(7A-2)

DEVELOPMENT TOOLS FOR SIMULATION PROGRAMS APPLICATION TO MULTILAYER PROCESS SIMULATION

B. BACCUS*+, D. MOREL**, D. COLLARD* and E. DUBOIS*
* ISEN, 41 Boulevard Vauban, 59046 Lille Cedex, FRANCE
** DIXILOG, 41 Boulevard Vauban, 59046 Lille Cedex, FRANCE
+ Visiting scientist at TOSHIBA ULSI Research Center, Kawasaki, JAPAN

INTRODUCTION

Due to the size and complexity of modern simulation programs, the development aspects require special attention. In the following, we propose a strategy aimed to facilitate this work, with the example of process simulation. These development aspects must be considered as internal to the code, in opposition to the data structure generated for process and device simulators interfaces (1).

The goal of such strategy is as follows:
- to facilitate the writing of algorithms and code programming,
- from the "developer" point of view, to avoid data management or any dependence on the operating system,
- to enhance the possibility of program transfer and to minimize the maintenance efforts.

ANALYSIS OF THE NEEDED DATA

When studying the data, informations fluxes as well as the algorithms needed to perform a multilayer process simulation such as represented in fig. 1 (2), several basic objects can be distinguished. These objects (or entities with specific qualifiers for each one), are domains (layers), elements, nodes, materials. They must be easily accessible through the Fortran language (taken as example, due to its wide use). For each one, the informations about an individual are obtained by transfer through Common statements. As an example, in order to obtain (or to store) concentrations, coordinates... of a node:
- read the informations : CALL GETNOD(id,in,iret)
- write the informations : CALL PUTNOD(id,in,iret)
- the associated Common is : COMMON /NOD/ X,Y,CAS,CBO,CPH,CSB

where "id" is the domain number and "in" the node number in this domain. This kind of calls has proved to be very efficient for complex algorithms. Indeed, the point to emphasized here is that this type of strategy is always used in the development of the code and not only to define a general structure. It should be noted that although this structure has been used in 2D, it can be extended directly to 3D.

The next point to consider is the management of this amount of data. It can be done by the use of sophisticated data base or with simple arrays management. We have used both approaches in IMPACT4, whether the limitations on internal memory must be considered or not.

STRUCTURATION

The goals of structuration are as follows:
- to define several specialized libraries (probably to be written by different developers),
- to determine common notations for a possible integration of these libraries into the same program (or to be re-used in other codes).
- to facilitate the program transfer, due the independence of these different libraries.

Indeed, this point concerns not only the application part (linear solvers, mesh generators...), but also all the data base and the graphics. Moreover, the instructions depending on the operating system are also considered in a separate module (fig. 2).

APPLICATIONS

The two-dimensional multilayer process simulator IMPACT4 has been developed on the basis of this strategy (2,3). In order to show the possibilities of these tools, fig.3 presents a complex example where adaptive mesh refinement is used to enhance the precision in determining the stress, during 2D oxidation. The management of the important amount of data needed for these calculations is facilitated with this overall strategy.

(1) C. Lombardi, Nasecode V conf., short course, Dublin, June 1987.
(2) B. Baccus, E. Dubois, D. Collard and D. Morel, Solid-State Electronics, vol. 32, No 11, pp. 1013-1023, 1989.
(3) B. Baccus, PhD, University of Lille, 1990.

