailed analysis of a limited number of factors over their respective operating ranges. Interactions between factors can be assessed and a series of linear or polynomial relationships linking the responses to the factors can be determined. The model relationships are then able to predict responses at any point within the operating region. By combining the individual responses into one composite response, an optimised set of factor values may be found which meet a particular design goal. This technique provides a systematic approach to a multi-factor/multi-response optimisation problem. Consequently all the complex trade-offs which were traditionally made in a designers head or via a vast number of 1-D plots, can now be considered in a structured manner and presented in one or two contour or 3-D surface plots.

The Taguchi technique, like RSM, provides a series of relationships between the factors and responses concerned. As before this enables response values to be predicted for factor settings which lie within the chosen operating region but have not been explicitly determined by experimentation. In addition, the method permits the inclusion of small perturbations on each factor value to investigate explicitly any problems that may arise from poorly controlled or non-uniform factors.

Both RSM and Taguchi require a reasonable number of runs to be performed in order that the direct and interactive relationships between factors and responses may be determined. On this basis, it is not advisable to consider more than six or seven factors using these techniques since the number of runs will rapidly become excessive. Hence an initial screening experiment is commonly used to produce a suitable set of significant factors from a much wider range of factors associated with the process concerned.

6.2.2 Design Templates

There are a number of *classical* experimental designs, all of which have fixed properties such as the number of runs, number and type of factors and the appropriate models. For situations where this becomes too restrictive, there is also a more flexible, but also computationally intensive, *D-Optimal* design. A brief summary of each design type is presented in the following paragraphs (further details may be found in [8]).

The simplest design is the Plackett-Burman design which is most commonly used for screening objectives. It results in the minimum number of runs to assess the importance of each factor. As shown in table 6.1, the total number of runs is of the same magnitude as the factors, even for a large factor count. Once more than ten factors are included, the choice of experimental points assumes a significant similarity to that attained using a fractional factorial design and the number of runs rises slightly.

Number of Factors	Minimum number of runs
7	8
9	12
15	16
17	20
23	24
27	28
29	32

Table 6.1. Plackett-Burman Designs

The second category of design comes under the heading of *factorial* designs[9], of which there are two classes, *full* and *fractional*. These permit both the main effects and interactions between several factors to be investigated at once. Using two levels for each factor, then a full factorial design is appropriate for up to five factors and results in 2^k runs for k factors. This generalises to m^k runs for k factors each at m levels. In other words, all possible combinations of factor settings are incorporated into the design. Geometrically, a full factorial design may be represented by the diagram in figure 6.1.

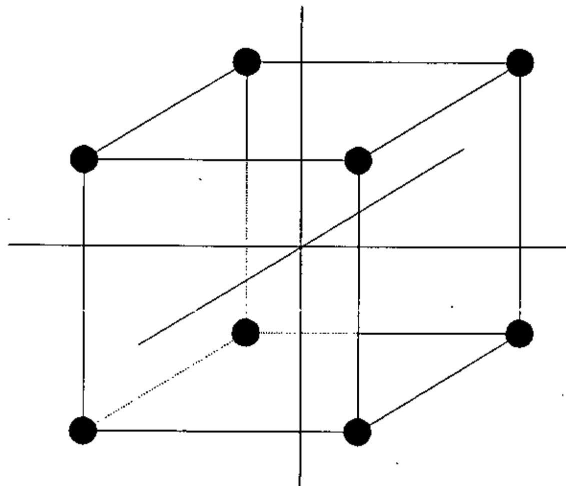


Figure 6.1. Geometric Representation of a Full Factorial Design.

Hence when considering six to eight factors, a fractional factorial design may provide a more convenient way to reduce the number of runs from that required by the full factorial design. The price to be paid for this reduction is the loss of capability to measure some of the inter-parameter interactions. Figure 6.2 shows the geometric representation of a fractional factorial design.

As was mentioned earlier, once the original factor count exceeds eight, then it is probably worthwhile performing an initial screening exercise to find which of these are of greatest interest to the problem under investigation. Neither of the factorial designs include *centre* points (i.e. points located about (0,0) on the axes shown above). This means that mathematical techniques cannot be used to assess

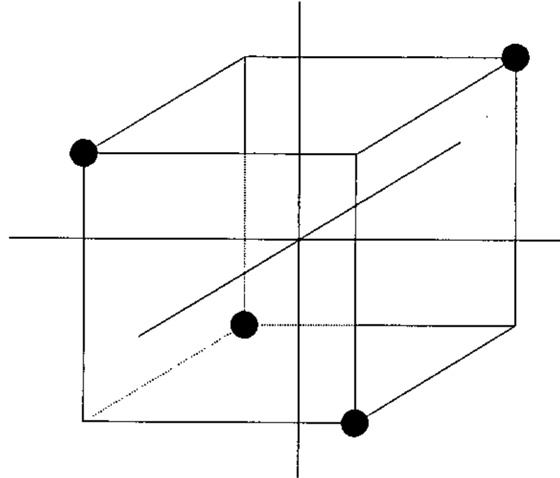
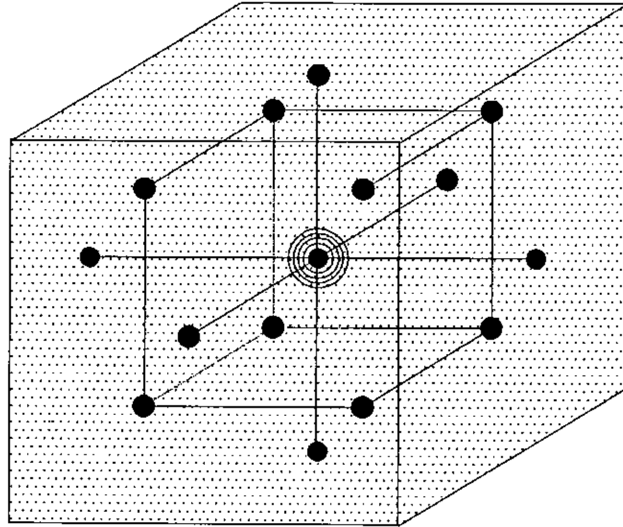


Figure 6.2. Geometric Representation of a Fractional Factorial Design.

the quality of fit of the models to the data, nor can any estimate for the curvature of the response surface be obtained. Hence the factorial designs are termed *limited response surface* designs.

The remaining two design types are termed *full response surface* designs in that they include three level factoring plus centre points. The first of these, called the Box-Wilson design, is a central composite design[10]. The basis of this design, as can be seen in figures 6.3 and 6.4, is a central box of points formed from a factorial scheme which is then augmented by a series of axial outlying points. Depending on the span of these axial points, two different designs can be achieved, namely Central Composite Inscribed (CCI) or Central Composite Circumscribed (CCC). If *all* the points lie within the factor ranges, then a CCI design is produced. If, on the other hand, the axial points lie outside the factor range, then a CCC design is produced which enables the optimisation and prediction of responses over the entire factor range.



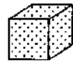
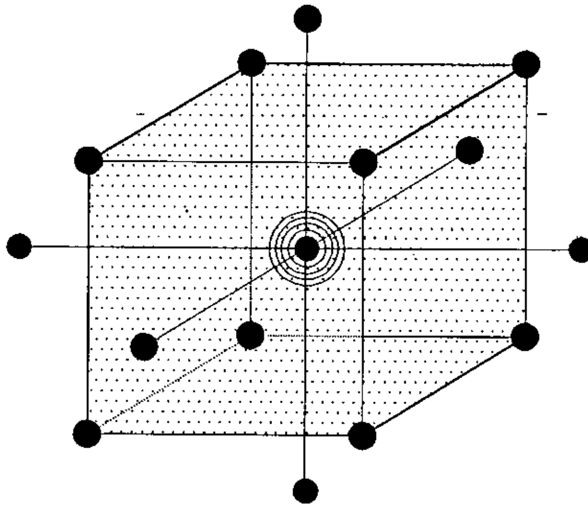


Figure 6.3. Geometric Representation of a Box-Wilson CCI Design.



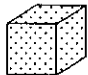


Figure 6.4. Geometric Representation of a Box-Wilson CCC Design.

The second of the full response surface design is the Box-Behnken design[11]. This provides precise, quality predictions over operating regions where linear, interactive and curvature effects are expected. Three level factor settings are used resulting in a series of replicated centre points. Figure 6.5 shows the geometric representation of a Box-Behnken design, in which the points can be seen to lie on the mid-points of each edge of a multidimensional cube that defines the operating region.

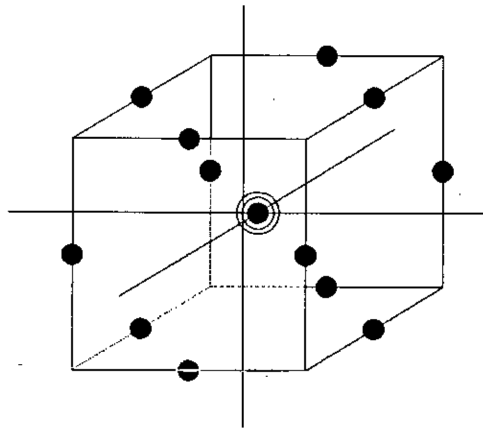


Figure 6.5. Geometric Representation of a Box Behnken Design.

6.3 RS/1

In the preceding sections of this chapter, the powerful nature of experimental design has been indicated and a brief resumè of both the theory and practise of applying the technique has been presented. However, in order to support efficient use of such methods, a comprehensive set of data management, statistics, modelling and graphics utilities are required. These facilities can be made available through a multitude of existing software packages, though this frequently results in the usual interfacing problems found under such circumstances. Alternatively, a complete software system, offering all of these facilities under an interactive menu-driven environment, is available from BBN Software Corporation. The 'RS' series of software is produced as a set of three fully integrated subsystems, RS/1™,

RS/Discover™ and RS/Explore™, all of which are presently available for both VAX/VMS™ and UNIX™ operating systems. In addition, a version of RS/1™ is available for operation on PC compatible computers running MS-DOS™.

RS/1 provides the main data structures and management modules, which also includes comprehensive statistical data analysis routines. RS/Discover provides the main experimental design functionality, covering the experiment definition, data analysis and data visualisation aspects. RS/Explore provides additional statistical capabilities to enhance the depth of information available from the basic RS/1 data structures and RS/Discover experiments. A general user interface to the host filing system allows data to be imported from or exported to other software applications. In common with most commercial database and spreadsheet systems, RS/1 has its own powerful programming language, called RPL, which enables many application specific operations to be conveniently automated. RPL's major limitation is that its capabilities do not extend to the control of RS/Discover, which requires all operations to be performed via the menu-driven interface. Various aspects of the 'RS' software directly relevant to this work will be discussed in the course of §6.4, but fully comprehensive details of the software are available in the appropriate manuals [12], [13], [14].

6.4 Optimised Process Design

In the manufacture of VLSI circuits, process induced variations of device characteristics can drastically reduce the performance of fabricated products. It is therefore most desirable from yield, time and economic considerations, that process conditions are optimised so as to minimise the device sensitivity to these variations. Strategies, mainly based on engineering judgement (guess-work!) do exist for producing devices which work at a nominal operating point. However, few strategies have been commercially implemented which go to the extent of analysing the operating point and its sensitivity to perturbations in the processing conditions. Thus many processes exist which are probably not as robust or viable as they could be, a fact which is hardly altogether surprising given the complex nature of the effects and interactions between the magnitude of process variables.

The experimental design procedures discussed previously in this chapter, lend themselves ideally to the cause of optimising a process towards a set of performance and sensitivity goals. Under most conditions, these goals, as set by the process and device engineers, are known in advance and hence a list of probable factors and responses requiring investigation can be drawn up. An initial consideration of the process parameters will most likely result in a substantial number of variables which could influence the characteristics of the fabricated devices. Therefore, as mentioned previously, it is advisable to perform a preliminary screening exercise to establish the five or six most significant parameters relevant to the application. Thereafter, a full response surface design could be produced, again using RS/Discover, to cover the chosen factors and responses in greater detail.

The use of RSM techniques to provide an efficient, structured design and analysis strategy has been well-known for several decades. However, its application to the semiconductor industry is much more recent and can be viewed in a novel manner by applying the methods to simulated results rather than genuine experimental results[15]. Although the use of experimental fabrication runs to provide the data is perfectly valid, it is an extremely expensive and slow approach by comparison. Thus the results are obtained from the combined use of process simulators such as SUPREM II, SUPREM III, SUPRA and device simulators like PISCES and MINIMOS as discussed in chapter 3. Obviously, it is important that the simulators have been well characterised for the particular process before the absolute values of their results can be confidently relied upon. The simulation results are then entered into the RS\Discover database where upon a multi-variable polynomial is fitted to each response to permit subsequent analysis. Each response can then be displayed through contour or 3-D plots obtained using the fitted models, in terms of any of the relevant process parameters. Such visualisation of the individual responses allows each one to be assessed in terms of optimum performance, however, this is more than likely to produce conflicting options which will require to be resolved. A flow diagram for the basic steps involved in this procedure is shown in figure 6.6.

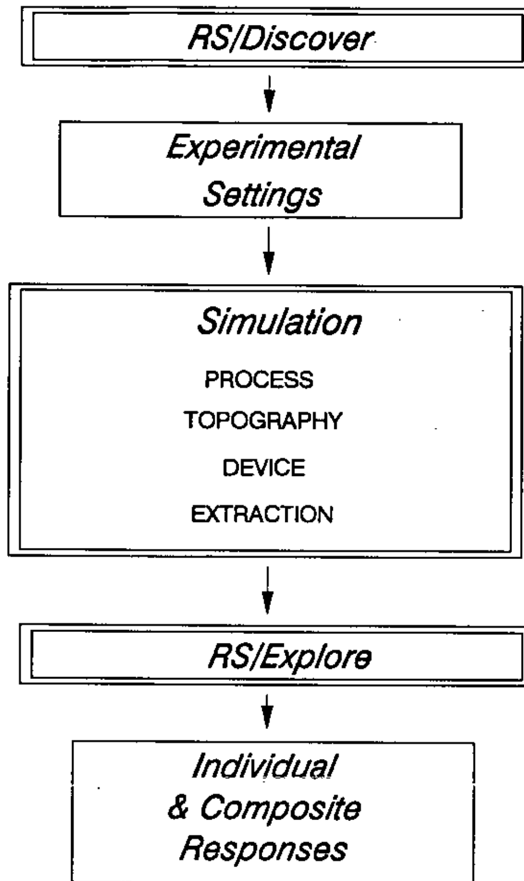


Figure 6.6. Basic flow diagram for simulation based optimisation.

The strength of RSM is that it enables *composite* responses to be produced which comprise a combination of the individual responses already produced. These composite responses can then be minimised or maximised as necessary using both visual and mathematical techniques, thus resolving any of the conflicts observed in the individual analysis previously undertaken. The optimal solution to the given constraints is then provided by the then relevant maxima or minima. In addition, it is very simple to incorporate further qualifications in order to guarantee a safe operating region. The final item which is easily performed with RSM is that of sensitivity analysis. By including terms for the gradient of the composite response and variances of the process concerned in the analysis, the response sensitivity to input parameter variations can be incorporated in the final assessment for determining a safe operating region.

Although these experimental design techniques substantially reduce the number of simulations required for a given problem, there are still a large number of repetitive file and system operations involved. Whilst these may all be performed manually, this is wasteful in terms of time and also provides much scope for introducing errors. Therefore, a more realistic flow diagram for an integrated optimisation system is shown in figure 6.7 and illustrates the additional features which have been included to automate the interface between RS/1 and the relevant simulators. The early development of this additional functionality is discussed in §6.5 by way of a worked example, however a more detailed discussion of these facilities may be found in [16] and [17].

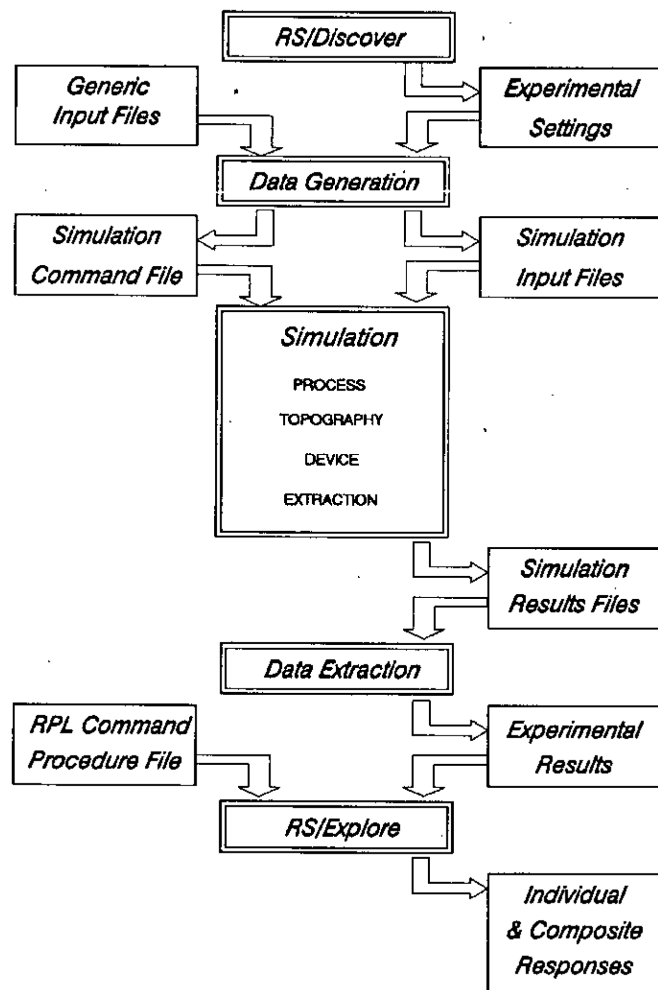


Figure 6.7. Complete flow diagram for simulation based optimisation.

The important point to be appreciated from the concept of experimental design is that the entire process can be thoroughly analysed *before* fabrication is commenced. In this manner most of the problems associated with production can be investigated and eliminated during design, which once frozen, should provide a robust, high-yielding fabrication schedule.

6.5 Optimisation Study

The procedures associated with a complete optimisation schedule, as illustrated in figure 6.7 above, may be divided into five distinct sections, of which only three (1, 3 and 5) are directly involved with the optimisation itself. The remaining two (2 and 4) have been included to automate the interface between RS/1 and the relevant simulation software with the intention of improving the time taken to handle all of the necessary simulation files. The procedures also reduce the scope for error whilst manipulating all of the appropriate simulation input and output data, the results of which would necessarily be obvious in the subsequent analysis.

1.	Experimental Design	RS/1
2.	Simulation input file generation	RPL and Fortran/C
3.	Simulation	
4.	Data Extraction from simulation output	RPL and Fortran/C
5.	Results Analysis	RS/1

A discussion of each of these steps is best provided by following through a typical optimisation problem and thus illustrating their role in the overall procedure. The problem to be analysed during this case study involves the simulation of a cMOS IC fabrication process. The factors to be varied and responses to be observed are listed below:

Factors	Implant Energy
	Implant Dose
	Gate polysilicon slope

Responses Series resistance
Peak electric field
Transconductance
Punchthrough voltage

With these parameters in mind, the first task is to produce the required experimental design using RS/DISCOVER. This requires that the range of values for all the factors and responses are nominated. Then when the particular design specification is chosen, a table will be produced which lists all the appropriate experimental points. Figure 6.8 shows the summary screen provided once the design process has been completed and highlights the significant design features and experimental status.

```
Summary of THESIS:  Process Optimisation Example          1 FACTORS
(Created 09-AUG-92 16:05:48)                               2 SPECIFICATIONS
                                                           3 RESPONSES
Controlled factors:   SLOPE, DOSE, ENERGY                  4 PERF_STAT
Responses:    PUNCHTHROUGH_VOLTAGE, PEAK_FIELD, ...        5 MODEL
Formulas:     COMPOSITE_RESPONSE                           6 DESIGN
                                                           7 WORKSHEET
Objective:    RSM                                          8 AUTOGEN
Design Type:  FULL-FACTORIAL                               9 NEXT
Model Type:   QUADRATIC

MODEL.......  10 terms
DESIGN......  27 runs
WORKSHEET...  27 total runs (0 completed)




Generating expanded terms.
Generating FULL-FACTORIAL design.
Generating standard ordered worksheet.
EXPER.DEFINE>
```

Figure 6.8. Summary of Experimental Design details

RS/1 keeps much of its information stored in the form of tables, some of which are directly accessed by the user, some which can be indirectly accessed and others which are for its own use only. The new design is stored in the *WORKSHEET* table and this has sufficient space allocated to store all of the assigned factors and the responses, which are as yet unknown. Figure 6.9 illustrates part of this table, showing the design data and the space for the results to placed in due course.

0	1 SLOPE (DEG)	2 DOSE (ATOMS/CM^3)	3 ENERGY (KeV)	4 PUNCHTHROUGH_VOLTAGE (VOLTS)
1	70	1e+13	30	
2	90	1e+13	30	
3	110	1e+13	30	
4	70	5e+14	30	
5	90	5e+14	30	
6	110	5e+14	30	
7	70	1e+15	30	
8	90	1e+15	30	
9	110	1e+15	30	
10	70	1e+13	60	
11	90	1e+13	60	
12	110	1e+13	60	
13	70	5e+14	60	
14	90	5e+14	60	
15	110	5e+14	60	
16	70	1e+15	60	
17	90	1e+15	60	

Enter /HELP for command explanations.
Row 1 Column 3: [30]

Figure 6.9. Part-filled experiment worksheet table

Within this worksheet, the data in one line contains the values for each factor for one particular simulation. Although this could be manually transferred into an appropriate simulation text file, the task lends itself to automation. Unfortunately, RS/1 stores all of its table data in a machine readable form, so it can not simply be accessed directly and it must first be exported to the host filing system as an ASCII file, as shown in figure 6.10.

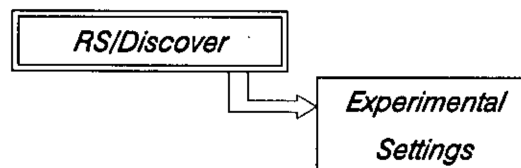


Figure 6.10. Flow diagram for exporting RS/1 design data

As mentioned previously, RS/1 contains a very powerful programming language, RPL, which enables the user to write mini-programs, called procedures, that can perform most RS/1 operations. Thus a short procedure was written, which

exports the design data to an ASCII file after performing a number of checks on both the data and the file system. An excerpt of this procedure is shown in figure 6.11 and the resulting text file is displayed in figure 6.12.

```
/* This procedure allows the user to create a system file to dump the input$
/* ,  the file created can then be used in generate                        $

Procedure;
         temptable = 'oops'; /* name of temporary table to create */
         expname = getobject('Enter experiment name: ', 'experiment');
         if obj$exists(expname,'experiment') then

         /* case when experiment exists */

         begin;
              designtbl = cat( expname,'@worksheet');
              if tableexists(designtbl)then

              /* case when worksheet exists */

              begin;
                   factortbl = cat(expname,'@factors');
                   if tableexists(factortbl) then

                   /* case when factors table exists */

== MicroEMACS 3.9e (WRAP EXACT ASAVE) == edit.tmp == File: /u3/wjca/rsuserhome/$
```

Figure 6.11. Extract from the "DUMPTABLE" RPL Procedure

```
70¦1e+13¦30
90¦1e+13¦30
110¦1e+13¦30
70¦5.05e+14¦30
90¦5.05e+14¦30
110¦5.05e+14¦30
70¦1e+15¦30
90¦1e+15¦30
110¦1e+15¦30
70¦1e+13¦60
90¦1e+13¦60
110¦1e+13¦60
70¦5.05e+14¦60
90¦5.05e+14¦60
110¦5.05e+14¦60
70¦1e+15¦60
90¦1e+15¦60
110¦1e+15¦60
70¦1e+13¦90
90¦1e+13¦90
110¦1e+13¦90
70¦5.05e+14¦90
90¦5.05e+14¦90
--More--(85%)
```

Figure 6.12. ASCII design data file

The next task, as depicted in figure 6.13, is to produce an input file for the appropriate simulator from the data in each line of this ASCII file and also produce a batch command file to actually perform all the simulations. Originally, this was achieved using a program written in FORTRAN 77, a language chosen not for its lexical capabilities, but because of constraints applied by the VMS operating system in use at the time. The software was subsequently re-written in 'C' and endowed with greater functionality and robustness on returning to a UNIX platform towards the end of the project.

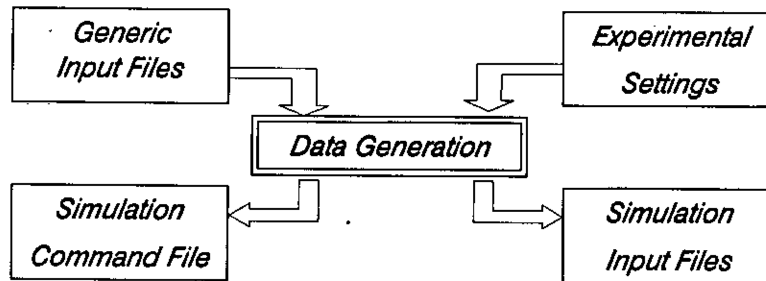


Figure 6.13. Flow diagram for generation of simulation files

This process requires the following information in order to produce the necessary simulation files for the correct simulation package: a *generic* simulation input file, highlighting the factors to be varied; the design data itself; general information pertaining to file name conventions, the simulator to be used, data directories and other details which will be discussed later. The generic simulation file contains a full description of the process to be modelled with the exception that the parameters under observation are replaced by a marker symbol, "?*n*", as can be seen in the file sample shown in figure 6.14 (?1 and ?2). These markers are then replaced by the necessary experimental values taken from the ASCII design file.