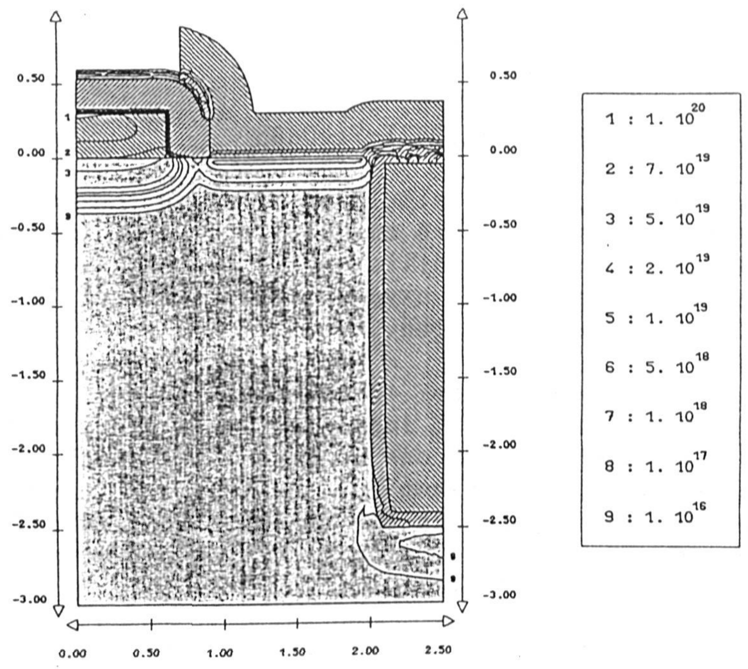
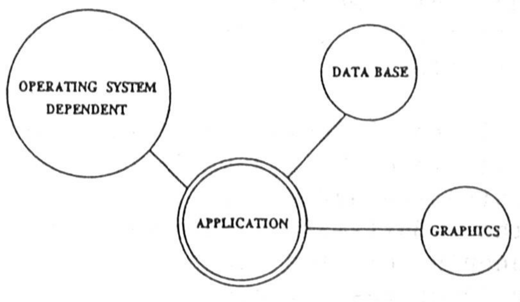


Fig.1 IMPACT4 simulation of an advanced bipolar transistor featuring polysilicon layers and trench isolation

Fig.2 General structure of a code. The operating system dependent instructions include the I/O operations and initialisations (named as IBMLIB, SUNLIB...)



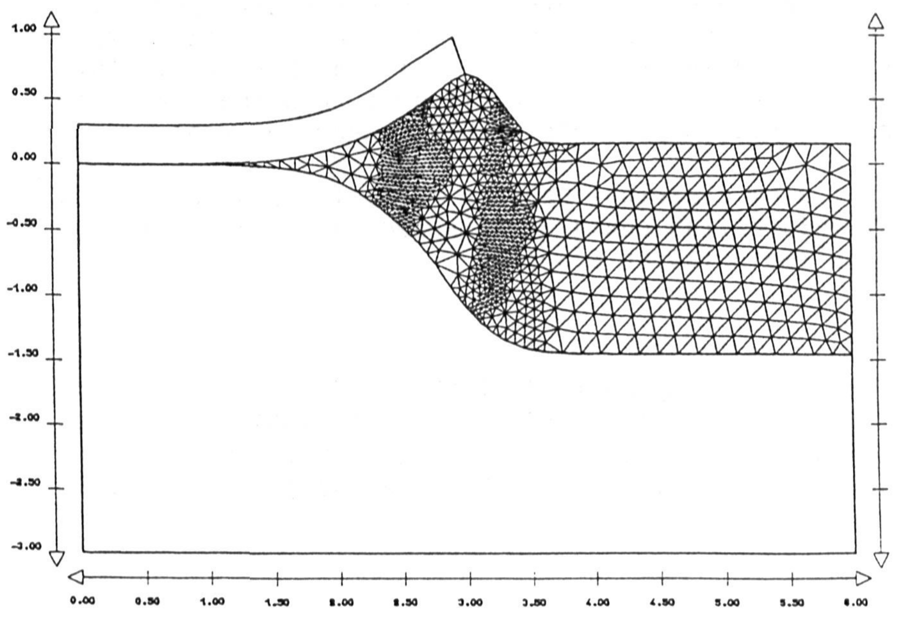


Fig. 3 Adaptive mesh refinement during stress calculation, for 2D oxidation