The generalised procedure data file contains all the information required to control the entire 'generation' sequence and as will be seen later, the subsequent 'extraction' process. It is this file which provides much of the flexibility offered by

```
TITLE           THESIS Process Optimisation Example

$               Initialise the silicon substrate.
INITIALIZE      <100> SILICON   PHOSPHOR=10 RESISTIV
+               THICKNESS=10.0  XDX=0.5 DX=0.002  SPACES=800

.
.
.

$               GATE Oxide
DIFFUSION       TIME=5 TEMPERAT=850 T.FINAL=900 F.N2=0.0 F.02=21.1
DIFFUSION       TIME=5 TEMPERAT=900 F.N2=0.0 F.02=21.0

$               Polysilicon deposition 550A
DEPOSIT         POLYSILICON  TEMPERAT=600  THICKNESS=0.055 SPACES=10

$               Enhancement implant
MASK            APPLY NAME=ENHANCE
IMPLANT         BORON DOSE=?1 ENERGY=?2
MASK            REMOVE

-* MicroEMACS 3.9e (WRAP EXACT ASAVE) -- generic.dat -- File: generic.dat ------
```

Figure 6.14. Generic simulation input file

this system and enables it to be used for virtually any application under a number of host environments. A segment of this data file pertaining to the generation process is shown in figure 6.15.

```
##################### BEGIN OF CONTROL FILE #####################################$
###### Title of experiment: #####################################################$
title Thesis Process Optimisation Example
####### Simulator name:(one word only) ##########################################$
simulator suprem
###### Design file name:(ie file in which data to simulator is stored )#########$
design_name exdesign.dat
##### Generic file name: (ie file which simulator accepts and contain search_va$
generic_name rslgen.dat
####### Prefix for drive (simulation) file name: ###############################$
prefix run
####### Extension for drive (simulation) file name: ###########################$
extension .dat
####### No. of drive (simulation) files to be generated: ######################$
no_runfiles 27
####### No. of input factors being varied in experiment: ######################$
no_factors 3
####### Names of factors to be varied in experiment: (separated by spaces) #####$
factors dose energy slope
####### Delimiter separating data values in design file: (one character only)##$
delimiter |
####### Variable to search for in generic file: (one character only) ##########$
-- MicroEMACS 3.9e (WRAP EXACT ASAVE) -- control.dat -- File: control.dat ------
```

Figure 6.15. "Generate.dat" - Overall procedure details

Once the *GENERATE* process has been performed, a full complement of simulation input files, each containing a unique set of experimental settings, will exist, as can be seen in figure 6.16 where the values, "1e12" and "30" have replaced the marker symbols.

```
TITLE        THESIS Process Optimisation Example

$            Initialise the silicon substrate.
INITIALIZE   <100> SILICON  PHOSPHOR=10 RESISTIV
+            THICKNESS=10.0  XDX=0.5 DX=0.002  SPACES=800

.
.
.

$            GATE Oxide
DIFFUSION    TIME=5 TEMPERAT=850 T.FINAL=900 F.N2=0.0 F.02=21.1
DIFFUSION    TIME=5 TEMPERAT=900 F.N2=0.0 F.02=21.0

$            Polysilicon deposition 550A
DEPOSIT      POLYSILICON  TEMPERAT=600  THICKNESS=0.055 SPACES=10

$            Enhancement implant
MASK         APPLY NAME=ENHANCE
IMPLANT      BORON DOSE=1E12 ENERGY=30
MASK         REMOVE

=* MicroEMACS 3.9e (WRAP EXACT ASAVE) == run01.dat == File: run01.dat ==========
```

Figure 6.16. Completed simulation input file

In addition to producing all the simulation files, GENERATE also creates a batch file to perform all of the simulations and uniquely rename all of the output files for subsequent perusal. Figure 6.17 shows part of this batch file for the current example. The syntax for this batch file is of course system dependent and hence there is an entry in the general data file for the operating system in use.

All the necessary information has now been produced to enable the simulations to be performed. This is done by executing the batch file produced by the GENERATE software. The value gained by this approach is that the methodology can be applied to almost any repetitive or batch-oriented process since all the required details are contained with the data file "generate.dat". Once all of the simulations have been performed, the appropriate information must be extracted from each

```
echo START OF SIMULATION
cd .
suprem  ./run001.dat
mv ./s3out ./s3outrun001.dat
mv ./RESULTS ./RESULTSrun001.dat
rm optim
cp ./OPTIM ./optim
suprem  ./run002.dat
mv ./s3out ./s3outrun002.dat
mv ./RESULTS ./RESULTSrun002.dat
mv ./optim ./tempfile
cat ./tempfile ./OPTIM > ./optim
suprem  ./run003.dat
mv ./s3out ./s3outrun003.dat
mv ./RESULTS ./RESULTSrun003.dat
mv ./optim ./tempfile
cat ./tempfile ./OPTIM > ./optim
suprem  ./run004.dat
mv ./s3out ./s3outrun004.dat
mv ./RESULTS ./RESULTSrun004.dat
mv ./optim ./tempfile
cat ./tempfile ./OPTIM > ./optim
-- MicroEMACS 3.9e (WRAP EXACT ASAVE) -- runfile -- File: runfile -------------
```

Figure 6.17. Simulation batch command file

output file and returned to the RS/1 worksheet. As before, this involves two processes, the first of which is depicted by figure 6.18 and is concerned with extracting the chosen response information from the simulation output.

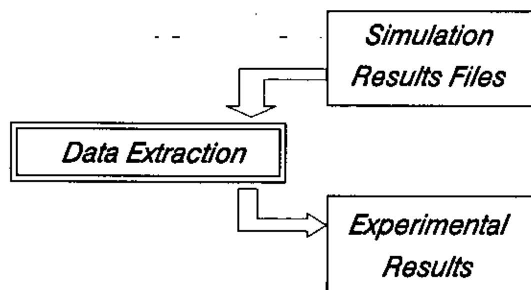


Figure 6.18. Flow diagram of data extraction process

Extracting the correct data from each simulation file is not a trivial process, since the location of a given piece of information may vary depending on how the simulation proceeded. Early versions of the extraction software used a similar method to that employed to generate the input files, namely a generic output file, with the chosen responses marked as before. Although this worked more often than

not, it was not a robust or flexible solution and was dropped in favour of the current lexical matching technique. This required the user to stipulate a character sequence pertaining to the required result, plus the number of words and/or lines between these characters and the actual response. Because many phrases, for example "Oxide Thickness", may be repeated a number of times within the output file, the user can specify which occurrence to search for. Other information such as which files to perform the extraction on and which file each extracted response is to be placed is all kept in the latter part of "generate.dat", as can be seen in figure 6.19.

```
final_commands echo STOP
####### No. of files to put extracted data ##################################$
no_extract_files 1
####### Names of files to put extracted data: (separate by spaces) ###########$
extract_names    rsl
####### No. of responses in experiment: ######################################$
no_responses 4
####### Names of responses:(separate by spaces) ##############################$
responses resistance pkfield gm vpt
####### Strings to search for to locate data: ################################$
search_strings  Resistance Electric Transconductance Punchthrough
####### No. of occurrence of each string: (count as one if string occurs more #$
####### than once in a line) #################################################$
occurrence 1 1 2 1
##### No. of lines at which data is ON after string occurrence has been found: $
data_line_no 1 0 2 4
##### position of data i.e. on which no. of word of the line is data situated: $
data_position 5 6 4 2
####### At which file should extracted data be written to : #################$
####### (ignore if no. of extact file is only one) ###########################$
extract_destination  rsl
############# END OF CONTROL FILE ############################################$
-* MicroEMACS 3.9e (WRAP EXACT ASAVE) -- control.dat -- File: control.dat ------
```

Figure 6.19. "Generate.dat" - extraction information.

A second program, again written in 'C', performs all the requested data extractions and places the results into the specified ASCII text files. The current version of this *EXTRACT* software informs the user of any unsuccessful extraction attempts and performs a great number of error checks to ensure robustness. The final task to be performed is that of importing all of the results back into the RS/1 experiment worksheet, as depicted by figure 6.20.

This process is performed by executing another RPL procedure, similar to the export one discussed earlier, which reads in the response values from the

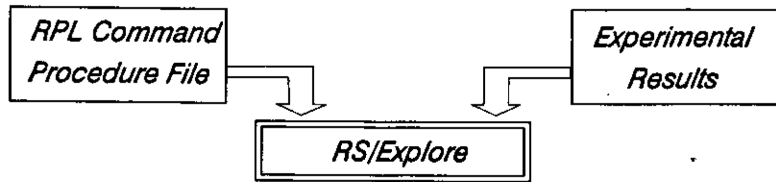


Figure 6.20. Flow diagram for importing results to RS/1

user-specified ASCII file directly into the appropriate columns within the experiment worksheet. Once this has been achieved, the completed worksheet table takes on the appearance of that shown in figure 6.21.

0	4 PUNCHTHROUGH_VOLTAGE (VOLTS)	5 PEAK_FIELD (KV)	6 TRANSCONDUCTANCE (Mhos)
1	19.80	280	5.20
2	17.80	340	6.70
3	14.80	560	8.80
4	19.70	310	6.20
5	16.70	420	7.70
6	12.70	640	9.80
7	16.30	370	8.00
8	14.30	480	9.50
9	11.30	770	11.60
10	17.90	260	6.20
11	14.90	320	7.70
12	11.90	530	9.80
13	16.60	290	7.20
14	13.60	400	8.70
15	10.60	620	10.80
16	14.20	360	8.30
17	11.20	470	9.80

Enter /HELP for command explanations.
Row 1 Column 5: [280]

Figure 6.21. Complete experiment worksheet table

Now that all the simulation data is residing within RS/1, it must be prepared for analysis by fitting a model to each one of the responses in turn and confirming that the fit is acceptable. RS/EXPLORE, the statistical analysis part of RS/1, can perform this task automatically, only pausing to inform the user when discrepancies are found. It is very important that this process is carried out if any confidence is

to be placed in any analysis of the simulation data. One erroneous data point is not necessarily obvious to the eye and yet can make an appreciable alteration to the model coefficients.

Once the models have been successfully fitted to all the responses, then *Response Surfaces* may be displayed as a contourplot for each response in terms of the various input factors. The two-dimensional nature of these contourplots means that there can be two variable factors along the x- and y-axes, and any other factors must be specified at a fixed value. Figures 6.22 and 6.23 show two particular response surfaces for the punchthrough voltage. This individual surface is plotted against the full range of implant dose and energy values, but at a fixed value for the polysilicon slope in each case.

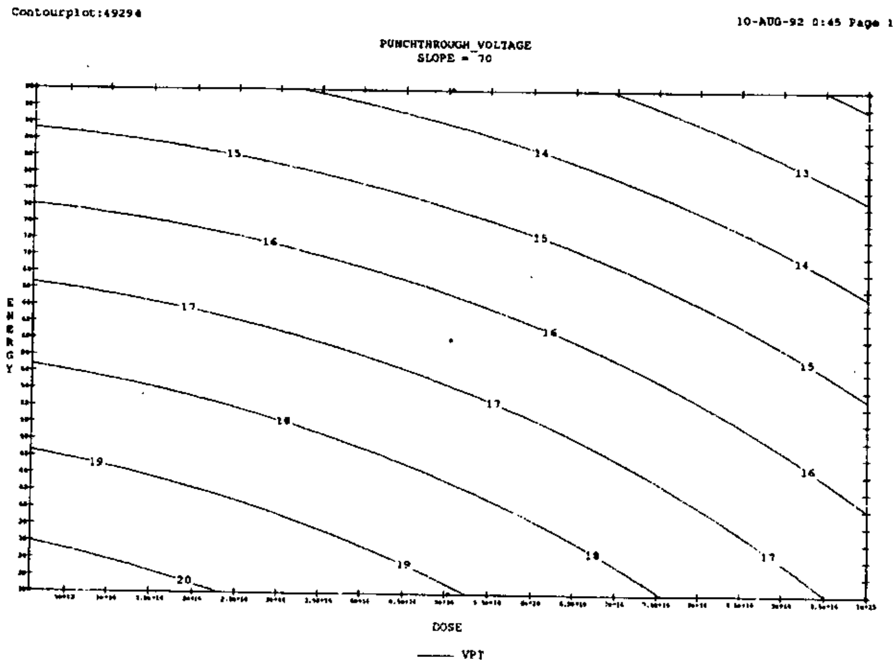


Figure 6.22. Punchthrough voltage response for a 70° slope.

It is then possible to interpolate the value for a particular response over any part of the simulation region using the model coefficients and so obtain response values for points where no simulation was actually performed. Although it is

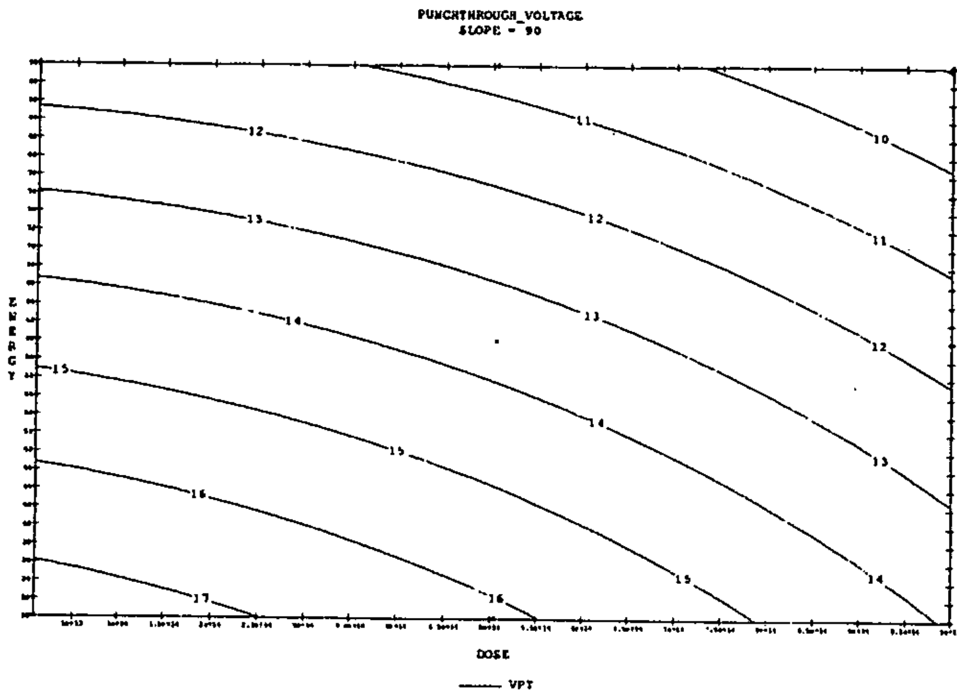


Figure 6.23. Punchthrough voltage response for a 90° slope.

therefore possible to produce a large number of these individual surfaces for each response, it is not necessarily inherently obvious how each factor may be traded off against another in order to arrive at a particular set of response conditions. This is of course the underlying concept of process optimisation and it is rarely possible to visually assess all of the parameter interactions. However, apart from displaying individual responses, it is also possible to produce a *Composite Response* which incorporates any number of responses and/or functions of responses, into a single surface. In this way, it becomes feasible to find a set of input conditions which will maximise or minimise each response as desired and hence arrive at a suitable operating region. For the optimisation example illustrated, it has been decided that optimal device performance will result when the following response conditions be met.

- Maximised punchthrough voltage
- Minimised peak electric field
- Maximised transconductance
- Minimised series resistance

The method employed in constructing a composite response is to produce an overall expression which can either be minimised or maximised. For this example it was decided to minimise the value for the composite response, and therefore it becomes necessary to convert any expression which needs to be maximised into a form requiring minimisation, i.e. minimise the inverse expression. Hence the approximate form taken by the composite response is shown in equation 6.1, where *a*, *b*, *c* and *d* are weighting factors.

$$CR = a \cdot \left(\frac{1}{V_{pt}}\right) + b \cdot \left(\frac{E_{pk}}{1}\right) + c \cdot \left(\frac{1}{Gm}\right) + d \cdot \left(\frac{R_s}{1}\right) \tag{6.1}$$

It is also advisable to normalise all the responses by their mean values, so that the issues of scaling or units of each quantity may be ignored. Equation 6.2 shows the final form for the composite response, where all factors have been equally weighted and then normalised about their mean values.

$$CR = \left(\frac{13.13}{V_{pt}}\right) + \left(\frac{E_{pk}}{447}\right) + \left(\frac{8.79}{Gm}\right) + \left(\frac{R_s}{675}\right) \tag{6.2}$$

Figure 6.24 illustrates the composite response which has been plotted against the implant dose and energy at the fixed 90° polysilicon slope value. From this plot, it can be seen that an operating point selected from the lower, right-hand region of the display will meet the conditions previously set.

This technique can then be further extended to assess the sensitivity of these responses to variations in each factor by displaying the appropriate response derivatives instead of the responses themselves. Hence the potential exists to locate an operating point for the fabrication process which meets a number of specified constraints and yet remain relatively undisturbed by small, but realistic, variations in the input conditions. It is important to remember that this analysis is only as accurate as the simulation data it has been provided with, but if the simulation

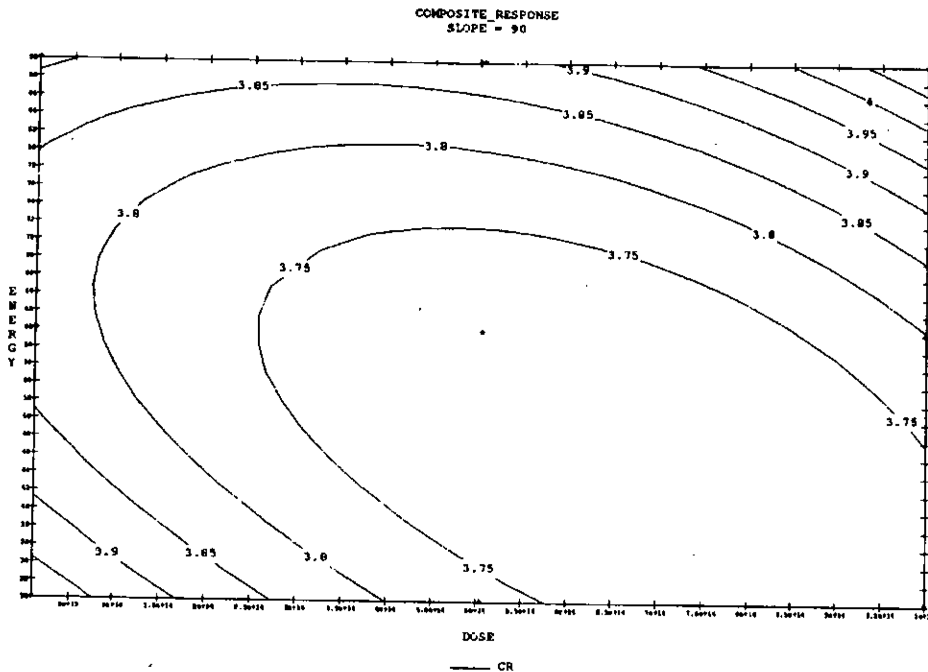


Figure 6.24. Contourplot of a composite response

packages have been fully calibrated then an acceptable level of confidence may be placed on any analysis undertaken and the concept of *design for manufacturability* has been fully implemented.

6.6 Summary

This chapter has introduced the concept of experimental design methods and a brief explanation of their underlying techniques has been presented. One particular software system, namely RS/1™, which implements experimental design techniques and manages the necessary databases is discussed and reviewed. The application of this simple, but powerful methodology, to present and future VLSI processing is then shown with the unusual approach of employing calibrated simulations to provide the results instead of experimental data. In this role, experimental design can not only ensure more robust process designs, but also enable them to be easily optimised against a set of performance criteria which might

pertain to a particular application. It is important to appreciate that the computational requirements are still excessive and that results are slow to acquire. However, if this hurdle can be overcome, then widespread use of this powerful strategy must be openly welcomed in commercial operations in order to ensure more economic and efficient fabrication processes. The next chapter will describe research carried out on a method of reducing the problem of computational costs in this area.

References

1. O. L. Davies, "The Design and Analysis of Industrial Experiments", Hafner Press, New York, 1954.

2. G. E. Box, W. G. Hunter and J. S. Hunter, "Statistics for Experimenters", Wiley, New York, 1978.

3. N. R. Draper ad W. G. Hunter, "Design of Experiments for Parameter Extraction in Multiresponse Situations", Biometrika, Vol. 53, 1966, pp525-533.

4. R. H. Myers, A. I. Khuri and W. H. Carter, "Response Surface Methodology: 1966-1988", Technometrics, Vol. 31, No. 2, 1989, pp137-157.

5. G. Taguchi, "Introduction to Quality Engineering", Asian Productivity Organisation, Tokyo, 1986.

6. E. C. Harrington, "The Desirability Function", Industrial Quality Control, Vol. 21, 1965, pp494-498.

7. R. L. Plackett and J. P. Burman, "The Design of Multifactorial Experiments", Technometrics, Vol. 25, 1983, pp279-284.

8. G. J. Hahn, discussion of "Experimental Design: Review and Comment" by D. M. Steinberg and W. G. Hunter, Technometrics, Vol. 26, 1984, pp110-115.

9. C. R. Hicks, "Fundamental Concepts in the Design of Experiments", Holt, Rinehart and Wilson, Montreal, 1964, pp75-92.

10. G. E. Box and N. R. Draper, "Empirical Model-building and Response Surfaces", Wiley, New York, 1987, p306.

11. G. E. Box and D. W. Behnken, "Some New Three Level Designs for the Study of Quantitative Variables", Technometrics, "Vol. 2, 1960, pp455-475.

12. "RS/1 Users Guide", BBN Software Products Corporation, Cambridge, MA, USA.

13. "RS/Discover Users Guide", BBN Software Products Corporation, Cambridge, MA, USA.

14. "RS/Explore Users Guide", BBN Software Products Corporation, Cambridge, MA, USA.

15. A. J. Strojwas, introduction to "Statistical Design of Integrated Circuits", Ed. A. J. Strojwas, IEEE Press, 1987, pp1-2.

16. G. J. Gaston, W. J. C. Alexander, L. J. Clarke, A. J. Walton, "PARTICS: A Parallel Taskfarm for Integrated Circuit Simulation", Edinburgh Parallel Computing Centre Technical Report, EPCC-TR91-08, 1991.

17. R. Ford, "The Application of Taguchi Methods to the Optimisation of IC Fabrication Processes", B.Eng Honours Project, HSP794, Department of Electrical Engineering, University of Edinburgh, 1991.

Never replicate a successful experiment.

Chapter 7
Parallel Implementation of Process Optimisation

7.1 Introduction

The important role of simulation procedures in the design and/or optimisation of semiconductor fabrication schedules has been illustrated during the preceding chapters and requires little elaboration here. It has also been shown that the application of experimental design techniques enables a reduced data set to provide comprehensive analysis over the entire region of interest. However, even on this minimised data set, it is not hard to envisage how the total number of simulations can escalate, given that each design point may require many different simulations (e.g. process, device, parameter extraction, etc).

To date these simulations, when attempted, would generally have been performed as one single serial batch process on an available workstation or minicomputer system within an engineering facility. Unfortunately, this approach has a number of drawbacks associated with it. The most obvious of these is the CPU limitation and consequently the elapsed time between batch submission and completion. Additionally, the memory and disk space claimed by such software usually renders most typical hardware platforms more or less useless for all but the smallest of other applications, due to the inefficiencies of continually swapping processes. Realistically, this either necessitates the provision of a dedicated simulation machine or enforces a limit on the frequency and size of the tasks submitted. Neither approach is particularly desirable or constructive for a commercial manufacturing or development establishment.

The second problem lies in the enormous number of data files which need verification and unique identities before each individual simulation is performed. Unless a robust method for controlling the file system is employed, then great scope exists for the wrong files to be used for one or more simulations leading to totally erroneous results.

One solution to this may be found by applying the research discussed during the course of Chapter 5, whereby a parallel version of each software package runs on a transputer network. This dedicated performance port can substantially reduce the overall computation time by accelerating every simulation and in addition it relieves the host machine of the most intensive computation. However, the resources required to develop parallel codes for all the software packages that might be required becomes excessively costly in economic and manpower terms as was clearly illustrated in Chapter 5, when a parallel version of SUPREM II was under development.

An alternative strategy can be considered for multiple simulations associated with the experimental design techniques previously described, since the problem can be broken down into subsets of independent task. Such a scenario may be referred to as *operational parallelism* and is ideally suited to a *task farm* solution (see Chapter 4.). Here, the individual tasks are performed on any available processor in a given network, each of which executes a serial program on a number of data sets. Using this approach, an approximately linear increase in the overall performance can virtually always be assured as the number of computational processors is increased. A more detailed analysis of the attainable performance and its limiting factors is provided later in this chapter.

The application of a task farm system to the computational problems presented by the Design For Manufacturability (DFM) issue, provides many advantages over its rival solutions. Each application software package only has to be compiled for serial execution on a single processor. Porting such software generally presents relatively few difficulties providing that the code does not rely extensively on system dependent features. An objective evaluation of the packages that would typically require porting for the particular application under consideration is made, then the issue of system dependence should not create many problems. The individual packages are frequently supported on a variety of platforms and thus it is in the vendors' own interests to minimise such dependencies. With this in mind, under most circumstances it is therefore relatively simple to create a comprehensive

library of binary codes capable of running on the farm. The one outstanding issue which prevents progress is the availability of the source code. However, although most software vendors are understandably unwilling to release their source code, other than at extortionate prices, they can be more amenable to providing cross-compiled binary code for the transputer.

Little imagination is required to see how it is then feasible to produce an very flexible computing environment, whose scope ranges far beyond the bounds of applying DFM to the semiconductor industry. A selection of such applications will be proposed and discussed later in this chapter (§7.4).

This chapter discusses the design and development of one task farm system to provide an efficient and economic environment within which the DFM concept previously presented can be pursued. Firstly an overview of the entire hardware and software system will be provided, followed by a more detailed discussion of the constituent modules. Thereafter, the link with the RS/1™ experimental design tools will be shown. This automates the generation of the simulation data files, the subsequent program execution and extraction of the desired output data, as required by a particular experiment. Finally, the system performance will be demonstrated using a mixture of SUPREM II simulations and a set of programs designed specifically to enable the effect of altering the balance between communication and computation to be observed. Consequently, this allows the overall system performance to be concisely specified and permit users to maximise the data throughput for each application.

7.2 System Overview

The design of the task farm architecture is most effectively illustrated by investigating, in the first instance, the system services required to port a serial application onto a single transputer. Then the subsequent replication of these services for multiple applications enables the modular structure of the complete system to be defined. Figure 7.1 shows the starting point for this work by illustrating a general application running within its original host environment. Although a PC

host has been depicted in the following illustrations, any of the hosts currently available to support transputer systems may be used. At present these are IBM PC or compatibles, NEC PC, DEC MicroVAX II or Sun-3 [1].

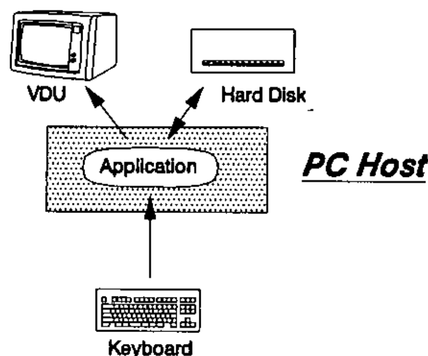


Figure 7.1. Application running within host environment.

There are two particular points to be highlighted in this generalised view of a typical application. Firstly, the only assumption made of the host is that the application only accesses the screen, keyboard and file system facilities by standard calls to the run-time library. Secondly, it is significant that this scheme provides no access to host-dependent interfacing features. As will be discussed in §7.2.1.3 such facilities can be made available indirectly. Figure 7.2 shows the system once the application has been ported in its entirety across to a single transputer. This may be described as a *flat port*, in the sense that the program structure remains completely unaltered.

The transputer communicates with the host along a single link which is connected to the host bus via a dedicated *link adaptor*. A simple *server* program runs on the host at the same time as the transputer-based application in order to provide access to the screen, keyboard and filing system. A master/slave relationship exists between the transputer and host, with the transputer assuming the master status and initiating all the communication messages. Within this system architecture, the transputer connected directly to the host bus is known as the *root* transputer. Additional transputers can be connected to the root transputer using

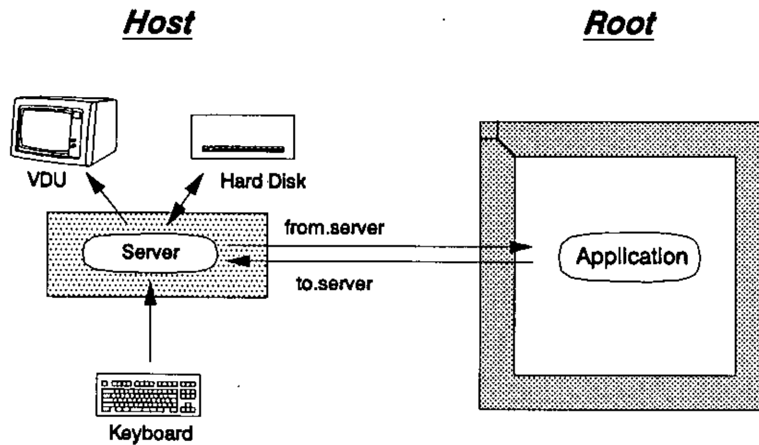


Figure 7.2. Serial port to a single transputer.

any of its vacant links. However, regardless of the number of processors in a network, the single link to the host bus provides the sole route for all system service access. The effect of this will vary depending on the functionality of the processes placed on each processor. For example, if an application lends itself to algorithmic parallelisation, (i.e. employing concurrent execution of separate functional modules) then wherever possible, all the I/O modules would be placed on the root processor where direct access to the host bus is available. Equally, in certain parallel application such as ray-tracing there is no system I/O required by the computing modules and again no problem exists. Unfortunately, the generalised task farm system which is being designed here, requires that all the computing processors have access to the system I/O just as though it were running either on the host or on the root transputer and thus possessed a direct link to the system services. A solution to this may be found by providing a multiplexing facility on the root transputer which provides each processor attached to it a communication path with the server process on the host. Thus, the system now appears as depicted in figure 7.3.

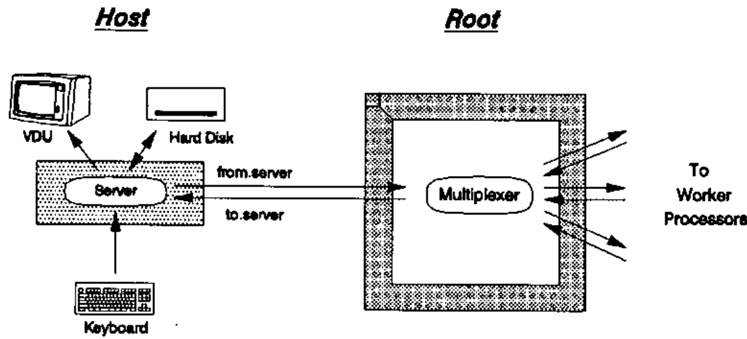


Figure 7.3. System structure showing I/O multiplexor.

Within a task farm environment, each of the computing processors is generally referred to as a *worker* and the controlling process as either the *farmer*, *master* or *driver* process. With only four physical links on the T*xxx* series of transputers, a maximum of three workers may be directly attached to the root processor, which somewhat limits the computing power available. Therefore, is becomes essential to introduce a method of routing messages around a transputer network from process(or) to process(or) to increase the number of workers. Such routing facilities may be simple, topology-dependent tasks or far more intelligent, topology-independent processes. The issue of inter-process(or) communications will be discussed later in §7.2.1. The system now stands as shown in figure 7.4 which includes all the modules presented up to this point.

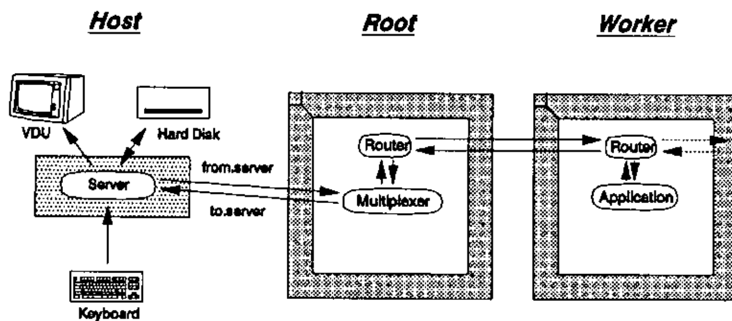


Figure 7.4. System structure showing worker and routing processes.

The performance attainable from a task farm system should be linearly related to the number of processors in the farm system. However, by multiplexing all of the standard system I/O channels onto a single channel pair, a potential communications bottle-neck has been introduced. The actual degradation in system performance will of course depend on the volume of traffic on these links and hence it is essential to minimise this. In considering this matter, it is important to reflect on the volume of communication traffic in relation to the computation done in between. As will be explained in §7.2.1.3, the data protocol supported by both of the standard server processes available, is excessively complex and verbose. It is, however possible to simplify the data transmissions across the farm channels, since the protocol is fully documented [2],[3] and the precise format of every command is known in advance. Each worker therefore has a process running concurrently with the application which compacts the out-going server data for transmission across the farm channels and expands the in-coming data before passing it back to the application. Obviously, this procedure is repeated in reverse on the driver processor for communication with the server process across the host bus interface.

The last major module lies only on the driver processor and this controls the farm's operation - a *manager* process. This takes responsibility for supplying each worker with the relevant program code and data when required as well as monitoring the status and performance of each worker. A more detailed explanation of the manager functions is provided in §7.2.4. A much simplified manager process resides on each worker to supervise the loading and execution of the program codes passed to it.

Hence, the overall system design may be schematically represented by figure 7.5. In the following sections of this chapter, the function and implementation of each module will be reviewed in detail.

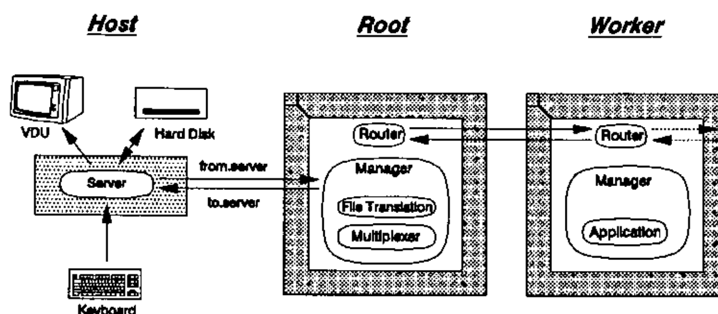


Figure 7.5. Schematic representation of the overall system architecture.

7.2.1 Communication Procedures

7.2.1.1 Introduction

Before discussing the communication requirements and capabilities of the T*xxx* series of transputers, there are a few background concepts relating to this issue which would be most appropriately explained at this juncture.

The rationale behind the design of a transputer based system is closely linked to the relationship between the software and hardware architectural models. OCCAM may be considered as a 'process-oriented' language which enables concurrent systems to be defined in terms of an interconnected set of *processes*. These processes can be regarded as black boxes which can be specified completely in terms of their communication requirements along the point-to-point channels which link them together. Internally, each process can then be designed as a further set of communicating processes, thus automatically generating an inherently hierarchical design structure as illustrated in figure 7.6.

For the sake of completeness, it should be pointed out that although processes may be written in any supported language, any non-OCCAM ones are subsequently encapsulated by a simple OCCAM harness. This serves to make them appear as an equivalent OCCAM process and relieves any possibility of restricting their use within a multi-process (and hence multi-language) environment.

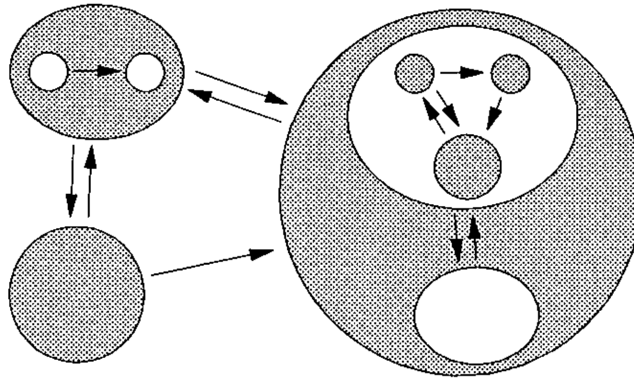


Figure 7.6. A collection of communicating processes.

This OCCAM programming model is implemented in hardware by the transputer itself. Within the bounds of a single processor, concurrent processes share the CPU/FPU time and inter-process communications are performed by data transfer in memory. Where multiple processors are involved, then the inter-process communications are directly implemented by the physical links. However, the important notion supported by OCCAM is that the *logical behaviour* of the software is independent of the mapping of processes to processors. This allows for flexibility and incremental expansion of transputer based systems. Figure 7.7 demonstrates this by showing two possible implementations of a three process system.

Logically, these two systems are identical, but will differ with respect to their real-time response. Consequently, a program may be designed and tested on a single transputer before being transferred to a network. Additionally, this also allows extra processors to be added to a network or alternative transputer configurations to be investigated without any alteration to the process definition of the system. However, there is one limiting difference between the OCCAM model and the T*xxx* hardware architecture which, under certain circumstances, can prevent the completely arbitrary process to processor mappings suggested above. The OCCAM point-to-point process interconnection scheme is simply supported for concurrent processes on a given processor, since the channels are implemented by memory locations. Hence it is possible to define and use a large number of channels within

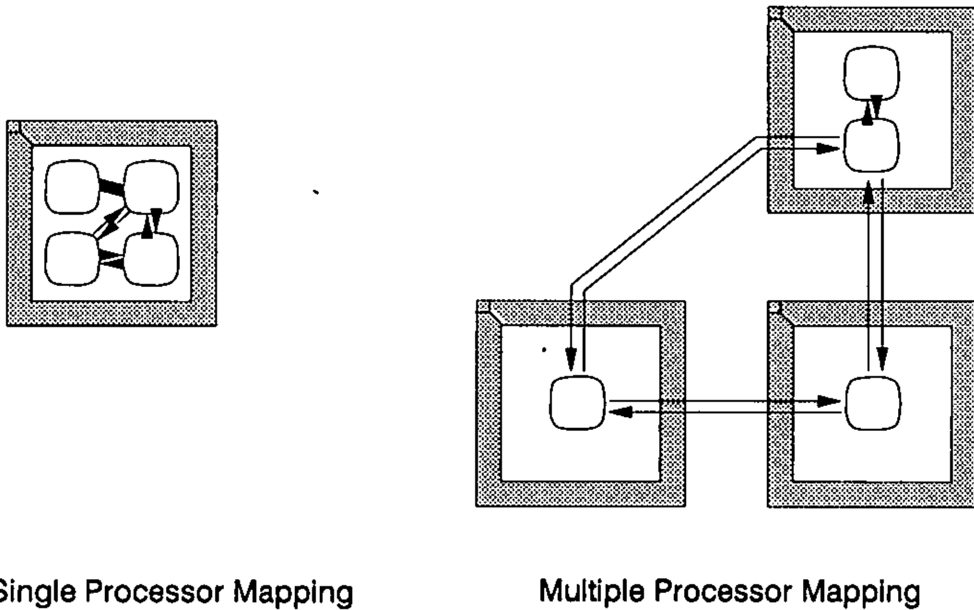


Figure 7.7. Different process-to-processor mappings with identical logical results.

the confines of a single processor. If on the other hand, some of these processes are now placed on neighbouring processors, there are only four physical links onto which the relevant channels may be mapped and there-in lies the problem. Primarily, there are three potential solutions to this dilemma, of which two are relevant to the Txxx transputer series. Firstly and least favourably, this hazard may be considered at the process design stage and the number of inter-process channels limited at all levels in the hierarchy. This then permits total freedom in the process to processor mappings when the system is subsequently configured. A variation on this concept, is only to limit the channel usage at particular process hierarchies and in this case it is the selection of permissible process to processor mappings which becomes limited. Both scenarios compromise the conceptual freedom of system design and implementation built into the OCCAM language and are thus undesirable. The second approach is to produce a dedicated communications harness which automatically multiplexes the users' channels onto the available links. Two examples of such harness' which have been developed on the Meiko Computing Surface at Edinburgh University are TITCH[4] and TINY[5]. The former was

OCCAM based and the through-routing overheads associated with it were fairly high - in excess of 330 microseconds[6]. The development of TINY evolved away from traditional OCCAM programming with processes sharing data buffers and was subsequently written in a combination of C and transputer assembler alone. With these improvements, the overhead was reduced to 20 microseconds, a value that could only be further diminished by hardware modifications within the transputer itself[7]. TINY was chosen as the harness for this task farm system and consequently it will be discussed in greater detail in §7.2.1.2. The final solution is only applicable to the recently announced T9000 transputer, which is capable of supporting virtual channels across its four physical links through the use of a *virtual channel processor* within the hardware architecture[8]. This permits channels to be defined between any two processes using hardware as opposed to the software solution provided by TINY. Apart from the upgraded performance to be attained from the T9000, this virtual channel capability must be the greatest single improvement over the T*xxx* series of transputer.

Throughout the course of the following subsections, an outline of the role and structure of each major process within the system will be presented. Although the system has been entirely coded, compiled and configured using 3L Parallel C [9], OCCAM has been used for the sake of clarity in many of the pictorial representations of code segments since it can explicitly display concurrency using the PAR construct. Any code examples provided within the following text have been written with the intention that no particular knowledge of the language be required for their comprehension. However, a complete definition and thorough tutorial of the language may be found in [10] and [11].

7.2.1.2 The TINY Communications Harness

The issue of communication poses one of the greatest challenges to the parallel computing applications programmer. This is especially true in the case of distributed memory MIMD machines, such as a multiple transputer network. When messages only need to be exchanges between directly coupled processes or a predetermined pattern of communications exists, then it is not that difficult to

create a suitable communications system. However, once the situation involves disjoint processes or indeterminate communications patterns, then the solution no longer can be considered trivial and may seem virtually intractable. Under these circumstances, the inclusion of a general purpose communications harness capable of forwarding messages from any user process to any other process, becomes highly sensible, if not essential.

There are two approaches to the provision of such communications facilities. The first is to write and run the application under a dedicated parallel programming environment such as CSTOOLS from Meiko [12], Helios from Distributed Software Ltd [13] or TAOS from TAO Systems [14] all of which inherently implement the required logical process connectivity. With respect to this work, use of either of the latter two options would have required the acquisition of the respective operating system for both the development and subsequent use of the application, whilst the first would have constrained the system to a Meiko computing surface. Whilst these restrictions are surmountable, it must also be noted that at the outset of the project, none of the previously mentioned tools was openly available in a fully operational form and thus this route was discarded. The second approach is to access a library of procedures which are callable from a variety of user programming languages such as the TINY harness developed at Edinburgh. This then allows complete portability of the software and retains the desired independence from any particular operating system. With these factors in mind and the ready availability of the TINY libraries for inclusion in 3L Parallel C programs, this harness was therefore chosen for development of the task farm system being presented here.

The basic concept behind a topology independent communications harness can be illustrated by figure 7.8, which shows the processes residing on every processor.

Each user process is connected to a harness process whose responsibility it is to provide a data path between all the user processes whilst minimising message latency, CPU impact and avoidance of deadlock conditions. Details of how TINY

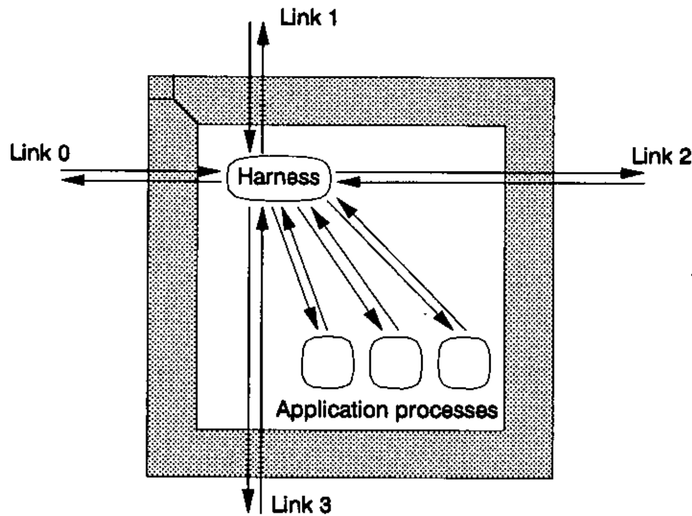


Figure 7.8. Process structure within the harness environment.

concerns itself with these issues is presented in [15]. The remainder of this section will provide a brief summary of the TINY harness and its application to the task farm.

TINY is available as a set of library procedures in either C or OCCAM, which call an underlying series of C functions to perform the following tasks.

- Determine the processor topology
- Determine the process mapping
- Calculate the routing tables
- Spawn the router process
- Read and Write messages

Each user process is given a unique integer address for indentification purposes throughout the network. In addition, either a single or multiple channel interface to the local routing process may be specified, where every channel may have separate properties assigned to it in terms of message synchrony and read/write permission (channels can be defined for read only, write only or

read/write operations). Thus a portion of a configured network could take the form shown in figure 7.9, which illustrates two processors each of which is supporting a number of differently defined user processes.

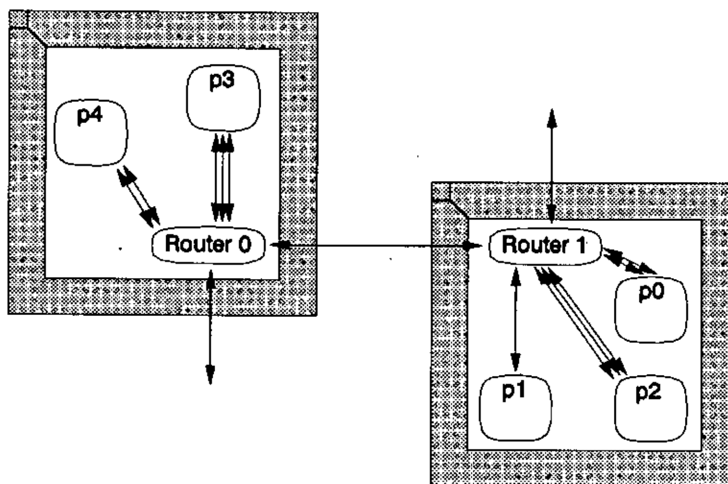


Figure 7.9. Process structure within part of a configured network.

Three different routing strategies are supported under TINY, which are referred to as broadcast, sequential and adaptive respectively. The function of the first is fairly straightforward. The message is forwarded to all other user processes in the network, with the distribution ensuring that only one copy of the message is received by any node in the system. With sequential transmission, the message is forwarded along a fixed and predetermined path to its eventual destination, thus ensuring that message ordering is preserved. Finally, with adaptive transmissions, the message is forwarded initially over the same path as used for the sequential case, but then branches along the least busy path whenever a choice of equidistant routes exists. In this way, an attempt to balance the overall network traffic is made at the expense of message ordering which can no longer be guaranteed. Figure 7.10 shows an example of each transmission strategy along with their associated datatype definitions, as would be instanced using a C syntax. It is assumed that TINY has been successfully initialised previously within the code.

```
#include   "tiny.h"

int ok;                              /* Success or failure error code */

int type,dest,length;
int *source;
void *data;

ok = t_sseq(type,dest,data,length);

ok = t_recv(type,source,data,length);
```

Figure 7.10. Example of TINY message transmit and receive strategies.

The transmit or receive buffer may take any valid C data type providing the user ensures the length is specified in terms of bytes and not multiples of the particular data type. Thus for use within the task farm, a unique buffer structure was defined for each message type that would be transferred across TINY from a worker process to the driver and back again. Figure 7.11 shows an example of one such buffer and figure 7.12 illustrates how all the buffer structures are brought together using the C *union* construct.

```
struct afsOpenFile
{
    int accessMethod;
    int openMode;
    int existMode;
    int recordLength;
    int nameLength;
    int priorityLevel;
    char fileName[MAX_RECORD];
};
```

Figure 7.11. Buffer structure for *OpenFile* file server command.

```
struct afsBuffer
{
    int tag;                    /* Command(to.server)   or Result(from.server)   value */
    union
{
    struct afsOpenFile   OpenFile;
    struct afsReadBlockReadBlock;
    .
    .
} data;
};
```

Figure 7.12. Complete buffer structure for all file server commands.

In this way, the required command and parameter values could be easily packetised and unpacketised into the correct formats. Finally, all message transmissions use the sequential transmission strategy since the ordering of messages is vital.

7.2.1.3 The INMOS Alien File Server Protocol

As was indicated in §7.2.1.1, an interface exists between user processes on the root transputer and the hosts' operating system. The host executes a simple server process which supports a two-tier communications protocol which enables the transputer processes to access the appropriate host services. The original INMOS Alien File Server (AFS), **afserver**, supports access from the D705A OCCAM toolset and scientific language processes. The OCCAM-2 toolsets use a different server process, **iserver**, which permits binary-level platform portability and is therefore recommended [16]. However, the **afserver** source code is supplied along with its associated compilers which facilitates user modifications and hence was chosen for this system. It is intended to provide only a brief overview of this server protocol at this point, to serve as sufficient background for discussing its modification and subsequent implementation over TINY within the task farm system. A more complete description is available in [17], [18].

The basic protocol between processes consists of an explicitly tagged protocol, where the tags indicate the data type of the value which follows immediately. There are 13 tags within the full protocol specification, although only a reduced subset of these are actually implemented in current servers. For ease of programming it is customary to define all of these as constants within a header file which can be included where necessary. An excerpt from an OCCAM definition of the tags is shown in figure 7.13. At present, only the final three tags are supported, though the remainder are reserved for use in future implementations of the server.

```
bool.value        IS    BYTE    0:
byte.value        IS    BYTE    1:
.
int32.value       IS    BYTE    4:
.
nilrecord.value   IS    BYTE    8:
.
record32.value    IS    BYTE    12:
```

Figure 7.13. OCCAM definition of AFS tag values.

Thus the basic *integer* and *record* data types can be produced from these tag variants as shown in figure 7.14. This indicates how an integer is sent as **int32.value** (showing that a 4 byte integer is about to be sent) followed by the actual integer value. Similarly a record of bytes is transferred by first sending, **record32.value**, then a 4 byte integer representing the number of bytes in the record and finally the appropriate sequence of bytes.

```
integer   =  int32.value  ; INT32

record    =  record32.value  ; INT32 :: []BYTE
          =  nilrecord.value
```

Figure 7.14. Tagged definition of integer and record data types.

As can be seen, a means to send zero-length record has been incorporated to maintain continuity. In this case, the **nilrecord.value** tag is sent with no following sequence of bytes.

AFS commands all consist of an integer command value followed by a predetermined set of integer and/or record parameters. The server will respond to this command information and return an appropriate group of integer and/or record values. There will always be a minimum of one integer result returned from the server, whose value indicates the success or failure of the operation. In the same manner as with the basic data tags, it is normal to define all the command values and their associated constant parameters in a header file. Figure 7.15 shows a few such definitions, whilst figure 7.16 illustrates the syntax of some commands. Both of these figures use OCCAM definitions, as seen previously, but the concept is transferable to whatever language is being employed for a given process. A full listing of all the available command values and their respective syntax is given in [19].

```
OpenFile :
to.server=OpenFile.Cmd   filename  access.method   open.mode  exist.mode  record.length
                 integer     record     integer      integer    integer    integer

from.server        =  stream.id  result
                      integer  integer

ReadBlock :
to.server      =  ReadBlock.Cmd   stream.id  record.length
                     integer      integer   integer
from.server        =  record  result
                      record  integer
```

Figure 7.16. AFS Command syntax.

When the basic tagged protocol is added to this AFS protocol, the extreme verbosity of this syntax becomes fully apparent. This is illustrated in figure 7.17

```
OpenFile.Cmd              IS     1:     -- Commands
OpenTemp.Cmd              IS     2:
OpenInputStream.Cmd       IS     3:
.
ReadBlock.Cmd             IS     12     :
.
Terminate.Cmd             IS     24     :
.
RunTimeData.Cmd           IS     34     :
.
.
TextByteStream.Access     IS     1:     -- File access and
Read.Mode                 IS     0:     -- Open modes
Write.Mode                IS     1:
.
.
OperationOk               IS     0:     -- Error conditions
EndOfFile                 IS     1:     -- Server errors
FileNameTooLong.Err       IS     3:
.
FileAlreadyOpen.Err       IS     10     2:-- Host errors
```

Figure 7.15. AFS Command value definitions.

below where an example of one AFS command is shown in full. The total number of bytes for the given operation are also indicated with each appropriate value and parameter.

The negative aspect of this verbose syntax is further compounded by the fact that link communications operate by the transmission of single bytes [20] rather than by packets of data as has been incorporated into the T9000 communications protocol [21]. Hence, even when only a single process is communicating with the server, the number of byte transmissions will be substantial. However, if, as in the case of the task farm, there are many independent processes each attempting to access the link adaptor, then the motivation to compress each message is very apparent.

to.server	=	OpenFile.Cmd	integer	int32.value;OpenFile.Cmd
		filename	record	record32.value;12::"TESTFILE.DAT"
		access.method	integer	int32.value;TextByteStream.Access
		open.mode	integer	int32.value;Update.Mode
		exist.mode	integer	int32.value;OldFile
		record.length	integer	int32.value;16
from.server	=	stream.id	integer	int32.value;5
		result	integer	int32.value;OperationOk

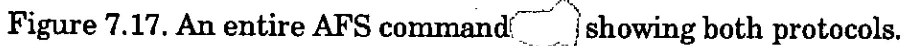


Figure 7.17. An entire AFS command showing both protocols.

If the above example of an AFS command in its expanded form is briefly examined, then it can be seen that the information provided by the basic protocol tags is actually redundant. The length of the message to the server could be reduced from 42 bytes to 36 bytes and that from the server from 10 to 8, by stripping the tags from the data transmission.

7.2.2 Worker Process

The worker process resides on all but the root transputer and is responsible for running any application available for the system. Figure 7.18 shows the processes that go to make this worker and their associated communication channels.

In terms of these sub-processes, TINY has already been discussed in §7.2.1.2, leaving this section to cover the application itself along with the basic manager and its protocol translation process. The application may be simply considered as a black box with a pair of communication channels which the process assumes are connected to the server, but which are in fact attached to an intermediate translation task before being routed to the server via TINY. This view of the application illustrates the concept behind the OCCAM model of concurrent programming mentioned previously. The role of the *manager* is perhaps most suitably explained by first looking at a simplified OCCAM description of its operation, as shown in figure 7.19.

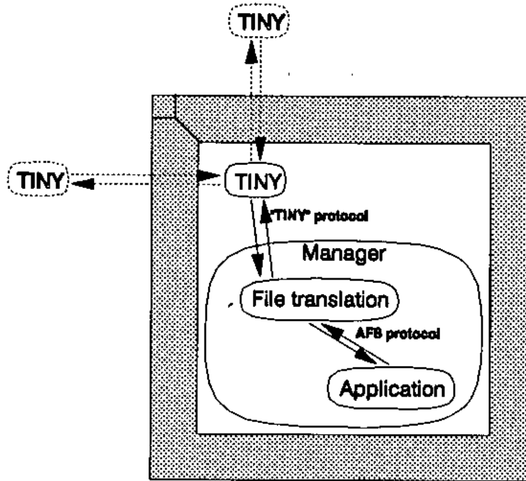


Figure 7.18. Worker processes and communication channels.

```
-- initialise router
initialise_Tiny

-- main job loop
WHILE (Jobs to be done)
  SEQ
    IF
      New program
        Receive code details
        Receive code
        Receive job details

      Current program
        Receive job details

    PAR
      start program
      link AFS protocol to TINY

    inform DRIVER when program has finished
:
```

Figure 7.19. Pseudo-OCCAM description the manager process.

The design and functionality of this process could have been much simplified by electing to nominate a single application at the compilation stage and thus statically including it in the overall worker process. However, this compromises much of the flexibility that could otherwise be offered by a task farm system. Therefore it was decided to permit different applications to be nominated as required at run-time and consequently this influences much of the logic within the process.

When the task farm is initially booted up, it is necessary for TINY to be initialised on each processor and so enable system intercommunications to proceed. If for any reason this initialisation fail, then the entire system will shut down and inform the user to this effect. Assuming TINY initialises successfully then system-wide process communications will be available until an explicit terminate procedure is activated at the end of all the required computation. All subsequent communications received by the worker will be with respect to loading, starting and running the users applications.

Before discussing the basic operation of the worker, it is beneficial to provide a brief summary of the various modes of communication available to a particular process. Depending on the *harness* and run-time libraries that are linked to the process binary image, one of three alternative communication schemes can be invoked, each of which requires to be handled differently. A more comprehensive discussion of these task variants may be found in [22].

The first case is that of the *Standalone* task, which may be considered as per figure 7.20. Such a task can **only** communicate using channel I/O functions (e.g. chan_in_word in 3L C) and **not** with the standard I/O functions (e.g. printf in C). This is because it is linked with a special standalone version of the run-time library of the relevant language which includes no support for these functions. Within this task farm system, the worker process would be configured as a standalone task, since it only communicates with neighbouring tasks and never directly to the server itself.

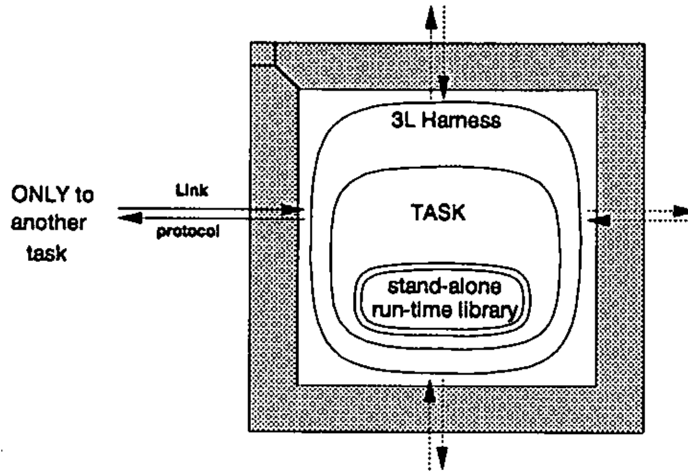


Figure 7.20. Standalone task communications.

The second case is the *Supported* task, as shown in figure 7.21. This task again uses the 3L harness as with the standalone task above, but now includes a full version of the run-time library to permit standard stream and file I/O using the standard File Service Protocol. The *driver* process from this task farm system would be configured in this manner since it is required to communicate directly with the server and with the workers attached to it.

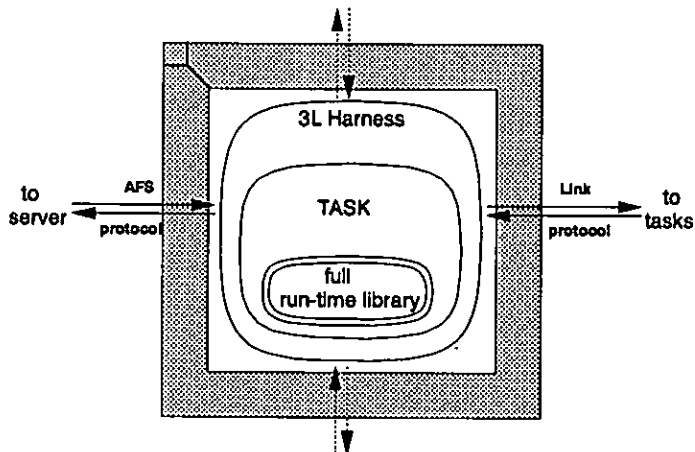


Figure 7.21. Supported task communications.

The final case is that of the *Inmos* task which is illustrated in figure 7.22. Whilst the full run-time library is incorporated, it is combined with a different harness which **only** permits direct communications with a server over a specified channel pair. These channels adhere to a modified version of the Inmos tagged File Service Protocol, adjusted to be tolerant of a problem with T414 Rev A transputers which is incapable of sending one-byte messages over its links.

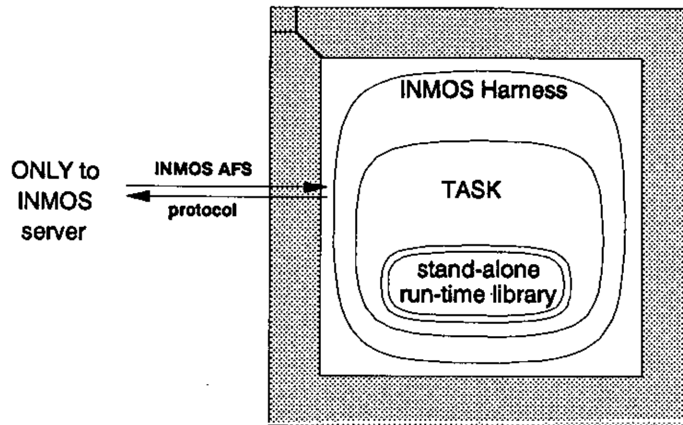


Figure 7.22. Inmos task communications.

It may be noted that this task is identical to the basic flat port of a general application to a single transputer environment which was presented earlier. Indeed, the same code may be used to boot and run the application in such a fashion in addition to being loaded and run from within another task as required by the task farm. On the basis on this last feature, the *Inmos* task is preferable to the *supported* task for use within the proposed system.

One other general point of some importance is that of the run-time implications brought about by the inclusion of either the full or standalone run-time library. If the full run-time library is linked to the task binary code to permit use of the standard I/O functions, then certain procedures are automatically invoked when the application is booted up. There are normally four standard streams associated with these functions - two input and two output, whose roles can be listed as follows.

Standard input stream 0 — This is the regular input stream which is usually associated with keyboard input It may be redirected on the command line by the operating system. This is the *stdin* stream within the C syntax.

Standard input stream 1 — This is a character buffer containing any command line parameters supplied by the user for use within a program.

Standard output stream 0 — This is usually reserved for the screen output, though may also be redirected from the command line. The *stdout* stream within C.

Standard output stream 1 — The output stream usually reserved for sending error messages to the user on the screen. The *stderr* stream from C.

The run-time library endeavours to open all four of these streams and allocated stream identifiers to them for later use by the calls to functions using them. In the case of the command line stream, an attempt is made to read in any characters that may be present in the relevant buffer and then the stream is closed. Obviously, it is therefore vital that any task using this library has access to the server, in order for these operations to succeed. Alternatively, any task which is not able to access the server, must **not** be linked with the full run-time library.

The final significant issue to be discussed at this time pertains to the access of these standard streams and the general filing system within a multiple application environment. Little imagination is required to envisage the chaos that could ensue when a number of applications all tried to write to the screen or read from the same file or the keyboard at once. Although two of the standard streams can be simply redirected via the operating system, the remaining two can not. In addition many programs use fixed, predefined filenames for specific data output such as debug information which again can not be easily be altered without changing the source code, which is compromising one of the original targets. It was therefore

decided that between the worker and *driver* processes, a suitable solution could be incorporated, thus leaving the application process unaltered and none the wiser of how and where it is being executed. As will be indicated in §7.2.4, with each application that is submitted to the task farm, a number of stream and filename alterations are supplied to provide unique stream and file identities for every application.

With the preceding background information in mind, it is now intended to continue with details of the *manager* process operation as shown earlier in this section. With TINY initialised, the *manager* must now load and then invoke the application task (as an Inmos task). The following actions must be performed for this to be done.

- Allocate memory for application code and workspace
- Load application code
- Arrange communication channels to/from application
- Start application task
- Handle application task communications

Information pertaining to the task memory requirements can be extracted from the executable code and are passed to the *manager* from the *driver* in advance of the actual code itself. Obviously, this only occurs when a new application is needed on a particular processor since the stored code may be invoked as often as is demanded. However, for every job, the *manager* requires a list of stream and filename translations to ensure unique identities are maintained throughout the system. In addition to this list, details of any command line parameters for *standard input stream 1* are also included. A more detailed examination of this data will be presented in §7.2.3 and §7.2.4. The precise details of invoking the code may be found in [23]. In parallel with the application task, is the process which links the AFS task channels to TINY, adapting the format and content of all messages as necessary. The basic operation of this protocol interface process may be summarised by the pseudo-OCCAM description provided in figure 7.23.

```
WHILE task running

    Receive AFS command value from task channel
    SWITCH(AFS command value)

    CASE OPENINPUTSTREAM_CMD:      IF stream_1
                                       ignore -- see later
                                   IF stream_0
                                       apply stdin translation
                                       fill TINY buffer with OPENFILE_CMD
                                       break
    .
    .
    CASE General_CMD             : read in remaining parameters
                                   apply any translations necessary
                                   fill TINY buffer with CMD info
                                   break
    .
    .
    END SWITCH

    send TINY buffer to DRIVER
    receive TINY buffer from DRIVER

    SWITCH(AFS command value)
    .
    .
    CASE GENERAL_CMD             : unpacketise TINY buffer
                                   undo any translations applied
                                   send results in AFS protocol to task
                                   break
    .
    .
    END SWITCH

Send TERMINATE_CMD to DRIVER
```

Figure 7.23. Pseudo-OCCAM description of protocol-translation process.

The majority of the AFS commands require some alterations to be performed whilst packing up the information into the TINY buffer structure. Obviously any involving filenames require this string to be changed to its new unique value on-the-fly, and likewise any filenames being returned from the *driver* must be

translated back to their local value. However, one or two commands require a slightly more subtle approach to be taken. For example, take the RunTimeData.Cmd, which would normally read the command line buffer from standard input stream 1 (which, for a genuine standalone task would already have been opened). In this case, the request to open this stream is merely ignored, and the subsequent 'read' is replaced by copying the contents of the command line buffer sent down with the job details directly to the task.

A more devious technique is necessary to deal with the OpenInputStream.Cmd with reference to input stream 0 (i.e. keyboard). Here, the command is changed to an OpenFile.Cmd with the filename being the translation for *stdin* supplied with the job details. It is important to note that however much a particular command may be altered for transmission to the *driver*, the result which is returned to the application must acknowledge the **original** command, so as not to lead to confusion.

The only other command which is added to, rather than altered, is the Terminate.Cmd sent at the end of each application run. This is returned to the *driver* along with a number of extra parameters relating to system performance and errors applicable to that particular job. These parameters are then used to produce formal system log files for the user when the system is finally shut down. Although on a single transputer application, this command would also be used to terminate the host server process, within this system it merely signifies to the *driver* that a particular worker is available for the next task to be performed. Only when all the jobs submitted to the system have been processed, will the server be instructed to terminate and return control to the host operating system.

7.2.3 Driver Process

The *driver* process resides on the root processor and is responsible for managing and monitoring the entire system operation. Therefore, it provides each worker with continual stream of jobs, automatically ensuring that the correct application code resides within the respective workers and then multiplexes all the TINY communications from each task onto the server channels. The basic process structure along with the communication links is shown in figure 7.24.

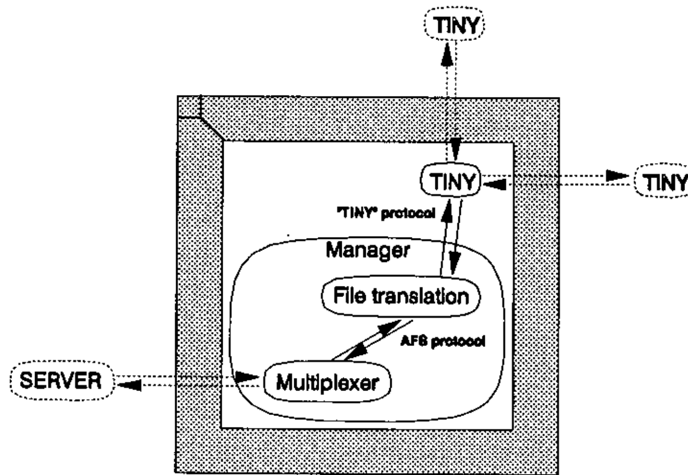


Figure 7.24. Driver processes and communication channels.

This figure includes one extra process, *filesys*, over and above those previously illustrated for the driver. As will be explained shortly, this process is required to overcome specific limitations imposed by the host operating system. Of the remaining processes, TINY has already been discussed in §7.2.1.2 and the *mux* process will be shown to be inherently contained within the scope of the basic *manager* logic. It is best to lead into the discussion of this logic by considering the simplified OCCAM description of its algorithm shown in figure 7.25.

The details of the command file from which system instructions are taken will be presented in §7.2.4. Hopefully, it is also possible to see how the manager inherently multiplexes the incoming application messages by not reading another TINY buffer until the previous message has been replied to. In this respect, TINY not only services the system communications, but also provides an implicit message buffer for all the processes it supports by the nature of its internal operation.

In contrast to the worker, very few AFS commands require direct modification before being passed on to the server itself, mainly because this has already been done prior to their transmission. Equally, certain commands will never be received by the driver since they will have been trapped and dealt with within the confines of the worker (well that is the plan !). Obvious examples of these are the OpenIn-

```
.
.
initialise TINY
open batch command file

WHILE jobs to do

    receive message buffer from TINY
    IF AFS message
        IF OPENFILE_CMD
            IF files open > MAX_OPEN_FILES
                close a lower priority file
        END IF
        unpacketise TINY buffer & send to server in AFS protocol
        get result back from server
        packetise result into TINY buffer
        return buffer to respective worker

    ELSE
        – node awaits new job
        read batch file entry
        IF program not on this worker
            send out new code
        send out job details

    update screen with system and job status
.
.
```

Figure 7.25. Pseudo-OCCAM description of the manager process.

putStream.Cmd and OpenOutputStream.Cmd commands which will either have been transformed to OpenFile.Cmd or dropped altogether on reaching the driver as indicated in §7.2.2.

However, it is on account of limitations placed on run-time file system access by the operating system, that the most demanding process, *filesys*, need be introduced. Most operating systems place a limit on the number of files which may be opened at any given time. For SunOS™ this value is 64, although mechanisms exist for extending this in some cases [24], but for (MS)DOS™ the value is set at 16. This low value becomes somewhat limiting in a multi-application system where each

task is trying to open its own complete portfolio of files under the pretext of being the sole application accessing the file system. In short this means that OpenFile.Cmd's can not be simply passed directly to the server, since the operation will fail once all the available file handles have been allocated. However, the application will be expecting a successful outcome and most tasks will either exit or crash should failure be detected. Consequently, it is necessary to produce and maintain a *logical* file system within the driver, which is capable of supporting as many files as may be required by all the applications. In this way, each application will be given results which enable it to assume that these file operations have been successful and that the relevant files are currently in an open state.

In order to achieve this logical file system, the driver maintains a file table for each application running under the system. Whenever this requests an OpenFile.Cmd, the file is temporarily opened and an entry made in the respective file table as to its genuine file identifier, the pseudo identifier returned to the application, the file pointer position, the file access mode (read/write/append) and type (binary/text) and its true open/closed state. The file is then actually closed. Whenever subsequent operations are required on this file (except CloseFile.Cmd, obviously) it is reopened, the file pointer correctly positioned and the resulting pseudo identifier used to enable the desired operation to proceed. Again on completion of the action, the file pointer entry in the correct file table is updated and the file is then reclosed. Unfortunately but hardly surprisingly, this open file, move pointer, act, update pointer and close file cycle is substantially slower to execute than the original file operation. This can dramatically influence the overall performance of batches which make continual use of the file system during the course of their operation. As a compromise, it is therefore desirable to keep as many files genuinely open and only perform this open/act/close routine when the system file limit has been reached. In this way the maximum performance can be attained for a given file limit. The logic required to implement this would be trivial if it were not for the fact that an intelligent method of allocating the available open files amongst all the application must be included.

To this end, two steps in particular were taken. Firstly, it was decided that without prior knowledge of the file usage of all the applications that might be running on the system at a given time, an equal allocation of the available files would be made to all workers. This allocation is made during system initialisation, with each worker being given (maximum system files/No. workers) files. Secondly, the files relevant to each application would be given one of two priority levels with respect to it remaining open for as much of the time as possible. In this manner, it was intended to keep as many of the frequently used files open across the system so as to incur the least performance degradation. The specification of a files' priority then becomes part of the job description within the batch command file, as will be illustrated in §7.2.4. Thus when the genuine file system is at full capacity, the driver will attempt to close files in the following fashion.

1st Low priority files on same worker.

2nd Low priority files on other workers.

3rd High priority files on same worker.

4th High priority files on other workers.

By following this approach, it is possible to maintain virtually the maximum attainable performance for a given file system capacity, without either prior knowledge of the applications requirements or the use of a complex optimisation algorithm. By monitoring the system performance, it was possible to confirm that the incorporation of this pseudo file system concept did indeed boost the file access time and results to this effect are presented in §7.3.

7.2.4 Control Procedures

The previous sections have provided a brief summary of the structure and function of all the main process modules which form the task farm system. It is now necessary to consider the way in which the user specifies the operation of the system. The driver process receives all the required details from a single control file which is specified when invoking the system from the host. There are two levels of information contained within this control file. Firstly, there are groups of data

statements pertaining to each job requiring to be processed. Figure 7.26 shows the basic format of the statements required to describe a single job whilst in figure 7.27, example statements for a SUPREM II job are illustrated.

```
Do        program name    command  line
Assign    filename-1      new  filename-1
..        ..                    ..
Assign    filename-n      new  filename-n
Stdin     redirected   filename
Stdout    redirected   filename
Done
```

Figure 7.26. General format of control file statements.

```
Do        suprem2.b4
Assign    SUPSEG          SUPSEG01
Assign    SUPSYNT         SUPSYNT01
Assign    SUPDEBUG        SUPDEBUG01
Stdin     RUN01.IN
Stdout    RUN01.OUT
Done
Do        suprem2.b4
Assign    SUPSEG          SUPSEG02
Assign    SUPSYNT         SUPSYNT21
Assign    SUPDEBUG        SUPDEBUG02
Stdin     RUN02.IN
Stdout    RUN02.OUT
Done
```

Figure 7.27. Control file statements for two SUPREM II jobs.

Around these 'job' statements can be placed a series of 'block' statements which can divide the jobs up into sub groups which must be completed before then next sub group is commenced. In this way, it is possible to ensure that, for example, all the process simulations are completed before the device simulations depending on

their output are started. This concept can be considered to be analogous to the dependencies listed in the *make* file for a software package. This 'block' concept is shown in figure 7.28, where the particular job details have been omitted for clarity.

```
Begin BLOCK-1          -- first block name

Do
..                     -- job No. 1
Done
Do
..                     -- job No. 2
Done

End
Begin BLOCK-2
.
...                    - more job assignments.
.
End
```

Figure 7.28. Block structure within control files.

As discussed previously in §7.2.3, the priority of any particular file is given by its order within the job description, with the highest priority being defined first. When the driver reads the final job description from the control file, it awaits termination of any outstanding applications on the system before shutting each worker process down in turn.

The final point to be mentioned concerns the link between an experimental design within RS/1™ and the production of its associated control file and all the simulator files required by each job. The repetitive nature of all these files is highly apparent and thus their production lends itself to automation. Taking a SUPREM II experiment as an example, it can be seen that the following files will be needed to successfully run the batch.

- SUPREM II data files for each experimental point
- Control file for entire batch

The former are produced from a single SUPREM II template file, where the relevant parameters have been replaced by an identifiable character sequence. For each experimental point, the data is read out from RS/1™ and used to replace the relevant parameters within the simulation template file. An incremental name is then applied to this file and this is also written into the control file as the translation for *stdin*. The remaining details for the control file come from another template file which describes the actual simulator in terms of its filename and command line requirements. Again any standard application files are given incremental names for individual identification. Hence, a simple user program can be used to call RS/1™ and produce the desired experiment. It can then generate all the necessary files from two generic templates and then instruct the task farm to run this batch. On completion of all the jobs concerned, a similar approach is used to extract the desired results from the simulation output files and return them to RS/1™ for subsequent analysis.

7.3 System Performance

OCCAM and its complimentary set of parallel scientific language developments all consider the real-time aspects often associated with concurrent system, to a high degree. Therefore ways of measuring and apportioning time have been included in these languages using the TIMER construct [25].

The task farm performance was thus able to be measured by including a number of these timers throughout the driver and worker processes. From the results returned from these, the time associated with loading in the application code, subsequent application server communication and with the application computation can be individually identified for each job performed. In addition the elapsed time for an entire batch of tasks to be performed is measured. The results relating to each task are passed back to the driver process as part of the modified AFS Terminate.Cmd command and placed in the main system log file for later assessment.

The screen image shown in figure 7.29, illustrates the user interface whilst the system is in the midst of processing a particular batch. It can be seen that details are provided for both the system as a whole and the current state of each worker in turn. In this way should any worker cease to function then this will quickly be picked up.

```
                                                                 14:45:39

      System Info                                   Task Info
      -----------                                   ---------

      System Nodes :  5                             Task File      : cputest.ic
      Nodes Busy   :  3
      Nodes Ready  :  2                             Current Block : Block-1
      Nodes Dead   :  0                             Jobs completed: 2
                                                    Code in Memory: cputest.b4

  Node    Program     Ready Busy   Jobs done
  ----    -------     ----- ----   ---------
   1      cputest.b4   1     0      1
   2      cputest.b4   1     0      1
   3      cputest.b4   0     1      0
   4      cputest.b4   0     1      0
   5      cputest.b4   0     1      0

      Message from  3 :: len  20 :: tag  24
      Message to    2 :: len   4 :: tag   0
```

Figure 7.29. Screen image captured during system operation.

In order to show both the performance and shortcomings of the system, results from a series of experimental batches are presented. Each batch has been produced with the intention of highlighting a different feature of the system. However the first objective was to compare the execution of a representative application on a number of typical computing surfaces. Therefore the elapsed times for five similar SUPREM II simulations run on three different systems are shown in table 7.1. All the times quoted are in seconds.

Various points should be noted in terms of these results. Firstly, all times represent the time between issuing the initial host command and control being returned to the host command line. Secondly, the overall elapsed time in the case of the task farm includes the time taken for the system to be booted up. Unfortunately, because the T800 boards available outside the PC host only possess 256 Kbytes of memory, SUPREM II could not be run on them and so the slower T414 boards which possess 2 Mbytes RAM had to be used. Because SUPREM II relies

Simulation file	VAX 11/750	Tandon 286 PC	Worker T414	Root T800
RUN-01	502.5	695.6	1823	——
RUN-02	416.3	672.1	1834	——
RUN-03	524.3	692.3	1812	——
RUN-04	433.5	715.4	1776	——
RUN-05	493.8	940.5	1754	——
Elapsed Time	2,370.40	3,715.90	1875.00	

Table 7.1. Comparison of SUPREM II simulation times.

heavily on floating point arithmetic, there is a dramatic performance loss sustained when using the non-floating point T414 transputers. Therefore conclusions should not be drawn from the direct comparison of the last two columns in this table.

Whilst still considering the system performance relating to these SUPREM II simulations, the next issue to consider is how the reported elapsed time is apportioned between communication and computation. Figure 7.30 shows the system log file for the above simulations which includes the worker monitoring data.

Two particular points arise from this data. The first of which is the time taken to load all five copies of the application code onto their respective workers. The second is the ratio of communication to computation times, which are excessively out of balance. When these results were taken, each worker loaded its code directly from the host file system. Unfortunately, all block data transfers between the host and the root transputer have a maximum block size of 512 bytes which is set within the run-time library. Thus, no matter what size of data block the user requests, if this is in excess of 512 bytes then the run-time library will automatically perform multiple block transfers until the desired limit is reached. The application code for

```
                    Edinburgh Microfabrication Facility

                          Simulation Taskfarm System

System Nodes : 5

Batch File   : suprem5.rc

Node            Program             Calc    Comm    Load    Misc
====            =========           ====    ====    ====    ====
  1             suprem.b4            628    1164      27       0
  2             suprem.b4            638    1165      27       0
  3             suprem.b4            642    1140      27       0
  4             suprem.b4            632    1113      27       0
  5             suprem.b4            644    1079      27       0

 Completed 5 jobs : Elapsed time 1875 seconds
```

Figure 7.30. System log for SUPREM II simulations.

SUPREM II occupies just over 300 Kbytes which requires in excess of 600 block transfers, and leads to the 27 second load time. When applications such as SUPREM IV and PISCES are considered, the problem is even worse, as both of these codes occupy over 2.5 Mbytes at which point the initial code loading time would outweigh much or all of the advantages to be gained in using the high performance T800. The driver logic was therefore altered so as to store the loaded application code in its memory, assuming sufficient capacity, and load any other workers requiring this code directly from memory, thus cutting out the slow disk transfer. Only when a new code is needed does the slow file system transfer have to be repeated. In this manner, the above system log now takes the form shown in figure 7.31, where the load times reduce from 27 seconds down to 4 seconds for all but the first node and the elapsed system time now becomes 1787 seconds.

It is apparent that the time spent by the application communicating with the host server via TINY is somewhat disproportionate to the onboard computation. One major factor behind this imbalance is the way the driver handles the file system. As was discussed in §7.2.3, (MS)DOS™ places a limit on the number of files

```
                    Edinburgh Microfabrication Facility

                              Simulation Taskfarm System

System Nodes : 5

Batch File   : suprem5.rc

Node          Program               Calc    Comm    Load    Misc
====          =========             ====    ====    ====    ====
  1           suprem.b4              628    1164      27       0
  2           suprem.b4              638    1165      5        0
  3           suprem.b4              642    1140      5        0
  4           suprem.b4              632    1113      5        0
  5           suprem.b4              644    1079      5        0

 Completed 5 jobs : Elapsed time 1787 seconds
```

Figure 7.31. System log with reduced loading times.

which may be open at any one time. The above results were taken when **all** the files were open and closed around every operation introducing substantial delays. Table 7.2 shows the times once the driver maintains its full capacity of open files.

Although this table confirms a moderate improvement in file handing performance, it must be noted that there are 30 files (6 per worker) competing for the 13 system files available (The PC requires 3 of the 16 for its own *stdin*, *stdout* and *stderr*). Thus many file operations require the inefficient open and close actions. In order to confirm this assumption, a further batch of simulations was processed with 3 jobs which meant that there were 18 files competing for the 13 available slots. As can be seen in table 7.3, this produces a substantial performance increase over the original system.

Data file	Comms. No files open	Comms. Max. open files	Comms. % increase	Total No files open	Total Max. open files	Total % increase
RUN01	1164	1025	11.9	1823	1689	7.4
RUN02	1165	1079	7.4	1834	1753	4.4
RUN03	1140	1002	12.1	1812	1680	7.3
RUN04	1113	984	10.7	1776	1652	5.4
RUN05	1079	940	12.9	1754	1620	7.6

Table 7.2. Improved file system performance.

Data file	Comms. No open files	Comms. Max. open files	Total No files open	Total Max. open files	Total % increase
RUN01	1164	379	1823	1053	42.2
RUN02	1165	442	1834	1106	39.7
RUN03	1140	367	1812	1045	42.3

Table 7.3. Performance with full file system access.

The method of server communication over TINY with its associated message packetising and unpacketising on both worker and driver, does degrade system performance. But it has been shown that by carefully selecting the use of application files, this can be minimised so that a more acceptable communication to computation ratio is attained. It should also be noted that in all the examples above, each SUPREM II job writes out approximately 250 Kbytes of debug files and only 20

Kbytes of user data. This means that only 12.5% of the communications are actually necessary. In the case of SUPREM II, the user has no control over the volume of the ancillary files generated, but this is not the situation for most of the commercial simulation packages now in use. Simulators such as SUPREM III & IV and PISCES all permit users to define what files are generated for a given job, and hence much of the inefficiency identified within the above results can be limited.

7.4 Conclusions

This chapter has presented the practical design of a transputer based task farm system, with particular application to semiconductor process simulations. The structure of each major software process has been discussed in turn, with a special emphasis being placed on the means by which every worker gains what it considers to be unique access to the host system functions. There is at least one AFS channel multiplexor available, provided with the 3L scientific language software packages [26], which permits each attached process to access the server. However, it does not provide the additional support required to alter filenames on-the-fly or operate with restricted access to the file system. Consequently, an entirely general multi-processing computing environment has been developed which enables large batches of independent or semi-independant programs to be efficiently executed with **no** alterations to the application software. In particular, it is this last feature which particularly enhances the attraction to its use. In support of this, the most suitable applications for execution under such a system, are usually highly numerically intensive and have a relatively low utilisation of I/O services in general. The majority of currently used software which falls into this category is written in one of the common high-level scientific languages for which transputer compilers already exist and so can be simply ported across to this environment. Thus a comprehensive selection of applications can be rapidly generated and made available.

This system is also extremely flexible in terms of the number of processors available for use at any one time. No alterations are required in order to remove or insert processors as circumstances dictate, since every time the system is booted

up, the routing process establishes the size and connectivity of the attached network. With respect to the hardware requirements, there is only one big difference between this approach and that suggested in chapter 5, where each instance of the application was partitioned across the multi-processor network. In most cases, this distribution of modules results in the data structures held within each module being diminished. However, in the case of the task farm, every processor contains the entire data set and thus the memory requirements do not reduce. When considering the simulation packages used within the semiconductor industry, this places fairly large demands on the available memory, with a minimum configuration being 4 or 8 Mbytes per transputer.

The results presented during the course of this chapter, confirm the benefits which the system can provide, especially when applied to the large batches of repetitive simulations demanded by the experimental design techniques discussed earlier. Although the system was hosted by a standalone IBM compatible PC for this work, there are many advantages in using a multi-user or networked host. In the first instance, multi-user operating systems generally permit more generous access to the file system, thus removing one of the main limitations of the present environment. However, in terms of future system development, the most flexible and beneficial operating environment would be provided by permanently running the task farm as an additional networked device, much like a printer or plotter, using the INMOS B407 ethernet interface card. In this way, simulations or any other suitable tasks can be spooled into a dynamic version of the present control file and multi-user access to the system has been achieved. The farm could then be interfaced either directly to the host or via an ethernet link enabling it to be operated remotely from the host.

Whether a single- or multi-user system is acquired, an economic solution to providing flexible, high performance facilities for computationally intensive tasks has been proposed. This will enable the full potential of DFM and other similar methods to be commercially applied to development and production problems industry-wide.

References

1. "Transputer Development and *iq* systems Databook", Inmos Ltd, Prentice Hall, 1989, p236.
2. "Transputer Development System", Inmos Ltd, Prentice Hall, 1988, pp375-424.
3. Old Server Manual - spiral bound from INMOS
4. "TITCH User' Guide", ...
5. "TINY Overview", ...
6. D. Prior, "Communication Harnesses", Edinburgh Concurrent Supercomputer Newsletter, No. 8., July 1989, p5.
7. L. J. Clarke, "Focus: Communications in networks of parallel processors", Edinburgh Concurrent Supercomputer Newsletter, No.7, April 1989, p18.
8. C. Dyson, "INMOS H1 architecture revealed", New Electronics on Campus, Spring 1991, p6.
9. Parallel C Users Manual, 3L Ltd, Nov. 1989.
10. "OCCAM 2 Reference Manual", Inmos Ltd, Prentice Hall, 1988.
11. D. Pountain, D. May, "A tutorial introduction to OCCAM programming", Blackwell Scientific Publications/McGraw-Hill.
12. S. Brown, "Communicating Sequential Tools", Edinburgh Concurrent Supercomputer Newsletter, No. 10, April 1990, pp6-7.
13. "The Helios Operating System", Perihelion Software Ltd, Prentice Hall, 1989.
14. S. Turner, "TAOS", Parallelogram International, No. 36, April 1991, p20.
15. L.J. Clarke, "Focus: Communications in networks of parallel processors", Edinburgh Concurrent Supercomputer Newsletter, No. 7, April 1989, pp14-15.
16. "Transputer Development and *iq* systems Databook", Inmos Ltd, Prentice Hall, 1989, p239.
17. "OCCAM Toolset - User Manual", Inmos Ltd, Bristol, 1987, pp88-99.
18. "File Server Protocol", Technical Note No.3, 3L Ltd, 1987.
19. "OCCAM Toolset - User Manual", Inmos Ltd, Bristol, 1987, pp100-103.

20. "Transputer Reference Manual", Inmos Ltd, Prentice Hall, 1988, pp6-7.

21. R. Causey, "The design behind the T9000 transputer", Electronics Weekly, April 24, 1991, p15.

22. "Developing Parallel Systems", 3L Parallel C Users Manual, 3L Ltd, Nov. 1989, pp39-67.

23. "Loading and Running Tasks from Parallel C", Technical Note No.5, 3L Ltd, August 1988.

24. "C Library Functions", Sun Operating Systems Manual, Version 4.1, January 1990.

25. "OCCAM 2 Reference Manual", Inmos Ltd, Prentice Hall, 1988, pp37-38.

26. "Task Data Sheets - Filemux", 3L Parallel C Users Manual, 3L Ltd, Nov. 1989, pp346-351.

When working towards the solution of a problem, it always helps if you know the answer.

Chapter 8

Conclusions

The concepts of *Design for Manufacture* (DFM) and *Design for Quality* (DFQ) are now recognised by the major semiconductor companies as vital to the success and survival of their IC manufacturing operations. The implementation of these methodologies has considerably altered the importance and the role of computer simulation techniques during the design of IC fabrication processes. Under such strategies, computer simulation would no longer be employed in an *ad hoc* manner, but would instead be coupled with statistical design and analysis techniques to provide a structured approach to the design of IC fabrication processes. It is therefore necessary that a system be developed which supports and promotes this concept. The work undertaken during this project has concentrated on the provision of an affordable and flexible simulation environment which has then been integrated with the requisite statistical tools to obtain an operational and functional design system. A brief synopsis of the work which was carried out is presented here and highlights some of the major issues that have arisen and their significance to the project as a whole. Some areas for future research efforts are then indicated to conclude this discussion.

The international market for semiconductors is highly competitive, volatile and fickle, which may be attributed to its demand-driven nature and the consequential susceptibility to influence from a variety of external forces. On account of this, manufacturers are being placed under increasing pressure to respond more rapidly to market demands. This can only be achieved by the adoption of flexible manufacturing technologies, in contrast to the more traditionally established production environment. The ability to switch between existing processes and introduce new ones at frequent intervals is essential for servicing the rising demand for low volume application specific IC (ASIC) production. Similarly, technological trends towards smaller device geometries, higher packing densities, greater yields

and shorter product lead-times place an enormous strain on equipment capability and process control. The overall consequence of these issues is that semiconductor manufacturers must expand their design criteria to include manufacturability as well as performance.

The fabrication of IC's is one of the most complex manufacturing processes currently performed, with a typical MOS process consisting of many hundreds of individual steps. Although there is only a limited set of basic process operations, 6 in total, it is the sheer number involved which, when coupled with their inter-dependencies, leads to the degree of complexity concerned. If IC fabrication processes are to be accurately controlled then it is essential that each operation is fully understood both in isolation and more vitally, with respect to its influence on the final product. The yield of any process, considered in terms of both product specification and functionality, is dependant on the control parameters associated with each operation as well as the accuracy of the processing equipment. Since the opportunity to perform in-line wafer testing in order to verify individual process operations is limited, the importance of process control cannot be over-emphasised. Chapter 2 provides a description of the six basic process operations related to IC fabrication, with a bias towards the physical parameters that are associated with their control and which may be altered by the process engineer.

Throughout the past two decades software models have been developed which describe the physical effects of performing a process step on a wafer. These models have been combined together to form complete process simulation packages that can be used to investigate the interactions that take place between process steps and calculate doping profiles, electrical characteristics and structural features of the fabricated devices. Physical process simulation has been primarily applied to date by process engineers in order to simplify the development and optimisation of fabrication processes by reducing the number of experimental runs required to meet a particular specification. Simulators are now available to perform one-, two- and three-dimensional process modelling. The models associated with one-dimensional simulation, which considers a section normal to the wafer surface, are

much simpler and more robust than their advanced two- and three-dimensional counterparts. However, once geometries approach micron and sub-micron levels, it is no longer possible to neglect effects relating to the second and third dimensions, which are in the plane of the wafer. Although these advanced models are less physically complex and robust than their predecessors, their enormous numerical complexity means that immense computational power is required to solve them. Chapter 3 first presents the one-dimensional numerical models that have been developed for each of the process steps, before providing a more general discussion of advanced process models and quantification of their respective computational requirements. Few semiconductor companies possess the necessary resources to support the widespread application of these simulators that is demanded by the advances in processing technology. Hence an investigation was made into alternative sources of economic computational power to act as a dedicated simulation engine.

The regular, grid-oriented structure which lies at the heart of all these simulators directed the research towards the promised power afforded by parallel computing technologies. A brief summary of both sequential and parallel computing paradigms introduces a more detailed view of the various parallel technologies available today that is provided in Chapter 4. Parallel technology extends from the traditional and prohibitive supercomputers, to a number of comparatively inexpensive desktop systems based around familiar microprocessor units. However, the relatively cheap, flexible and scalable computing power offered by the INMOS Transputer appeared to be the most suitable for this application. With this technology, sufficient computational resources could now be made available to the process engineers either as a large, dedicated simulation engine supported by the original host system or alternatively as a single extension board in each desktop PC or workstation.

Obviously, the simulation software requires to be adapted in order to take advantage of the parallel technology and the one-dimensional process simulator SUPREM-II, predecessor to the widely used SUPREM-3, was chosen as a demon-

stration. Numerically, SUPREM-II bears a close resemblance its successor which is important for the inference of results and although its physical capabilities are much less, this manifests itself more in the computational load rather than the choice of algorithms.

Ultimately, the overall objective of any parallel implementation is to increase the throughput of these advanced simulation tools. This objective may be realised by either reducing the cycle time associated with each simulation or by performing multiple simulations at once. The former approach has the advantage that the performance of every simulation will be improved, regardless of the number in total. Unfortunately, the penalty to be paid for this fully customised software is in the effort required to adapt the original code, which is highly manpower intensive and this obviously reflects in the overall project cost. Chapter 5 documents the work done in developing a parallel implementation of SUPREM-II to run on an arbitrarily sized transputer network. This task was far from straightforward to perform since, at that time, the software development tools available were not particularly comprehensive. This situation has been successfully remedied during the intervening years and there is now an extensive choice of development tools and environments. Other difficulties related to the partitioning of both code and data, inter-processor communications and substitution of algorithms were encountered, all of which are typical of custom developments. Once these obstacles were surmounted, an approximately linear increase in simulation performance was obtained as further processors were added to the system. This parallel methodology may well become essential for the excessively intensive computation that is expected of three-dimensional simulations or where the performance of a limited number of tasks is crucial. However, for most applications, the development time and cost will continue to outweigh the benefits returned until such time as intelligent parallel compilation can automatically perform this role.

Simulation has usually been employed to assist engineers establish the nominal process operating points required by their fabrication technologies. No conclusions can be drawn as to the suitability of this nominal operating point from

such a *design synthesis*. If, however, a design can be made insensitive to realistic processing perturbations, then additional savings can be obtained on unnecessarily expensive fabrication equipment, potential process control problems, lost yield and engineering time spent tuning the process. With simulation, all of this is achievable before the technology is ever manufactured. This *design analysis* role for simulation, which combines both engineering and statistical functions, plays a vital part in the overall concept of *designing for manufacturability*. Although the experimental design and statistical techniques described in chapter 6 minimise the number of simulations required to design a manufacturable process, the overall tally still makes a substantial demand on computational resources. Obviously, the additional simulation throughput afforded by a parallel implementation of the relevant software, as discussed in chapter 5, offers one solution.

However, due to the large number of independent simulation involved, an alternative strategy avails itself whereby each processor runs a single simulation autonomously. Chapter 7 describes the design and operation of such a transputer based simulation network, with particular emphasis on the supporting communications system and infrastructure. The advantages associated with this *taskfarm* approach to simulations include: no custom software development is required for each simulator, most programs can be compiled for the transputer, different software can run on each processor and both the architecture and performance scale in a simple linear fashion. In order to produce a usable engineering tool, an interface was developed between the statistical software packages, RS/DISCOVER and RS/EXPLORE, and the transputer taskfarm which manages the creation, execution and analysis of all the simulation data.

The software developed during this project significantly enhances the role of process simulation in the promotion of DFM and DFQ within the semiconductor industry. However, a great merit of the taskfarm system as it stands, is that it may be applied to *any* scenario, for which suitable software exists, without any alterations. It is also worthy to note that although a transputer based solution was

selected for this project, there are a number of other similar parallel or concurrent systems which are equally applicable and the final choice for a particular application will probably be dependent on existing technology.

The advances in simulation technology and systems described within this Thesis provide a significant basis for the support of the DFM and DFQ methodologies in the semiconductor fabrication industry. However, the infancy and enormity of this field mean that many issues remain unresearched. The most logical continuation of this work would be to expand the horizons of the taskfarmed design analysis to include parameter extraction, device and circuit simulation. This would require a more sophisticated management facility than presently exists, but would vastly extend design capabilities. Currently the inclusion of device simulation is being investigated in the Edinburgh Microfabrication Facility by G. J. Gaston. Previous work performed by A. J. MacDonald in this facility has investigated the integration of process simulation and the Computer Aided Manufacturing (CAM) system, COMETS(WORKSTREAM), though poor simulation performance prevented real-time control issues being tackled. A worthwhile supplement to this work, utilising the performance of customised parallel simulations, could now investigate the application of expert systems to the control of process simulation in providing on-line analysis and control of fabrication processes. Another area which might be investigated is the incorporation of process simulation into manufacturing simulation software, which is already used by the semiconductor industry. However, unlike the more traditional manufacturing scenarios for which it was originally designed, the process flow within IC fabrication bears a greater relationship to the process itself rather than the equipment or its layout. The integration of a process modelling capability in describing particular aspects of the manufacturing operation could vastly enhance the overall simulation capacity.

If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilisation.

Appendix A

Reprints of Published Papers

"The Implementation of Process Simulation on Transputers for the Production of ASIC's", Microelectronics '89, Brisbane, Australia, July 1989.

8th Australian Conference
on Microelectronics
Brisbane 12-14 July 1989

The Implementation of Process Simulation on Transputers for the Production of ASICs

W.J.C. ALEXANDER
Ph.D. Student, University of Edinburgh, U.K.
A.J. WALTON
Reader, University of Edinburgh, U.K.
J.M. ROBERSTON
Lothian Professor, Microelectonics, University of Edinburgh, U.K.
R.J. HOLWILL
Director, Edinburgh Microfabrication Facility, University of Edinburgh, U.K.

SUMMARY The role of process simulation in the semiconductor industry and the restrictions imposed on its use due to the computational requirements are discussed. The use of parallel computing techniques implemented on a transputer network is proposed as a solution, and results from a parallel version of the one-dimensional process simulator, SUPREM-II, are presented.

1. INTRODUCTION

The reduction in device geometries and the introduction of low-volume Application Specific Integrated Circuits (ASIC) is changing the role of process simulation within the semiconductor industry. To date, process simulation has largely been viewed as a tool for use within a research environment, and has therefore not been widely used by process engineers. However, with the current market climate dictating a fast, punctual and economic design turnaround, the margin of error has been reduced and the process must meet the full specification first time. To this end, process simulation can be employed to help ensure that these goals are achieved. Process modelling software has the major disadvantage that it is very CPU intensive and consequently only limited access has been available to process engineers. This paper details how parallel computing techniques can provide a cost effective process simulation system which will enable these tools to be used to their full potential.

Semiconductor process simulation has two principal roles within the fabrication and production environment. Firstly, it is an essential component in the design and development of a new process or optimisation of a current one. Process development involves the generation of a sequence of fabrication steps to produce a specified device structure, while still remaining within the constraints of the chosen technology and the available fabrication hardware. Traditionally this has been achieved empirically by performing sets of experiments and then following an iterative loop of measurement, modification and refabrication. This however, is a costly and inefficient approach since prototype production costs are high and may take up to eight weeks to perform. If process simulation is used for the initial iterations, then only a reduced prototype fabrication schedule is necessary, and this offers vast savings in both time and money.

Secondly, there is the ability to integrate process simulation with Computer Aided Manufacturing (CAM) systems to enable on-line analysis of any production batches held within the CAM database (MacDonald et al, 1989). Thus the potential exists to perform real time process monitoring together with the opportunity to provide corrective modifications for subsequent steps if poor or wrongly processed samples are detected. For this role in particular, it is paramount that software execution times remain low so as to produce a marginal computational overhead on the CAM host system.

The computational complexity of process simulation packages obviously depends on the sophistication of the incorporated models, and modern packages are now reaching the limits of acceptable CPU times. Although such simulations are still quicker than the equivalent experimental turnaround, this factor still discourages process engineers from using these tools to their full potential. One solution to this dilema is to draw on the resources made available by parallel computational techniques, and in particular those suitable for implementation on realistically priced hardware. The *transputer* fulfills these criteria and increased computing power can be provided by the simple addition of more transputers.

This paper first discusses process simulation, detailing the various models found within such packages and highlighting the balance that must be achieved between accuracy and speed. There follows an overview to the transputer concept and how it can be beneficially employed in improving the execution time of process simulation tasks. The objective of producing a parallel version of one presently available simulator is then explained, demonstrating some of the problems which have been encountered. Lastly, consideration will be given to the future potential of this concept particularly with relevance to the ASIC production cycle.

2. PROCESS SIMULATION

Process simulation programs contain a series of numeric, and in some cases analytic, models which attempt to describe the effect of each fabrication step on the dopant concentrations and characteristics of each material. They generally include modules to cover ion implantation, chemical predepostion, diffusion, oxidation and epitaxial growth, which are linked together by a controlling shell. The fundamental factor separating the available simulators is the provision of one or two-dimensional analysis, and to a lesser extent the number and types of layers that can be modelled. One-dimensional models analyse sections of the structure normal to the silicon surface, hence a number of these are required to fully model the device. The models in these simulators are generally more highly developed than those in their two-dimensional counterparts, and once calibrated provide an accurate picture of a given process. Figure 1 shows a section of an MOS transistor and the regions which must be simulated in 1-D to fully model the device. Figure 2 shows a 1-D section corresponding to one of these regions.

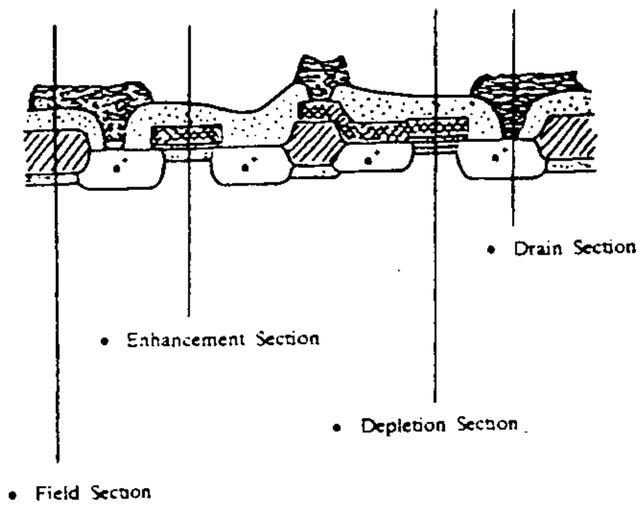


Figure 1 NMOS device structure plan, indicating all four sections requiring 1-D simulation.

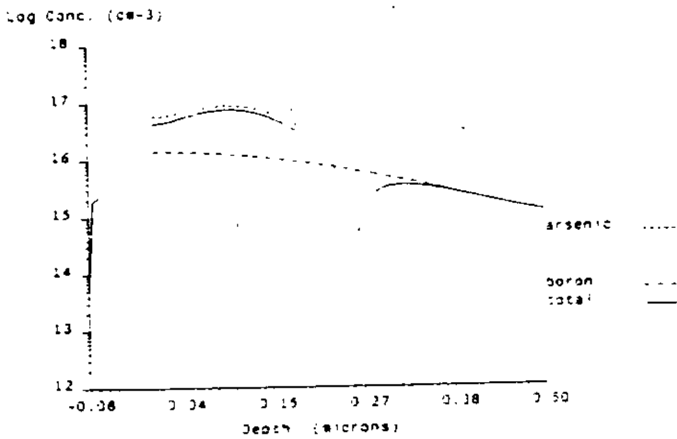


Figure 2 One-Dimensional profile of the depletion region from an NMOS inverter.

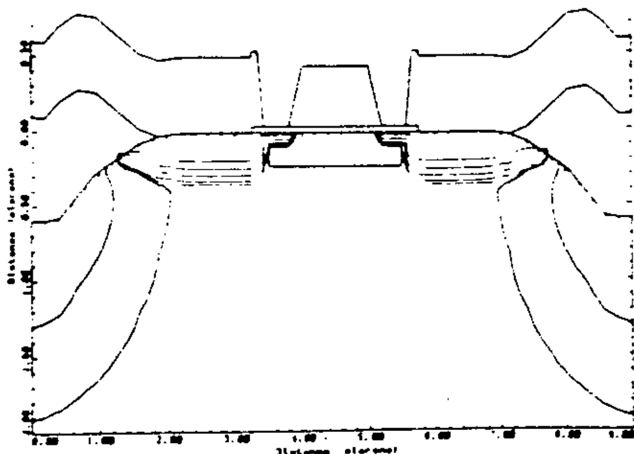


Figure 3 Two-dimensional MOS transistor contour plot showing the source, drain and field implant profiles.

When device geometries are relatively large (around 5 microns), one-dimensional models such as SUPREM-II, SUPREM-III and ICECREM provide sufficiently accurate data to understand the impurity behaviour. As feature sizes have reduced to 2 microns and below, short channel effects can no longer be ignored and the two-dimensional simulators such as SUPREM-IV, SUPRA and COMPOSITE are required. Figure 3 shows a 2-D profile for a typical MOS structure.

In progressing to 2-D simulation, grid resolutions and model complexity are traded-off to compensate for the increased computational requirement. Consequently to gain a more complete understanding of a small feature size device , it is often advantageous to also perform a series of 1-D simulations. As device geometries shrink still further, 3-D modelling will become necessary and run-times which already pose a problem in 2-D simulation (24 hours of CPU time (Singer, 1987)) will require even greater computational power.

If process simulation is to be used in a monitoring role, then the models must be calibrated to the particular process which they are to simulate.Once this is done, the process engineer will then have the confidence to use the software in a predictive role. Process simulation can also be used in a diagnostic role to 'look inside' the devices and observe parameters such as potential distributions and lateral impurity profiles. This capability often provides the solution to otherwise unexplained characteristics.

3. TRANSPUTER CONCEPT

The transputer provides a complete microprocessor computing surface including CPU, on-board RAM and four fast I/O links within a single chip. The general transputer architecture,is shown in figure 4 and covers a family of products with differing bus and memory specifications.

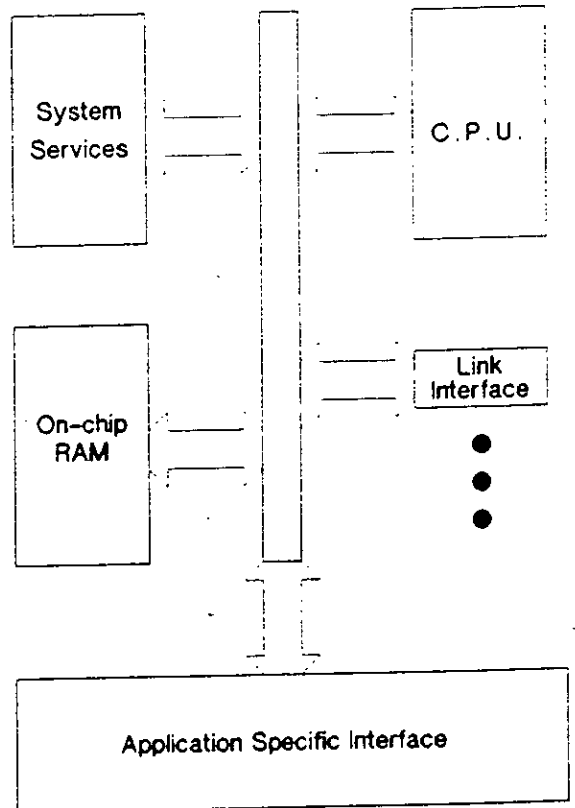


Figure 4 General transputer architecture.

Of particular interest here are the T414 and T800 32-bit devices, whose primary difference is the inclusion of a dedicated Floating Point Unit (FPU) on the T800 (Inmos, 1988). These may either be used as a single high performance processor system, or connected together using the links to form a point-to-point communicating network as seen in figure 5 with each machine *directly* linked to a maximum of four others.

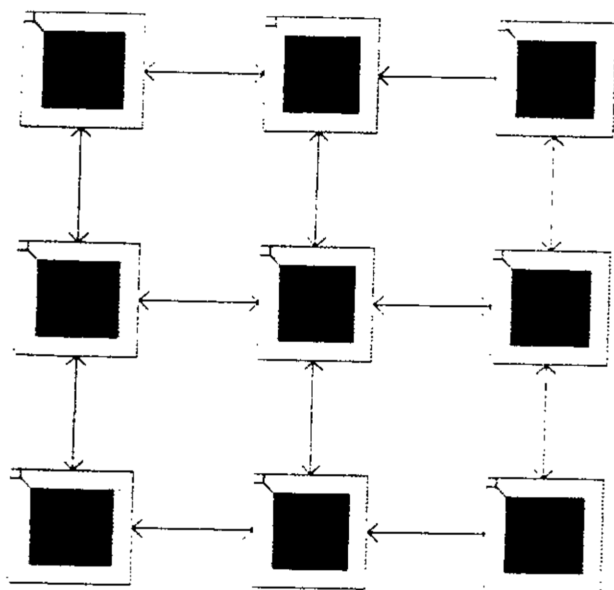


Figure 5 Arbitary transputer network showing the point-to-point communication links.

Multi-transputer systems possess many advantages over more traditional bus based multi-processor systems, reflecting the nature of these asynchronous serial links. Firstly, the point-to-point links eliminate all possibility of communication contention. Secondly, the memory and communicaion bandwidths are directly proportional to the number of processors in the system, meaning that the addition of further transputers to the system will not produce performance limiting bottle-necks. This last feature leads to the freedom to design arbitrary network architectures that optimise the data flow for a given problem, and to simply increase the computing power by adding extra transputers to the network.

The transputer family operates in a very flexible software environment, by being programmable in standard high-level languages such as FORTRAN, PASCAL or C. Concurrency can be achieved by the use of an OCCAM harness to link modules of code written in one or more of these languages, but to exploit the maximum potential the entire transputer system should be programmed in OCCAM.

4. PARALLEL PROGRAMMING

The concept of parallel processing is simple in theory, but much care and attention must be paid to the particular problem in order to achieve an efficient solution. It is quite possible for a multiprocessor solution to take longer to execute than on a uniprocessor if coordination of the individual processor tasks is not considered. Additionally, there are certain operations which are inherently sequential, such as disk and console I/O (custom graphics interfaces are of course an exception to this), and hence it is pointless attempting to parallelise them. Thus to map a problem onto a parallel architecture, the task must be divided into segments that will execute in parallel, and then determine the method of inter-processor communication (Howe and Moxon, 1987).

The distribution of these parallel tasks can heavily influence the efficiency of the scheme, so the degree of parallelism or *granularity* (Almasi, 1985) of the application must bear a direct relation to the communicative performance of the target hardware. The transputer network architecture shown earlier uses *message passing* for interprocessor communication, which means that any global broadcasting of information must involve the temporary synchronisation of all nodes. As a result local communication should be used since little processing overhead is required, making it very efficient.

The final issue in deciding how to coordinate the entire parallel scheme is whether there are any external constraints be applied to the system. If compatability with an existing code structure is required this may well force particular approaches to be taken which are not optimally suited to the overall parallel design. Depending on the magnitude of these constraints, it may prove beneficial to maintain the desired parallel format and incorporate additional pre- and post-processing elements to the system which provide interface compatability. Obviously the solution to this will be highly application specific.

5. PARSUP-II

SUPREM-II (Antoniadis and Dutton, 1979) was one of the early process simulation packages to be developed at Stanford University during the mid-seventies. It is a one-dimensional model capable of analysing all of the process steps mentioned earlier. Since its introduction 2-D simulators have generally become available, but the underlying algorithmic concepts have not changed significantly and the availability of the source code made it an obvious to choice for this project.

Referring the above analysis to the process modelling environment it becomes rapidly evident that the CPU intensive area is concerned with the diffusion of impurity atoms throughout the structure. Considering the problem in one dimension, it is found the diffusive flux of impurities, $F_D(x)$ can be formulated by a modified version of Fick's first law.

$$F_D(x) = -\frac{d}{dx}(D(x).C(x)) \quad (1)$$

where $D(x)$ is the diffusion coefficient of the impurity. If uniform diffusivity is assumed, then Fick's second law is obtained.

$$\frac{\partial C}{\partial t} = D\frac{\partial^2 C}{\partial x^2} \quad (2)$$

Although this law is adequate for low concentrations of impurity, the approximation fails for impurity concentrations at or above that of the intrinsic carrier (Dutton, 1983). Under these conditions the diffusion coefficient becomes a complex function involving impurity concentration dependancies. Thus it can clearly be seen that the formation and subsequent solution of this partial differential equation system becomes a complex and computationally intensive task.

At its simplest level, the solution of diffusion models is concerned with the generation and subsequent solution of the following matrix system.

$$\underline{A} \cdot \underline{x} = \underline{b} \quad (3)$$

where **A** represents the diffusion coefficients, **b**, the current impurity concentrations and **x**, the new concentrations after the process step.

Consequently, the following parallel methodology was chosen for distributing the diffusion process across the transputer network. The main bulk of the SUPREM-II code (written in FORTRAN) executes sequentially on a single *root* transputer, and when a diffusion step is encountered, the relevant data structures are downloaded into the network for the parallel stage to proceed. On completion of the diffusion calculation, these data structures are returned to the root processor which continues with any subsequent steps. This approach, represented in figure 6, ensures that investment is not wasted in sections of the software not suitable for parallelisation.

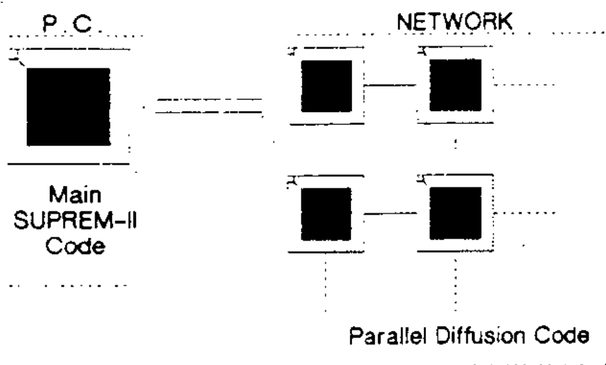


Figure 6 Distribution of the SUPREM-II code between the root and network processors

The partial differential equation can be formulated as a sparse matrix, so the first step was to select a suitable parallel matrix solution algorithm (Wang, 1981) and map this onto a network. A simple *pipeline* architecture was chosen for this task and the resulting execution times showed an almost linear improvement as the number of processors was increased. Figure 7 displays the performance attained in this application.

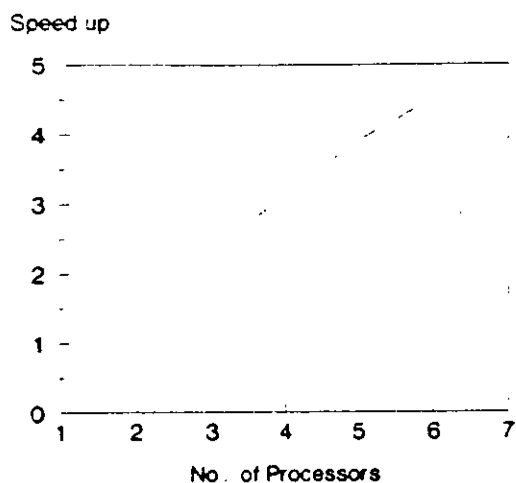


Figure 7 Pipeline solution speed up attained against the number of processors used. This is referenced to a sequential algorithm running on one processor.

A pipelined approach can maintain an efficient one-way data flow, but its performance degrades on the introduction of two-way flow if large amounts of data are being communicated. For real problems such as those found in SUPREM-II where relatively large data structures must be passed between the root processor and the target system, architectural modification is required to present an improved load/unload facility for the pipeline through the addition of intermediate *branches*. The resulting *spaceline* architecture is shown below in figure 8 using a tertiary tree approach as dictated by the limit of 4 links from any one transputer.

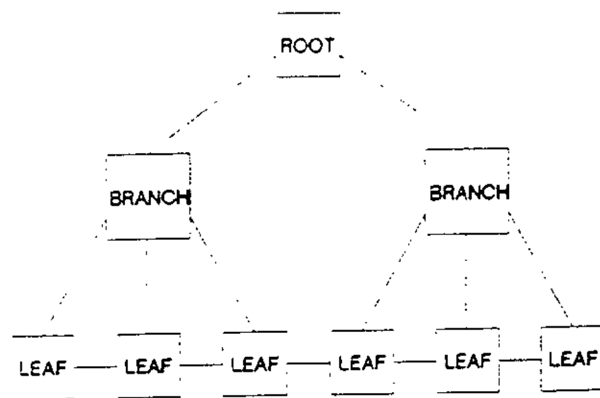


Figure 8 Spaceline network architecture

This architecture enables the entire SUPREM-II data structure to be distributed to each of the *leaf* processors which execute the parallel code, and returned to the global data structure on completion of the entire step in under 60 milliseconds*. Within each leaf node, a subset of the diffusion matrix system for a particular section is analysed, communicating with its neighbouring nodes whenever information regarding the boundaries is under consideration. Each leaf contains a complete and continually updated version of the global parameters required to control the conditional stages of the program.

6. APPLICATION OF PARSUP-II

So far the implementation of PARSUP-II has been discussed but no reference has been made as to its operating environment. One of the features of this system is its high degree of portability resulting from the substantial range of computers which support the root transputer. The host only provides file serving and console functions, with the entire processing being performed by the transputers. Hence, the requirement to have large mainframes to run this CPU intensive task no longer exists, and a Personal Computer (PC) of IBM-AT or similar specifications is more than capable of hosting such a system. Since many functions within the process engineers scope are already PC based, this makes it feasible for each engineer to have a personal and completely autonomous simulation system, without any of the drawbacks of the current multiuser approach.

* Typical execution times for a diffusion step are in the order of 10's of minutes on the T414 and 10's of seconds on the T800. A full process has approximately ten processing steps involving diffusion, so even small speed improvements lead to a significant reduction in the total execution time.

To place the performance of such a system in context, a PC with a single T800 transputer provides greater computing power than a VAX 11/780 system, at a fraction of the initial capital outlay and perennial maintainance costs.

The only criterion attached to the host choice is that a suitable transputer interface exists, but as these are available for PC's, VAX/VMS and VME based systems, a suitable host probably exist within most potential user environments. When considering the integration of process simulation with CAM to furnish the manufacturing system with real-time analysis, the transputer based approach has much to offer. Given the complexity of CAM tasks, there is a strong incentive to farm the simulation out to a network to be run in parallel with the ongoing CAM transactions, rather than impose further loading on the CAM host.

7. CONCLUSIONS

This paper has presented some of the reasons for the greater utilisation of process simulation within the semiconductor fabrication cycle, especially with respect to low-volume and multi-product production runs necessitated by the ASIC market. The drawbacks of current process simulation environments have been highlighted and a solution incorporating transputers has been proposed. This parallel process simulation concept has been demonstrated, indicating the improvement in run-times which can be achieved and although this has only been verified using a 1-D code, its potential in 2-D is self-evident. On the basis of these results, high perfromance and economic process simulation systems can be produced, enabling the full potential of this tool to be exploited in the semiconductor fabrication environment. Work is now in progress to extend this concept to 2-D process simulation software.

8. ACKNOWLEDGMENTS

The authors would like to thank INMOS (Bristol) Ltd, who have provided the transputer hardware and relevant software for this project. Additional support has been provided by the UK Science and Engineering Research Council (SERC). Sandy Alexander would also like to acknowledge financial assistance from SERC and INMOS.

9. REFERENCES

Almasi, G.S. (1985): Overview of parallel processing. Parallel Computing, Vol 2, pp191-203.

Antoniadis, D.A. and Dutton, R.W. (1979): Models for computer simulation of complete IC fabrication process. Electron Devices, Trans. IEEE, Vol ED-26, No. 4. April. pp490-500.

Dutton, R.W. (1983): Modeling of the silicon integrated-circuuit design and manufacture process. Electron Devices, Trans. IEEE, Vol ED-30, No. 9. Sept. pp 968-986.

Howe, C.J. and Moxon, B. (1987): How to program parallel processors. Spectrum, IEEE, Sept, pp36-41.

MacDonald, A.J. and Walton, A. J. and Robertson, J.M. and Holwill, R.J. (1989): Integrating computer aided manufacturing and process simulation for controlling the production of ASIC's. Proc. 8th Aust. Conf. on Microelectronics. To be published.

INMOS Ltd (1988): Transputer reference manual. Prentice Hall.

Singer, P.H. (1987): Process modelling and simulation. Semiconductor International, Feb, pp 74-80.

Wang, H.H. (1981): A parallel method for tridiaginal equations. Math. Software Trans. ACM, Vol 7 No. 2. June. pp170-183.

"Use of Process Simulators to Assist in the Design of Processes for Manufacturability", Conf. Proc. IEEE Advanced Semiconductor Manufacturing Conference, 1990.

USE OF PROCESS SIMULATORS TO ASSIST IN THE DESIGN OF PROCESSES FOR MANUFACTURABILITY

Michael R. Kump and Steve W. Mylroie
Technology Modeling Associates, 300 Hamilton Avenue, Palo Alto, CA

W.J.C. Alexander and A.J. Walton
Edinburgh Microfabrication Facility, University of Edinburgh, Edinburgh, Scotland

ABSTRACT

Increasing recognition of the role of random process variations in semiconductor manufacturing has led to wide use of statistical process control (SPC) techniques during the last decade. A companion to SPC that has seen much less acceptance to date is Design for Manufacturability (DFM). While SPC methods seek to control the variabilities in a process, DFM asks the question "How can the yield of a process be maximized, given the inherent variability in the underlying process steps?"

One of the factors limiting the use of DFM has been the large number of experiments required to characterize a process in terms of its sensitivities to random variations in its component steps. Linking process and device simulators with experimental design and analysis software such as RS/1 can greatly reduce the time and expense required to do a Response Surface Methodology (RSM) or Taguchi analysis of a process. Only a few split lot experimental runs are required for calibration purposes, with the majority of the experiments being done with simulators.

Applying these methods to several manufacturing problems has shown how minor process tuning can improve parametric yield and reduce variability with only a minor loss in peak device performance.

I. INTRODUCTION

Semiconductor companies are increasingly recognizing that to remain competitive, they must design new technologies not only for high performance, but also for optimal manufacturability. After all, a technology that has the utmost performance but cannot be manufactured with yield is of little value. Combined with this need for manufacturable processes is the exorbitant cost of new process development, which is forcing semiconductor companies to reevaluate the manner in which they develop new processes. Gordon Moore, Chairman of the Board of Intel, has estimated the cost of new process development at roughly $100 million, with the cost expected to increase in the future [1]. This paper will show how semiconductor process simulation, by replacing numerous run-splits, can be a cost-effective way to design new technologies for both optimal performance and manufacturability.

II. DFM CASE STUDIES

We will illustrate the use of process and device simulators for designing for manufacturability by considering two examples, both drawn from experiences in actual semiconductor manufacturing. The first is the design of a high sheet resistivity implanted resistor to minimize its variability. The second is the design of certain aspects of an LDD MOS process to both optimize the intrinsic device performance and to improve its manufacturability.

A. High-value Resistor Control

The objective of this study was to increase the percentage of die that qualified as high performance parts. This bipolar part was sorted into several product grades based on its switching speed and power consumption, with the market price of the product varying widely as a function of grade. The primary circuit design factor influencing the speed and power performance of the part was the bias current level of the devices on the chip. This current level was established by a high sheet resistivity implanted resistor.

In this manufacturing area the part was a medium volume product. Its volume did not provide sufficient data to allow the use of classical SPC techniques for yield improvement, nor could it justify a major yield enhancement project. However the lower yield of high performance parts represented a considerable revenue loss. During a four month period in which 17 runs of this part were processed, over 22% of the die produced were functional but unsaleable because either the switching speed or power consumption was outside specification limits. An additional four percent of the die where down-graded to lower priced parts due to marginal switching speed or power consumption. The net result was a 27% yield loss due to process varibility.

Initial yield improvement efforts had focused on improving the dimensional control in the masking process used to define the resistor. These efforts had little effect on the output product mix. As a result, an effort using process simulation and DFM techniques was undertaken to identify improvements in the process or in the process control limits that could improve the product's yield.

1. Process description

The resistor was fabricated by implanting a p-type layer into the n-type collector region of the circuit. Boron was implanted through the 320 nm of oxide grown during the previous base diffusion steps. Openings in a photoresiest layer delineated the resistor geometry. The implant was annealed during the subsequent emitter deposition and anneal furnace cycles. The SUPREM-3 process simulator [2] was used to simulate the resistor impurity profile, with a typical result shown in Figure 1.

2. Sensitivity Analysis

Based on an analysis of the historic yield data and the simulated impurity profile, the following five parameters were selected as the most likely sources of the observed variability in the resistor sheet resistivity:

- epitaxial layer thickness
- epitaxial layer resistivity
- oxide thickness
- resistor implant energy
- resistor implant dose

The process specificiations and equipment engineers were consulted to obtain estimates of the typical ranges of these variables. The values used in this analysis are listed in Table 1.

To limit the number of variables to be examined, the RS/Discover software [3] was used to design a fractional factorial screening experiment [4]. A recently announced TMA interface between the RS/1 series of software products and TMA's SUPREM-3 program automatically modified the process variables in a generic

parameter name	nominal	minimum	maximum
epi resistivity	$0.49\Omega/sq$	$0.44\Omega/sq$	$0.54\Omega/sq$
epi thickness	$4.2\mu m$	$3.85\mu m$	$4.55\mu m$
oxide thickness	3200Å	3000Å	3400Å
implant energy	135 KeV	115 Kev	125 KeV
implant dose	2.5×10^{13}	2.3×10^{13}	2.8×10^{13}

Table 1: Estimated variable control limits.

SUPREM-3 process simulation input file, ran the required simulations, and loaded the results into the RS/DISCOVER experimental worksheet. Once the generic SUPREM-3 input file was established, the "simulated experiments" with SUPREM-3 took less than an hour to run on a SUN SPARCstation 1 workstation. The experimental analysis tools in RS/EXPLORE were then used to fit linear response surfaces to the simulated sheet resistances and to calculate the sensitivity of the sheet resistivity to each parameter. Based on the results of this analysis, the epitaxial thickness was found to have the least effect on sheet resistivity and was discarded as a factor. RS/DISCOVER was then used to generate a central composite design for the experiments required to form a quadratic response surface for the four remaining factors. The TMA-RS/1 interface was again used to run the SUPREM-3 simulations and load the results into the worksheet. Finally, the multiple regression capabilities in RS/EXPLORE were used to fit a response surface to the simulation results. The response surface showed that the largest source of variability was the dose control in the implanter, producing a resistivity spread of $300\Omega/sq$ for the estimated variation in dose. The other three factors each contributed a variability of about $50\Omega/sq$ each (see Figure 2).

3. Process Centering

Once the response surface for sheet resistivity is fit, we can use it for design centering. Our goal is to achieve both a target resistivity of $1500\Omega/sq$ and to minimize variabilities, so we define a composite response as

$$CR = weight(RS - 1500)^2 + \left(\frac{\partial RS}{\partial ID}\right)^2 + \left(\frac{\partial RS}{\partial IE}\right)^2 + \left(\frac{\partial RS}{\partial OT}\right)^2 + \left(\frac{\partial RS}{\partial ER}\right)^2$$

where RS = sheet resistivity, ID = implant dose, IE = implant energy, OT = oxide thickness, and ER = epitaxial resistivity. *weight* is a weighting factor specifying the relative importance of attaining the target resistivity and minimizing the variabilities. (The value used for this study was 10.) The oxidation and diffusion steps

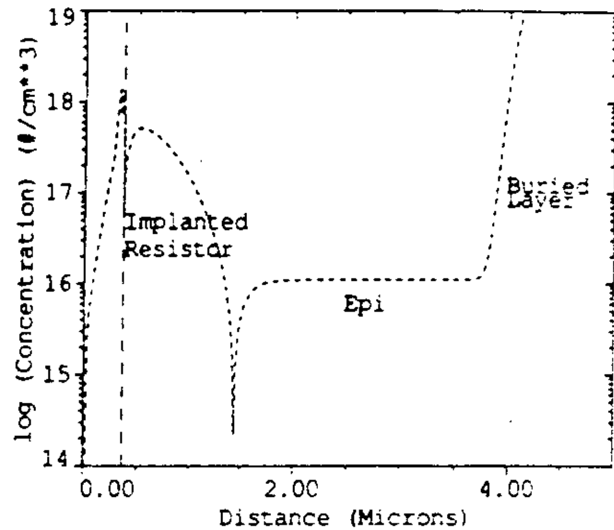


Figure 1: Resistor impurity profile predicted by SUPREM-3

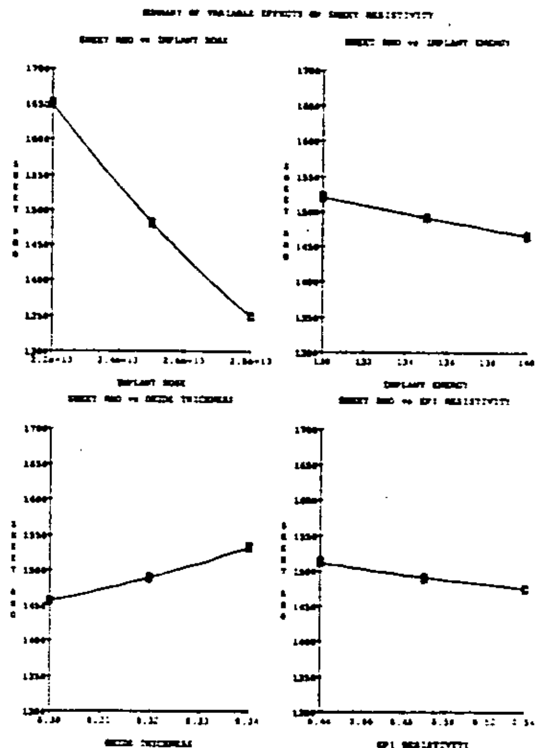


Figure 2: Sensitivity of sheet resistivity to each process factor.

used to fabricate this part are shared with a number of other parts manufactured in this fabrication area, whereas the implant step is unique to this product. In addition, changes in the implant would only affect the resistor, while changes in the oxidation or diffusion steps would affect all the devices in the circuit. For these reasons, the parameters to be varied during the optimization were limited to the implant dose and energy.

The optimization algorithm in RS/EXPLORE was used to minimize this function, subject to the constraint that the deviation of the resistor sheet resistance from 1500Ω/sq should be less than 90Ω/sq. The result showed that the current implant conditions, while not optimal, were not far from optimal. The most significant improvement that could be expected was a reduction of the spread in the mean sheet resistance from 420Ω/sq to 380Ω/sq.

4. Feed-Forward Process Control

Since the epitaxial resistivity and oxide thickness are known from in-process measurements before the resistor implant step is undertaken, the possibility exists to use an optimization based on the response surface to adjust the implant parameters to minimize process variation. Optimization showed that the variation in the mean value of the sheet resistivity could be reduced from 130Ω/sq to 55Ω/sq if the adjustment were based on the measured epitaxial resistivity alone and to 30Ω/sq if both the epitaxial resistivity and oxide thickness were used to establish the implant dose.

5. Conclusion

By using process simulation coupled with statistical design of experiment tools, the dominant factors causing sheet resistance variation were identified in a few hours without any silicon runs. Improved dose control in the implanter was identified as the factor that would yield the greatest improvement in process control. A minor decrease in variability was found from process centering, but a very significant reduction was possible if the response surface was used in a feed-forward manner based upon in-line epitaxial resistivity and oxide thickness measurements.

B. MOS LDD Design

The second case study is that of optimizing certain aspects of an n-channel LDD MOS device. A partial list of the process variables that can be varied to optimize device performance might be

- drawn gate length
- gate polysilicon thickness
- ⇒ gate polysilicon edge slope (θ)
- ⇒ LDD implant dose (D_{LDD})
- ⇒ LDD implant energy (E_{LDD})
- deposited oxide spacer thickness
- etched oxide thickness
- source/drain implant dose and energy
- drive-in time and temperature

We could create an equally long list of device characteristics that we would like to simultaneously optimize, a subset being

- ⇒ minimize series resistance (R_s)
- ⇒ minimize peak electric field (E_{peak})
- ⇒ maximize transconductance (g_m)
- ⇒ ensure punchthrough voltage (V_{pt}) > 10 volts
- maintain specified threshold voltage
- maximize subthreshold slope
- minimize junction and overlap capacitances
- ⇒ minimize all sensitivities ($\frac{\partial R_s}{\partial \theta}, \frac{\partial R_s}{\partial D_{LDD}}, \frac{\partial E_{peak}}{\partial \theta}, \ldots$)

Some process variables are *local* and will have a straightforward influence on only a single device characteristic. For example, increasing the dose of a source/drain implant in this LDD process will primarily simply decrease the series resistance of the device. Other process variables are *global* and will influence numerous device characteristics, and changes in one of these process variables may interact with others in a complex way. It is for these complex interactions in which the designer has to juggle many trade-offs that DFM using response surface methodology (RSM) is most useful.

In particular, the process variables considered for this case study are those indicated with arrows in the previous list of variables. The first variable is the slope of the polysilicon sidewall. This slope has an important influence on a number of device characteristics, and manufacturing engineers had indicated that sidewall slope was not controlled as well as they would like. The other two variables are the dose and energy of the LDD implant. These variables will interact with the poly sidewall slope and will affect the grading of the source/drain region and the effective channel length.

For this study, the responses that were of most interest were those indicated with arrows in the list of responses. The problem is to select the optimal set of the three process variables that simultaneously minimize R_s, minimize the peak electric field E_{peak} (so as to minimize hot electron injection), maximize g_m, ensure V_{pt} greater than 10 volts, and also, to improve manufacturability, minimize the sensitivities of all responses to all variables. Even this subset of the design problem would be quite formidable without a systematic approach. The myriad trade-offs in optimizing the design can be virtually impossible for the designer to keep in mind simultaneously.

1. Simulation Procedure

This process was simulated with the TSUPREM-4 [5] process simulator, which produces a two-dimensional cross-section of the device. The electrical performance of the device was then analyzed with the PISCES-2B [6] device simulator. Punchthrough voltage (V_{pt}) was determined by applying 0 volts to the gate and ramping the drain up until 1 nA of current flowed. The peak electric field (E_{peak}) was found with 5 volts on the drain and 3 volts on the gate, a bias condition that will likely be near the peak hot electron injection conditions. Finally, the gate I-V characteristics for different channel length devices were generated. These characteristics were fed into the TOPEX [7] parameter extraction program to extract series resistance (R_s) and peak transconductance (g_m).

Figures 3 and 4 are the structures from TSUPREM-4 corresponding to the two extremes of polysilicon sidewall angle that were investigated. With a polysilicon angle of 70° we have a wide spacer, and the LDD implant does not even extend to the gate edge, so the series resistance will be high in this case. At the other extreme, with a sidewall slope of 110°, we have a thin oxide spacer and good penetration of the implant under the gate. So, series resistance will be lower, but the peak electric field will be higher in this case. The optimal polysilicon slope will likely lie between these two extremes.

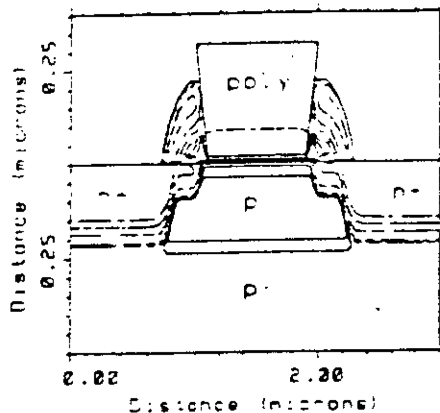


Figure 3: TSUPREM-4 simulation with 70° polysilicon edge slope. (Slope is measured from the horizontal axis.)

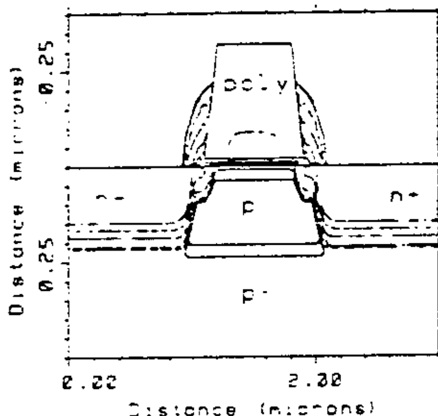


Figure 4: TSUPREM-4 simulation with 110° polysilicon edge slope.

2. Single Factor Analysis

The most obvious way to start optimizing this device structure would be termed "single factor analysis," in which we look at the influence of one design variable at a time, holding the remaining variables constant. This is the appropriate approach when we are just starting the design—to ensure that the values of the design variables are at least of the correct order of magnitude. So it is the convenient approach for an initial, rough design.

With single factor analysis we can roughly center the design about an optimum device and determine reasonable variable ranges in which a more sophisticated search for an optimal design should be conducted. Also, response surface methodology will involve fitting a polynomial model to the responses, so at this stage we can judge roughly what order of polynomial will be sufficient for the model. Typically a quadratic model is used, but a higher order model could be used if necessary.

Figure 5 shows single factor plots of R_s for the device under consideration. R_s is plotted versus sidewall slope with LDD implant energy as a parameter. We can see that R_s decreases rather sharply as a function of slope. If our design criterion is to have R_s less than 500Ω, we can see what combinations of design variables would give us that result, and we can see which variables most strongly affect R_s.

However, the disadvantages of this approach are numerous:

- First, it is awkward to treat multiple process variables. Figure 5 is only one plane through this 3D parameter space, so much of the behavior cannot be seen. For example, this plot is for an LDD dose of $5 \times 10^{12}\,\mathrm{cm}^{-2}$. The behavior might be quite different at a dose of $10^{13}\,\mathrm{cm}^{-2}$, but we must perform another set of simulations and another graph to see that. To investigate all possible combinations of the three process variables in this study would require a full factorial number of simulations and a great number of plots. The situation might be acceptable for three variables, but it quickly becomes unacceptable if we want to consider five or six process variables.
- Secondly, we do not know which combination of process variables is the optimal one for simultaneously optimizing the other responses. We could plot on top of Figure 5 the curves for g_m, E_{peak}, and V_{pt}, but the number of curves grows rapidly, and quickly becomes more confusing than informative.
- Finally, if we want to minimize sensitivities as well as optimize the device performance, we have no quantitative way of doing that—we can only look at the slopes of the numerous curves and somehow judge which set of process variables is best.

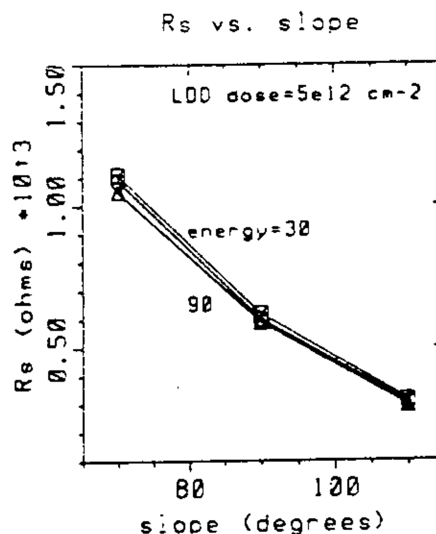


Figure 5: Single factor plots of series resistance.

3. Response Surface Methodology

A solution to this difficulty is the use of response surface methodology, which is a well-known approach to efficient experiment design and analysis that is increasingly being used in semiconductor process design. The novel aspect about the present approach is that experiment design methods are applied to efficiently analyze simulated experiments rather than actual fab line experiments. But the goal is the same—to obtain maximum information from a minimum number of simulations and then to optimize the performance of the process.

RSM analysis provides a structured, systematic approach to multi-parameter/multi-response optimization. It is no longer necessary for the designer to juggle all the trade-offs in his head or work with myriad 1D plots. This method easily handles a large

number of process variables and their interactions. We are only considering 3 process variables in this case, but we could easily treat twice that number.

4. Simulation Point Selection

The next step is to pick the points for our RSM analysis. The brute force approach is to do a full factorial experiment, which, for a quadratic function in 3 variables would require 27 simulation runs. This may be acceptable for 3 variables, but for 6 variables the number of simulations is 729—clearly unacceptable. A more efficient approach is the central composite design, which is one of the most common designs used for fitting RSM surfaces. For fitting a quadratic in each dimension of our 3 variable space, the central composite design consists of 1 center point, 6 axial points, and 8 so-called "factorial" points, where the axes of our 3D parameter space are poly sidewall slope, LDD dose, and LDD energy. We already had the axial and center points available from our single factor analysis, so we had only to add the factorial points to complete the design. From the results of the single factor analysis, we picked ranges for the variables in which we thought an optimal design would exist.

5. Response Function Fitting

After performing the simulations at the RSM points, the next step is to fit a multi-variable polynomial to each response. The polynomial has the form

$$\begin{aligned} \text{response} &= a_0 + a_1\theta + a_2 D_{LDD} + a_3 E_{LDD} + \\ &\quad a_{11}\theta^2 + a_{22} D_{LDD}^2 + a_{33} E_{LDD}^2 + \\ &\quad a_{12}\theta D_{LDD} + a_{13}\theta E_{LDD} + a_{23} D_{LDD} E_{LDD} \end{aligned}$$

We fit a separate polynomial for each response, in our case, for R_s, E_{peak}, peak g_m, and V_{pt}. TOPEX was used to determine the optimum values of a_0 through a_{23} that best fit the polynomial to the simulated responses.

6. Composite Response

The power of the RSM approach lies in the fact that since we now have a mathematical description of our responses, we can combine them to find a composite response, or overall figure of merit, which we can optimize.

The composite response (CR_1) is constructed such that the optimal design will be at its minimum. It is given by

$$\begin{aligned} CR_1 &= (\text{weight}_{R_s})R_s + (\text{weight}_{E_{peak}})E_{peak} + \\ &\quad (\text{weight}_{g_m})\left(\frac{1}{g_m}\right) + (\text{weight}_{V_{pt}})\left(\frac{1}{V_{pt}}\right) \end{aligned}$$

All responses are normalized by their value at the center point of the parameter space, so units and relative magnitudes are factored out. The weights can then be selected solely on the basis of which response is most important to optimize for our particular application.

The composite response function is shown in Figure 6. Our optimal operating condition will be at the minimum of this composite response, which is again found with TOPEX.

The trade-off, primarily between R_s and E_{peak}, has left a valley near the middle of the design space. At the minimum, the optimal process variables and corresponding optimal device characteristics are

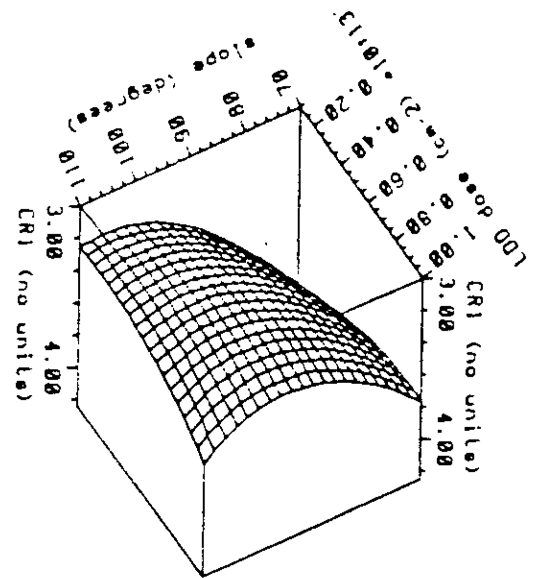


Figure 6: Composite response surface.

optimal process variables:	optimal responses:
$\theta = 104°$	$R_s = 352\Omega$
$D_{LDD} = 3.5 \times 10^{12}$ cm^{-2}	$E_{peak} = 5.08 \times 10^5$V/cm
$E_{LDD} = 65$ KeV	$g_m = 9.68\mu$S/μm
	$V_{pt} = 12.6$ V

In a sense, selecting the optimal device design is now complete. However, we should also place constraints on specific output characteristics, depending upon the constraints imposed by the circuit application for this device.

7. Feasible Operating Region

For our particular application, the circuit design places the following constraints on the device characteristics: $R_s \leq 500$ ohms, $E_{peak} \leq 6 \times 10^5$ V/cm, $g_m \geq 8\mu$S/μm, and $V_{pt} \geq 10$ V. These constraints form a *feasible operating region* in which the design must be centered (see Figure 7). This feasible operating region is found by plotting the contours corresponding to each constraint on top of the composite response and noting on which side of the contour we should operate. TOPEX can then search only within this constrained region for its minimum.

In this case, the minimum is well within the feasible operating region, so the operating point did not have to be moved to abide by the constraints.

8. Safe Operating Region

Now, to ensure that this technology will be manufacturable, we want to not only operate within the feasible region, but also within some margins inside the feasible operating region. This defines what might be called a *safe operating region*, which is the region defined by moving in 3σ in each of the parameter directions from the edges of the feasible operating region. From experience with the particular manufacturing facility in which this device is produced, we have estimates for the standard deviations of each process variable. Sidewall slope is a likely control problem, with a rather large standard deviation of 2.5°. The implant is better controlled, with dose controllable to within 5% of its nominal value and energy to within 1%. These standard deviations define a rather narrow safe operating region, as indicated in Figure 8. If we center our design within this region, we will have a high degree of confi-

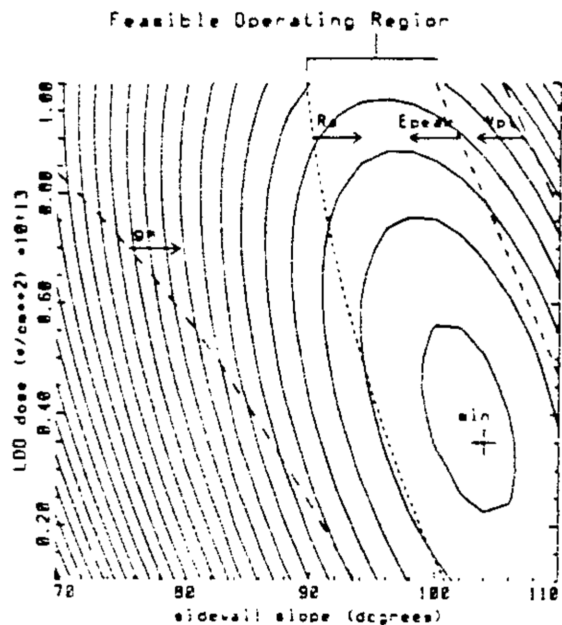


Figure 7: Feasible operating region. The optimal operating point, indicated by the cross, is not affected by the circuit constraints.

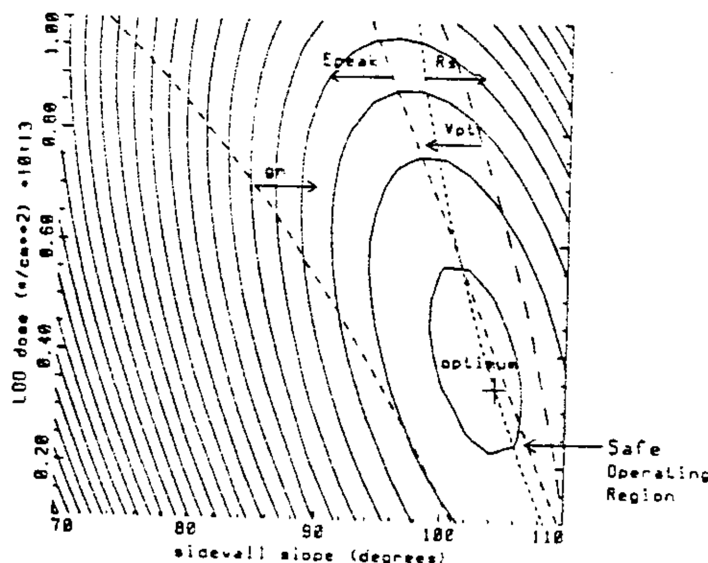


Figure 8: Safe operating region. The optimal operating point that is safely within the circuit constraints is indicated with the cross.

dence that for all manufacturing variations, the circuit requirements will be met. The optimal process variables and corresponding device characteristics that are safely within the circuit constraints are

optimal process variables:	optimal responses:
$\theta = 104.3°$	$R_s = 348\Omega$
$D_{LDD} = 3.4 \times 10^{12}$ cm^{-2}	$E_{peak} = 5.10 \times 10^5$V/cm
$E_{LDD} = 65$ KeV	$g_m = 9.69\mu S/\mu m$
	$V_{pt} = 12.6$ V

At least in this case we have a safe operating region. If the standard deviations had been slightly larger, we would have had no safe operating region, which would have been a signal that we may have manufacturing problems. In that case, we had better either reconsider building our device with this process, or we should investigate making whatever changes are necessary to reduce the standard deviations. With this procedure we have a clear indication which standard deviations are the cause of the control problem and a quantitative measure of how much they need to be reduced to make the technology manufacturable.

9. Composite Response with Sensitivities

Now, to improve the manufacturability of the technology, we can also recenter the design to make the technology less sensitive to manufacturing fluctuations. We can do that simply by including a measure of the variability into the composite response. We have defined a second composite response, CR_2, which is equal to the original composite response, CR_1, with the addition of a weighting factor times the product of the gradient of the composite response and the variances for our manufacturing line

$$CR_2 = CR_1 + weight_{sens}[\bar{\nabla}CR_1^T(\text{variances})\bar{\nabla}CR_1]$$

The gradient of the composite response ($\bar{\nabla}CR_1$) is the vector of partial derivatives with respect to each process variable, and (variances) is a matrix with the process variances on the diagonal and cross-correlations off the diagonal. If we now operate at the minimum of this new composite response, but still within the circuit constraints (Figure 9), we will be at a point with reduced variabilities. We will be sacrificing some device performance for this decreased variability. The process settings and corresponding device characteristics with decreased sensitivity are

optimal process variables:	optimal responses:
$\theta = 106°$	$R_s = 362\Omega$
$D_{LDD} = 2.3 \times 10^{12}$ cm^{-2}	$E_{peak} = 5.12 \times 10^5$V/cm
$E_{LDD} = 65$ KeV	$g_m = 9.71\mu S/\mu m$
	$V_{pt} = 12.3$ V

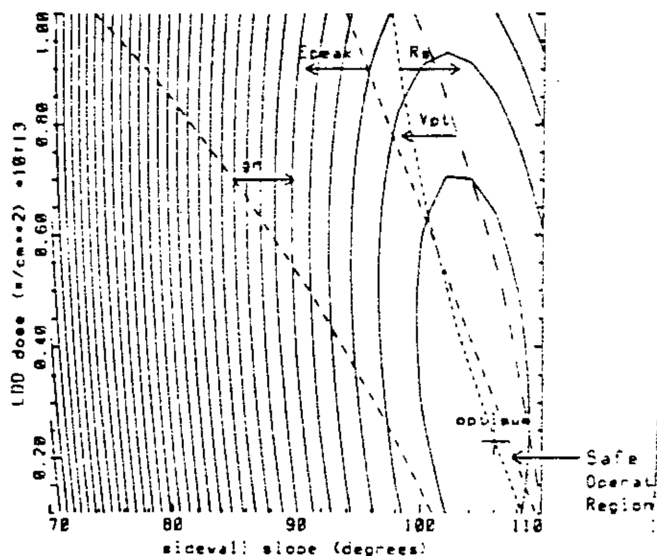


Figure 9: Composite response including variability estimate. The optimal operating point with reduced sensitivity is indicated by the cross.

III. SUMMARY

Simulation can be used to evaluate the manufacturability of a new technology before it ever enters manufacturing. So we can know before we make a large investment in fab equipment whether a proposed technology is feasible. Also, we can know where the control problems will be and where we should invest our resources to make the technology more manufacturable. Process and device simulation, combined with RSM analysis, provides an ideal way of making these evaluations.

Also, after a technology is in manufacturing, the case studies in this paper have shown that process and device simulation, combined with RSM analysis, can help diagnose control problems and can improve manufacturability with a minimum of actual fab line experiments.

References

[1] Gordon Moore, in an address to the 1990 Symposium on VLSI Technology, Honolulu, Hawaii, June, 1990.

[2] *TMA SUPREM-3 User's Manual*, Technology Modeling Associates, Palo Alto, California.

[3] *RS/1, RS/DISCOVER, and RS/EXPLORE User's Guides*, BBN Software Products Corporation, Cambridge, Massachusetts.

[4] G.E.P. Box, W.G. Hunter, and J.S. Hunter, *Statistics for Experimenters*, New York: John Wiley, 1978.

[5] *TSUPREM-4 User's Manual*, Technology Modeling Associates, Palo Alto, California.

[6] *TMA PISCES-2B User's Manual*, Technology Modeling Associates, Palo Alto, California.

[7] *TOPEX User's Manual*, Technology Modeling Associates, Palo Alto, California.

RS/1, RS/DISCOVER, and RS/EXPLORE are trademarks of BBN Software Products Corporation.
SUPREM and PISCES are registered trademarks of the Board of Trustees of the Leland Stanford Junior University.
TOPEX is a trademark of Technology Modeling Associates, Inc.

"PARTICS: A Parallel Taskfarm for Integrated Circuit Simulation", Edinburgh Parallel Computing Centre Technical Report, EPCC-TR91-08, 1991.

PARTICS : A PARallel Taskfarm for Integrated Circuit Simulators

G.J. Gaston, W.J.C. Alexander, L.J. Clarke † and A.J. Walton.

Edinburgh Microfabrication Facility, Department of Electrical Engineering, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

† Edinburgh Parallel Computing Centre, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

ABSTRACT

Statistical methods used in the manufacture of integrated circuits are discussed. A parallel taskfarm system, PARTICS, is outlined. It is shown how this system can easily and effectively be used to reduce the time taken for performing CPU intensive process and device simulations. The system gives an almost linear speed up is inherently flexible, and can be used with any application. Two different hardware platforms, an IBM PC compatible and a MEiKO computing surface are presented.

1. INTRODUCTION

As a result of the trends towards greater complexity and miniaturisation of VLSI devices, modern Integrated Circuits (IC's) are becoming increasingly susceptible to fluctuations in the manufacturing process. It is such complexities that have heralded the use of Computer-Aided Design (CAD) tools for process and device simulation. From quiet beginnings these tools are now widely used in the semiconductor industry.

The major advantage of these simulation programs is that costly prototype production runs, which can take months to complete, can be replaced by simulated experiments. These simulations can be completed in a matter of hours or days, depending on the processing capability of the computer and the type of simulation involved. As a result the length of the product development cycle is kept to a minimum, a critical aim of all semiconductor maufacturers.

Statistical methods, known as design centering or response surface methodology, are used to reduce the number of simulations required, in order to arrive at optimum settings for the manufacturing equipment [1,2]. Thus a set of input factors are varied and their effect on some output responses are analysed. Using RS/1 [1] a set of experiments is designed, containing different settings for each control factor. There may be upwards of 50 separate simulations that need to be carried out to determine the optimum setting, depending on the number of factors to be varied.

Although simulations reduce the time greatly, when compared to prototype runs, they can still be very CPU intensive and each simulation may take hours to

[1] A statistical package from BBN Software

complete. Since there are say, 50 different simulations to be executed, this lends itself greatly to a parallel taskfarm approach on transputers. Each transputer, called a slave or worker, in an array receives a task from a master transputer or driver and on completion the slave then receives the next job information.

2. SIMULATION

2.1. Process Simulation

As a result of the need to reduce the costs of prototype production runs, process simulation has become an essential tool in the process engineer's tool kit. Process simulation is particularly important in the evaluation of new structures or in the improvement of existing structures. These improvements may include changes for increased manufacturability.

Such programs can be divided into 1-D and 2-D process models. The 1-D programs such as SUPREM3 [3], have very sophisticated models for the different procedures in the manufacturing process. Accurate results can thus be obtained, but only a 1-D section through the structure can be simulated.

The 2-D programs such as SUPREM4 [4], SUPRA [5], are essential for analysing small geometry processes, where 2-D effects become more important. These simulations are, however, more CPU intensive.

The structure and doping profile calculated by process simulation, provide essential information required by device simulation, in order that accurate device characteristics can be simulated.

2.1. Device Simulation

The results of the device simulator are obviously profoundly affected by the input data generated by the process simulator. As the name suggests, device simulation is concerned with the device characteristics of the final product, eg I-V curves. Essentially, device simulation involves the solving of the basic semiconductor equations, which govern the static and dynamic behaviour of carriers in semiconductors under the influence of external fields [6]. There are a large number of device simulators available at present [7-9].

The output of the device simulator, in the form of I-V curves, enables circuit parameters to be extracted for use with circuit simulation programs. Such parameters include, threshold voltage, series resistance or transconductance.

Fig 1 shows the relationship between process, device and circuit simulation programs.

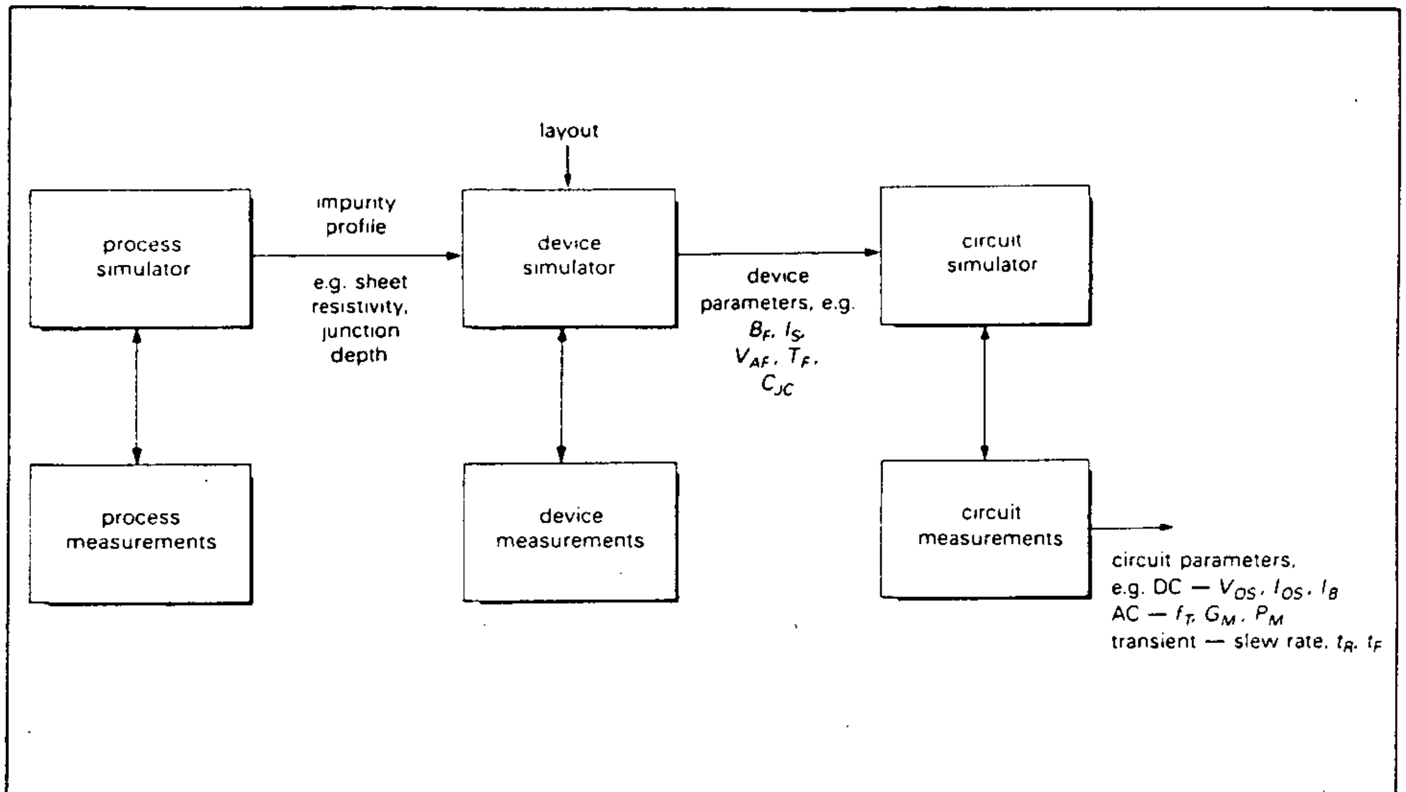


Figure 1: Relationship between process, device and circuit simulation

3. RESPONSE SURFACE METHODOLOGY AND EXPERIMENTAL DESIGN

Response Surface Methodology (RSM) techniques have been employed in order to reduce the number of simulations that are required to give the necessary information about a particular process.

As Box states, these techniques are essentially addressing the following questions [1]:

- How is a particular response affected by a given set of input variables over some specified region of interest?
- What settings, if any, of the inputs will give a product simultaneously satisfying desired specifications?
- What values of the inputs will yield a maximum for a specific response, and what is the response surface like close to this maximum?

The program RS/1, is used for setting up the experimental design. It enables the user not only to specify the required factors and responses to investigate, but also the ability to define a number of different attributes for each factor, eg

tolerance, settings, precision. It is also possible to model several responses in one experiment.

Depending on the choice of design, it is possible to greatly reduce the number of simulations that need to be executed. The design types that can be used in RS/1 include Full-Factorial, Box-Behnken and Central Composite design. Utilising RS/1 one can, therefore, use a planned program of experimentation (simulation) to obtain a complete response surface for all combinations of input factors from a reduced number of runs. An unstructured approach will not only be significantly heavier on CPU time, but also will not necessarily point to the optimum processing conditions.

RSM can be used to fit response surfaces, to analyse and interpret the results obtained from the simulations. The data may be visualised using graphical techniques, analysis of variance performed and models fitted to the results. Routines can be written using RPL, the RS/1 programming language, to enable quick and efficient transfer of files, tables etc. The RPL can thus be used as an interface between RS/1 and the simulation results.

Figure 2 shows a very simple response surface plot, showing the variation of theshold voltage and oxidation against dose and oxidation time.

4. SOFTWARE SYSTEM

Figure 3 shows how an the software system is used to automate the procedure. This is essential if human errors are to be avoided and the elapsed time for the procedure minimised. In the case of the example shown in Figure 2, 31 different control factor settings were generated using the experimental design capability of RS/1. This obviously requires 31 different simulations to be performed and the software automatically reads the factor settings from RS/1 and generates 31 different input data files for the process simulator, with the appropriate control factor setting. These 31 files are taken and farmed out on the parallel transputer system, using PARTICS.

4.1. Taskfarm

The taskfarm has been implemented on two different hardware platforms, namely an IBM PC compatible and the Edinburgh Parallel Computing Centre's MEiKO Computing Surface. The former system consists of an array of 5 T414 transputers, each with 2 MBytes of RAM, or 8 T800 transputers, each with 256 KBytes of RAM. For a modest sum, therefore, supercomputing power is available on the desktop. An array of 17 T800 tranputers, with 4 MBytes of RAM, was used on the MEiKO Computing Surface, although the total system has over 400 T800's available.

The taskfarm approach to parallel processing is often referred to as event parallelism. The configuration consists of a master processor which allocates the

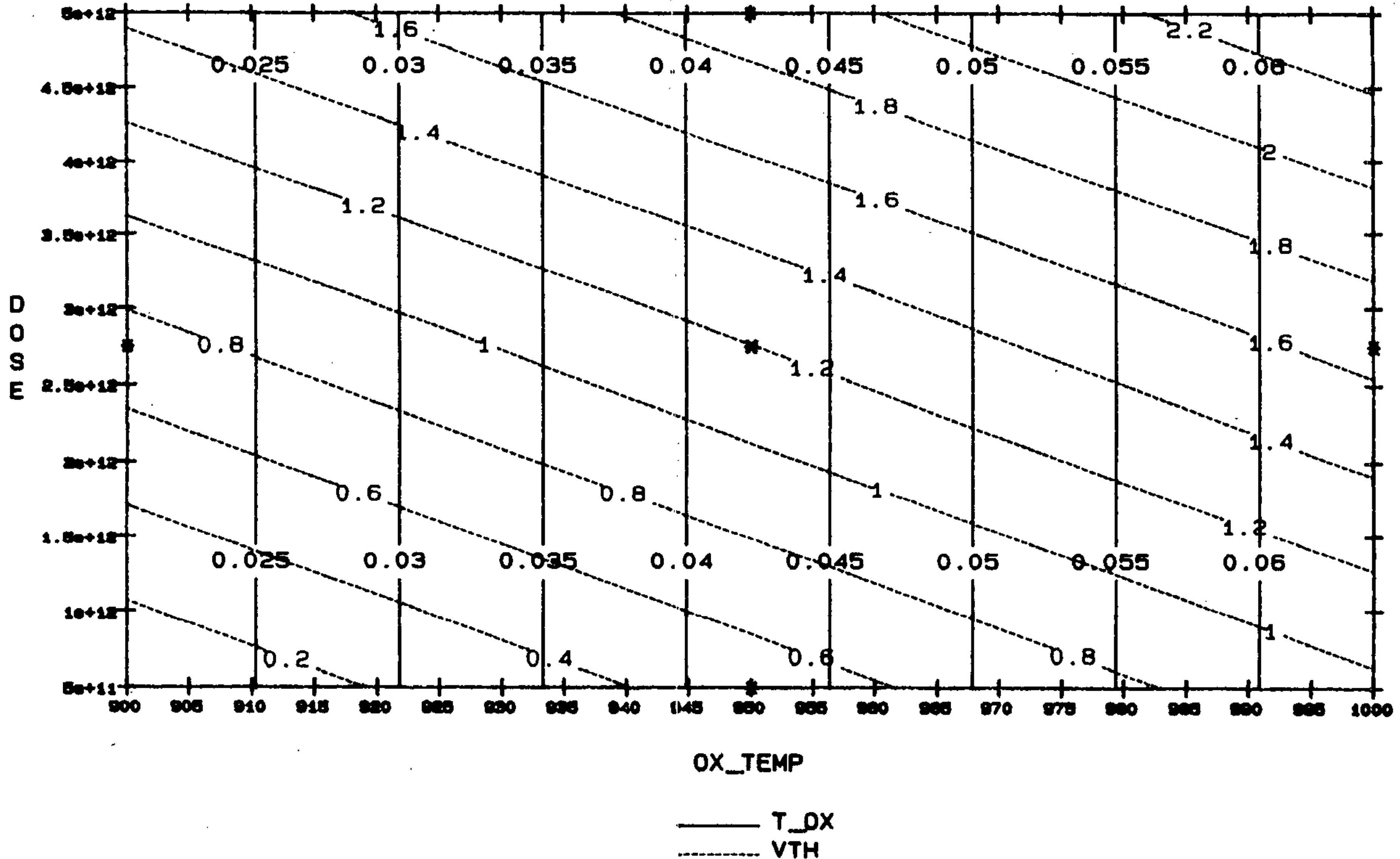


Figure 2: Example of an RS/1 contourplot

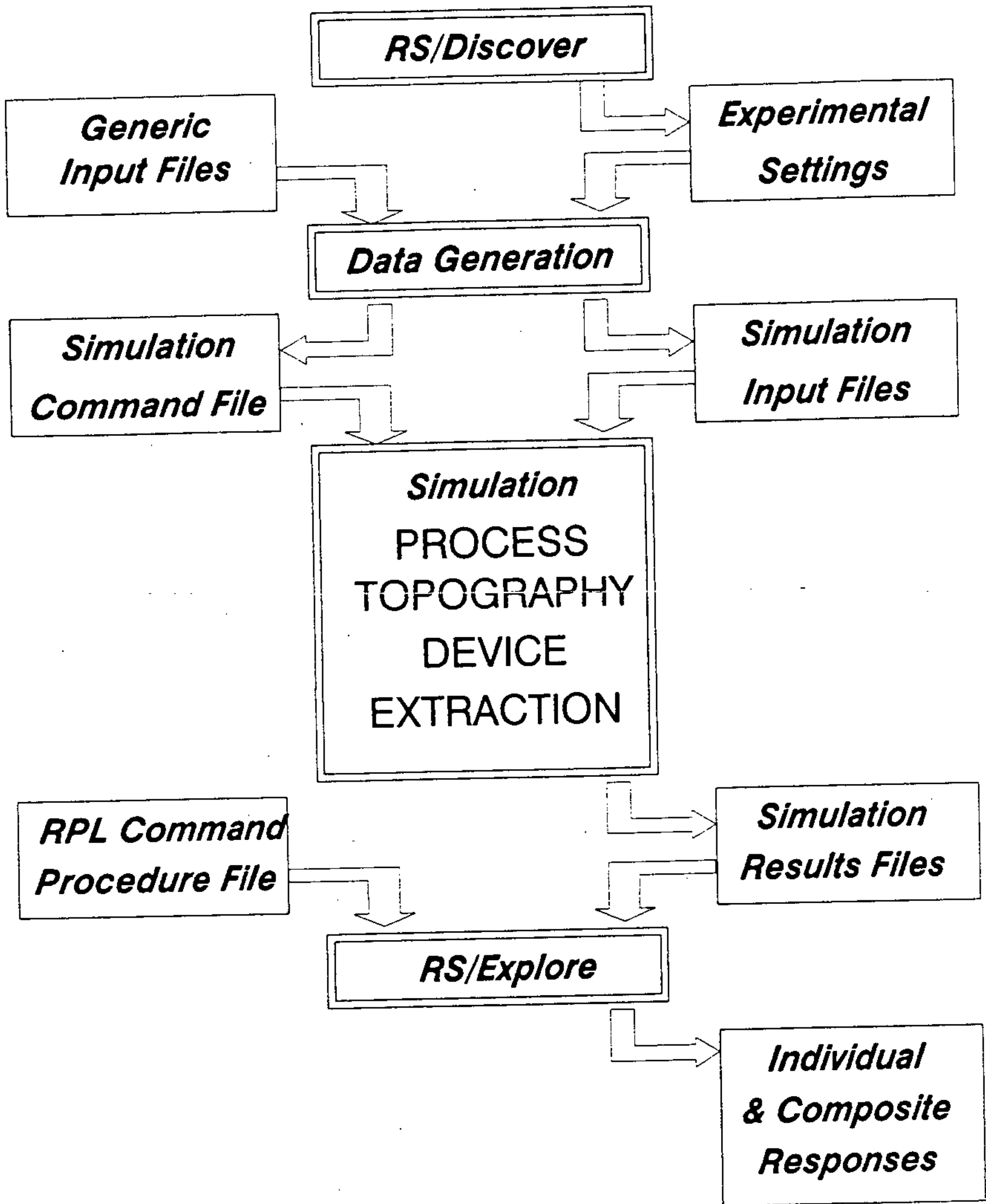


Figure 3: Complete software system

tasks to a number of slave processors and then collates the data when the slaves are finished. This process continues until all tasks have been allocated. The simplist analogy is when the slave processor executes the same executable serial code only on a different set of data points, as is the case in this system. This is obviously very efficient as there is almost linear speed up for each transputer added to the configuration.

The main objective behind PARTICS, was to place a "shell" around the executable code, running on each transputer. Thus no modifictions would be required to the source code to enable each simulation to proceed. Consequently the system is simulator independent. The major problem to overcome was that each transputer on the farm tries to read from stdin (keyboard) and write to stdout (screen). This required the formation of a jobfile which contains basic information concerning renaming of files, eg stdout, stdin and any other files opened by the simulator. An example of such a jobfile for running a batch of SUPREM files is outlined below:

```
begin BLOCK-1
do suprem.b4
assign SUPSEG SUPSEG1
assign SUPDEBUG SUPDBUG1
assign SUPSYNT SUPSYNT1
stdin run01.in
stdout run01.out
done

do suprem.b4
assign SUPSEG SUPSEG2
assign SUPDEBUG SUPDBUG2
assign SUPSYNT SUPSYNT2
stdin run02.in
stdout run02.out
done
.
.
.
end
```

This specifies that slave transputer number 1 will receive input from run01.in, print output to run01.out (rather than the screen) and assign any other files created by SUPREM to a unique name eg. SUPSEG to SUPSEG1 etc. The jobfile is specified in blocks, the next block will not be started until the previous block is finished. This adds an element of control over the order or preference in which the simulations are executed.

While this system was designed primarily for optimising the IC process,

it was deliberately configured so that it was neither simulator nor application specific. As a result it is extremely flexible and can easily be used with any user program.

4.2. TINY

In order to be able to open and assign different names to files, it is essential that each slave transputer has access to the filing system. To facilitate this, an efficient message passing is used to transfer messages from the slave to the master transputer. The system used was called TINY [10], a product of the Edinburgh Parallel Computing Centre. TINY can be used to determine processor topology, process mapping, calculate the routing tables and read and write messages. It has a specific protocol which is used to send and receive messages. To send a message the message type must be specified, along with the destination, and the name and size of the buffer to be sent.

4.2. Fileserver Protocol

A problem arises in that the simulation program will communicate with the filing system at a low level, using a file server protocol, but to pass messages around the network it is essential to use the TINY protocol. The file system expects to receive information in the following file protocol format [11]. For example a 32 bit integer is sent as `int32.value;INT32`, where `int32.value` indicates that a 32 bit integer is about to be sent and `INT32` contains the actual value. A record is sent as `record32.value;INT32::[]BYTE`, meaning a record is aboout to be sent, followed by the size of the record, followed by the record itself.

integer is defined as `int 32.value; INT32`

record is defined as `nilrecord.value` or `record32.value;INT32::[]BYTE`

This can best be illustrated by taking an example of the OpenFile command. As the name suggests this is sent to the filing system from the slave transputer each time a file needs to be opened in the simulation program. Figure 4 outlines the route this command must take to reach the PC (the filing system), or the host transputer in the case of the MEiKO. The SUPREM3 program requests to open a file. The filer protocol, as outlined in [11], needs to send the following parameters to the filing system: openfile.cmd, filename, access.method, open.mode, exist.mode and record.length. These represent a variety of `int32.values` and `record32.values`. This information is, therefore, stored in a buffer and sent down the TINY message passing system. This effectively changes the fileserver or AFS protocol to the TINY protocol.

At the master transputer the message is received and converted back into the fileserver protocol and the necessary file opened at the filing system. An

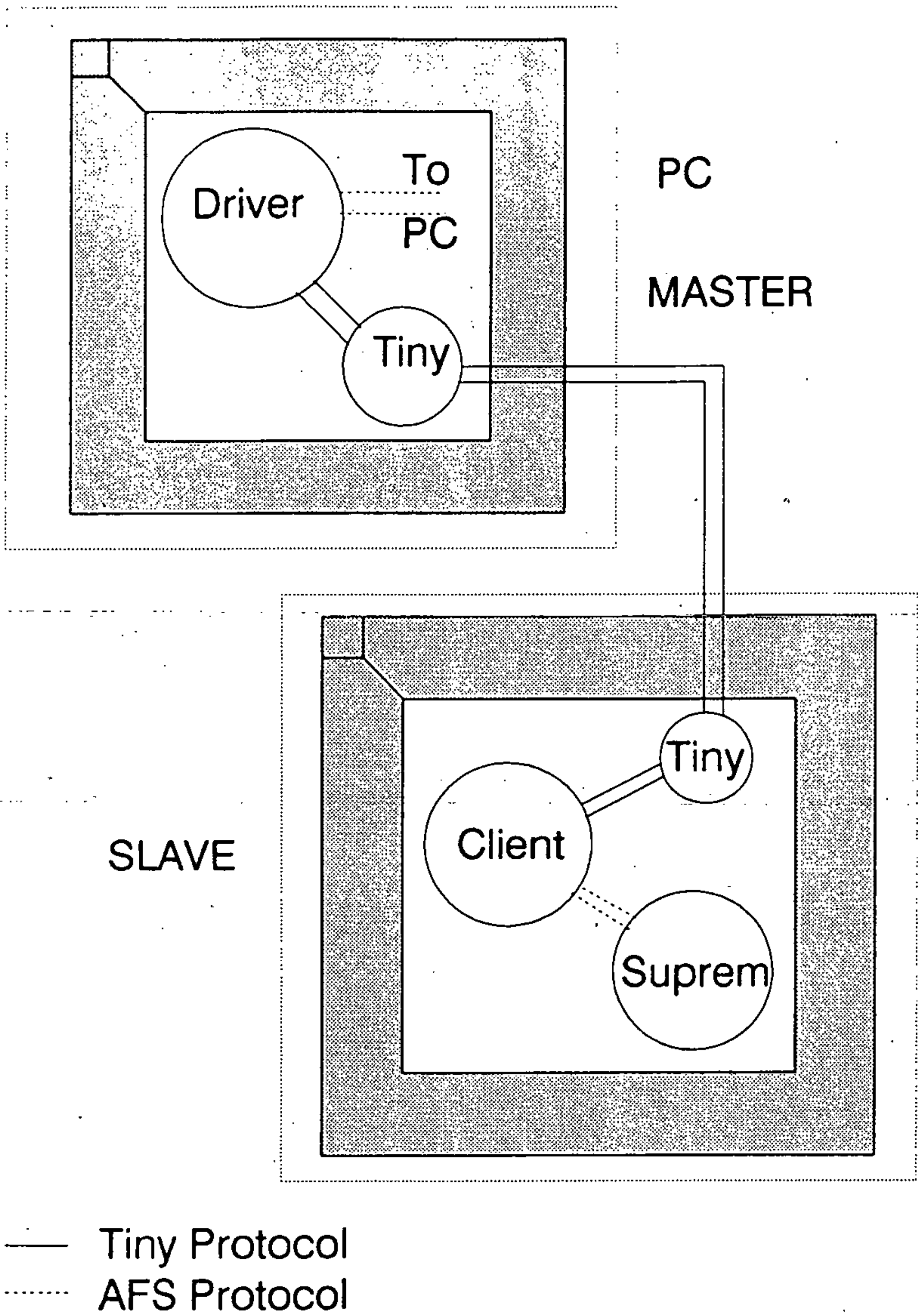


Figure 4: Outline of protocols and message transfer

```
                                                                  14:45:39

        System Info                              Task Info
        -----------                              ---------

        System Nodes :  5                        Task File       : cputest.rc
        Nodes Busy   :  3
        Nodes Ready  :  2                        Current Block : Block-1
        Nodes Dead   :  0                        Jobs completed: 2
                                                 Code in Memory: cputest.b4


    Node     Program     Ready Busy    Jobs done
    ----     -------     ----- ----    ---------
     1       cputest.b4   1      0       1
     2       cputest.b4   1      0       1
     3       cputest.b4   0      1       0
     4       cputest.b4   0      1       0
     5       cputest.b4   0      1       0

        Message from  3 :: len  20 :: tag  24
        Message to    2 :: len   4 :: tag   0
```

Figure 5: System screen information

acknowledgement is then sent back to the slave, with details of the ID number associated with that particular file, again changing from fileserver protocol to TINY protocol and back.

Figure 5 shows the information provided by the system at runtime. The 5 nodes refer to the 5 slave transputers on the PC system. In order to speed up the loading of large executable codes, they are loaded from memory, if they are already resident. This speeds up the total execution time as disk access is much slower.

With regards to the simulations, the 31 process simulations can be carried out, either on the 5 transputers on the PC or the 17 on the MEiKO. Once they are completed then the results can be fed into a device simulator and 31 device simulations carried out in a similar fashion to the process simulations. All this can be controlled from the jobfile.

Table 1 shows the comparisons between the PC system, (using both T4 and T8 transputers), the MEiKO system, and a SUN3. Three different programs, were run five times on each system and the execution times recorded. This was carried out in parallel for the transputer systems, and sequentially in the case of the SUN3. The MEiKO times are superior to the PC T8 times, due to reduced communications overheads.

	PC T8	PC T4	MEiKO T8	SUN3
Prog1	5.2 min	93 min	4 min	34.5 min
Prog2	2.5 min	37.7 min	1.7 min	13.2 min
Prog3	0.9 min	4.4 min	0.3 min	1.3 min

Table 1: Execution times.

5. CONCLUSIONS

In conclusion PARTICS offers almost linear speed up for each transputer added to the system configuration. It is not essential to change the source code of the application software and the "shell" allows redirection of the necessary files. The jobfile concept allows ease and flexibility, resulting in different simulation code running concurrently on different transputers. To use the system with a particular application, all that is required is a minor editing change to the jobfile and the system can be utilised with any user program.

The use of PARTICS has a very significant improvement on the elapsed time to optimise a process, reducing a week of CPU time to that of only a few hours. This can be achieved using a supercomputer like the MEiKO Computing Surface. A PC system can also be used to reduce the execution time, thereby reducing computing costs.

7. ACKNOWLEDGMENTS

The authors would like to thank the Dept.of Education N.Ireland for the funding of the project, INMOS (Bristol) for their help and support and the Edinburgh Parallel Computing Centre for the provision of the message passing system, TINY.

8. REFERENCES

1. G.E.P.Box and W.G.Hunter and J.S.Hunter. *Statistics for experimenters : An introduction to design, data analysis and model building* New York: Wiley 1978.

2. A.R.Alvarez et al. Application of statistical design and response surface methods to computer-aided VLSI device design. *IEEE Transactions on Computer Aided Design* 7(2): 272-278, 1988.

3. C.P.Ho and J.D.Plummer and S.E.Hansen and R.W.Dutton. VLSI Process modeling - SUPREM III. *IEEE Transactions on Electron Devices* 30(11):1438

1983.
4. SUPREM4 - Stanford University.
5. SUPRA - Stanford University.
6. S.Selberherr *Analysis and Simulation of Semiconductor Devices* Springer-Verlag Wien New York, 1984.
7. PISCES - Stanford University.
8. S.Selberherr and A.Shultz and H.W.Potzl. MINIMOS - A two-dimensional MOS transistor analyser *IEEE Transactions on Electron Devices* 27(8):1540-1550, 1980.
9. Z.P.Yu and R.W.Dutton. SEDAN-3 User's Manual. Technical report, 1985.
10. L.J.Clarke, Ph D. Thesis, University of Edinburgh 1990.
11. INMOS Limited. *Occam Toolset User Manual* 1988.

"A General Taskfarming Tool : Its Application to Semiconductor Fabrication", European Workshop on Parallel Computing, Barcelona, March 1992.

A General Taskfarming Tool : Its Application to Semiconductor Fabrication

G.J. Gaston, W.J.C. Alexander, L.J. Clarke [†] and A.J. Walton.

Edinburgh Microfabrication Facility, Department of Electrical Engineering, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

[†] Edinburgh Parallel Computing Centre, University of Edinburgh, The King's Buildings, Edinburgh, UK, EH9 3JL.

1. Introduction

Semiconductor fabrication technology has now increased to the level of several million transistors on a single silicon chip. Original methods of experimentation by trial and error to create a new process or to optimise an existing one are no longer financially viable. To alleviate these problems, the major semiconductor fabrication companies are now using simulation software to establish processes and to reduce the number of real experimental evaluations that would otherwise be required. Although simulations reduce the time greatly, when compared to prototype runs, they can still be very CPU intensive and each simulation may take hours or days to complete.

It is the purpose of this paper to outline a taskfarming environment on an array of transputers at the Edinburgh Parallel Computing Centre. The computer used is a MEiKO Computing Surface. The taskfarm operates in the mode of a master transputer allocating tasks to a number of worker transputers, taking the from of a pipeline.

2. General Outline Of Taskfarm

2.1. Taskfarm Requirements

The following is an outline of the salient features that were required in the design and construction of the taskfarm system:

- There should be no changes to the user source code to facilitate the operation of the system.
- The system should be flexible to use with any user software.
- One file should control all user interaction.

As noted above, it was considered important to be able to have a general taskfarm system that could be easily adapted to be used with any user software. This would again reduce the time and effort required in porting code onto parallel machines.

By providing the facility of a control file, any changes that are then required to the system can be implemented by a minor editing change of a single file.

2.2. Problems to Overcome

Given the strict guidelines outlined in the previous section, many problems presented themselves. The following section describes the major problems to be overcome. Section 3 explains how these complications were surmounted.

1. Due to an identical copy of the code running on each worker transputer, all workers would try to read from the keyboard, write to the screen and read/write from/to the same files. It was essential that any solution would remove such a scenario.

2. Since each worker thinks that it is the only code running and that it alone has complete access to the filing system, all workers must be given a pseudo access to the filing system. This will require some form of communication system to pass file requests to and from the filing system from each worker.

3. For large jobs ie. with a large number of worker transputers, there may be a system limit to the number of files that can be kept open at any one time. It is important that this limitation should be allowed for in the taskfarm.

For the purposes of this taskfarm the semiconductor fabrication simulation software, SSUPREM4 [1] was the code used. This particular package comes under the generic name of a process simulator.

3. Operation Of Taskfarm

3.1. Jobfile

In order to overcome the problem of each worker reading the keyboard and writing to the screen etc. it was necessary to define a convention for the renaming of files. This lead to the formation of a jobfile which details the renaming convention eg. redirection of stdin, stdout and any other files peculiar to the user code. Part of such a file is outlined as follows, eg. `stdout run01.out`.

This specifies that worker transputer number 1 will print output to run01.out (rather than the screen).

3.2. TINY

In order to be able to open and assign different names to files, it is essential that each worker transputer has access to the filing system. To facilitate this, an efficient message passing is used to transfer messages from the worker to the master transputer. The system used was called TINY [2], a product of the Edinburgh Parallel Computing Centre. It has a specific protocol which is used to send and receive messages. To send a message the message type must be specified, along with the destination, and the name and size of the buffer to be sent. eg. `t_sseq(AFS_TYPE,dest,Buf,sizeof(Buf))`. A similar convention is used to receive data except the routine `t_recv` is called.

3.3. Fileserver Protocol

SSUPREM4, which is written in C, was compiled on the transputers using 3L Parallel C [3]. Using 3L utilities, namely the afserver, (which is part of the Parallel C package) SSUPREM4 will communicate with the filing system at a low level, using a fileserver protocol. In order to pass messages around the network it is essential to use the TINY protocol. The file system expects to receive information in a file protocol format [4]. This file protocol information is, therefore, stored in a buffer and sent down the TINY message passing system. This effectively changes the fileserver or AFS protocol to the TINY protocol. This is outlined in figure 1.

At the master transputer the message is received and converted back into the fileserver protocol and the necessary file opened at the filing system. An acknowledgement is then sent back to the worker, with details of the ID number associated with that particular file, again changing from fileserver protocol to TINY protocol and back.

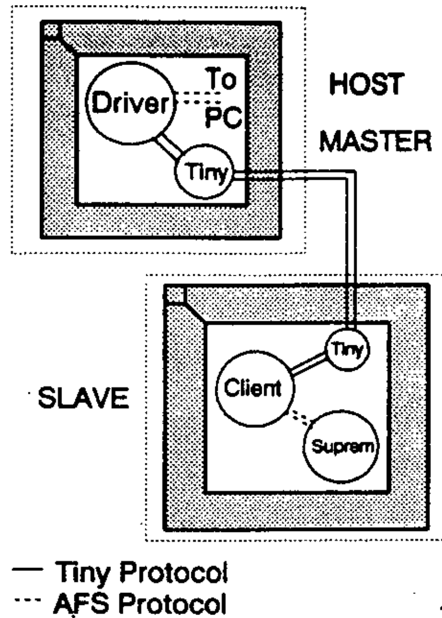


Figure 1: Outline of protocols and message transfer

3.4. General Structure of Program

As can be seen from figure 1 there are two main parts to the taskfarm, namely the driver and the client.

Driver Essentially the driver receives TINY protocol commands from the worker, converts them to an AFS Protocol, executes the required AFS function eg. openfile command and sends a TINY protocol acknowledge back to the worker.

It is also used to coordinate the allocation of tasks, collection of results, reading of the jobfile, etc. In addition the driver program is responsible for checking the maximum number of open files allowable on the system and comparing this with the number actually open at present. If, on receipt of an openfile command, the system number will be exceeded then it is the function of the driver to close a file to facilitate the opening of the new file. A priority is given to each file and thus low priority files will always be closed before a high priority file.

Client The client program receives AFS protocol commands from SSUPREM4, makes any necessary changes to the filename, converts the AFS protocol to a TINY Protocol, sends the command to the driver and finally receives an acknowledge from the driver.

The client is also responsible for translation of filenames as outlined in the jobfile. For example on request of an openfile for SSUP4USKEY on worker number 2, the client will translate this to an openfile for SSUP4USKEY2, and this information sent on to the driver as normal.

4. Results

Figure 2 shows the results obtained when using the taskfarm with SSUPREM4 and different data files. The results are for a farm of 10 transputers and the time for a SUN4 is the total time taken to run 10 sequential jobs. As the task number increases,

so does the computational requirements of the job. As a result any communications overhead of the transputer system becomes less and less significant as the cpu times increase. This is evident by comparing the elapsed times for task 1 and task 4 for the two different systems.

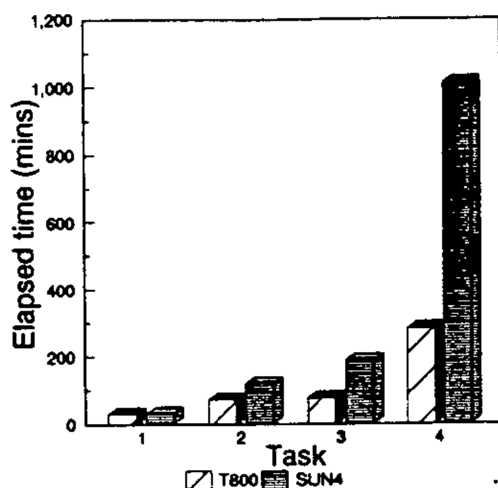


Figure 2: Comparison of taskfarm and SUN4

The above results were only obtained using relatively small simulations and if a complete process was to be simulated then the cpu time is likely to be in hours or days rather than minutes. Thus the speedup of the taskfarm system would be even more beneficial.

5. Conclusions

In conclusion the taskfarm offers almost linear speed up for each transputer added to the system configuration. It is not essential to change the source code of the application software and the "shell" allows redirection of the necessary files. The potential commercial value to this is obvious as no inherent knowledge about parallel computing is required, just the ability to recompile existing code. The jobfile concept allows ease and flexibility, resulting in different simulation code running concurrently on different transputers if required. To use the system with a particular application, all that is required is a minor editing change to the jobfile and the system can be utilised with any user program.

The use of the taskfarm has a very significant improvement on the elapsed time to simulate a process, reducing a week of CPU time to that of only a few hours. This can be achieved using a supercomputer like the MEiKO Computing Surface. This system has also been implemented on a PC system with very similar performance and at a system cost that is affordable for small companies.

6. References

1. SSUPREM4 Silvaco Data Systems 1991.
2. L.J.Clarke, Ph D. Thesis, University of Edinburgh 1990.
3. 3L Parallel C Manual V2.1.
4. INMOS Limited. *Occam Toolset User Manual* 1988.