Specification of a Software Architecture and Protocols for Automated VLSI Manufacturing System Operation

자동화된 VLSI 생산 시스템 운용을 위한 소프트웨어 구조 및 프로토콜 설계

Jonghun Park, Jongwon Kim and Wook Hyun Kwon
(박종헌, 김종원, 권욱현)

요 약: 본 연구에서는 자동화된 VLSI 제조 시스템 환경에서의 로트 조정기 및 범용 셀 제어기의 구축에 필요한 새로운 소프트웨어 구조 및 프로토콜을 제시하였다. 반도체 제조 시스템의 운용 제어 활동은 로트 조정기와 범용 셀 제어기가 상호 협조적으로 통신하는 클라이언트/서버 구조로 모형화 되었으며, 로트 조정기는 하나 이상의 작업을 수행할 수 있는 범용 셀 제어기에 작업을 의뢰하는 클라이언트로서 작동된다. 반도체 제조 시스템의 운용 소프트웨어와 관련된 기존의 연구들이 개념적인 구조와 전략 만을 다루었던 것과는 달리, 본 연구에서는 생산 설비 뿐 만 아니라 물류운반 장치의 제어를 위하여 상세한 수준에서의 설계가 제시되었다. 본 연구의 특징으로는 설비 구성, 로트 형태, 일정 계획 규칙 등의 변경에 대한 동적 재구성 가능성을 들 수 있다. 또한 제안된 설계는 상용화된 프로세스 통신 기능을 사용하여 구현이 용이하다.

Keywords: VLSI manufacturing, cell controller, manufacturing system operation

I. Introduction

The manufacturing of very large scale integrated (VLSI) circuits is perhaps the most complex manufacturing process found today [8]. Traditionally, this complexity has arisen from many sources such as process intricacy, product diversity, uncertainty and technology changes. Furthermore the continuous introduction of automation technology like CIM (Computer Integrated Manufacturing) has also created a new challenging problem of efficient management of automated VLSI facilicity. Therefore, as noted by Moyne [10], the cell controllers which are responsible for seamless integration of equipments and high-level controllers in a facility control structure should be clearly defined and characterized.

The objective of this research is to present a new architecture and protocol of lot coordinator and generic cell controller in which operational coordination is achieved through process cooperation in a fully automated VLSI manufacturing environment. In this research, the processes communicate with other distributed processes managing their own manufacturing resources by means of certain types of predefined messages. There are two basic types of messages : an operation request and an event notification. The coordination activities are modelled as a client-server interaction, which facilitates understanding the way it operates [1].

Our motivation comes from the fact that there has been a growing need for the control system which has reconfigurability and modularity to fully exploit flexibility of individual automated VLSI equipment. But, design ofthe control system for coordinating various equipment and material handling system is not an easy job and it must be pointed out that the software is inherently difficult [3]. The control system should be able to coordinate concurrent jobs competing for nonsharable and limited resources (for example, reactive ion etcher and furnace) and to synchronize the events occurring from VLSI equipments to perform an operation as required by recipe. Furthermore, it is commonly found that a recent automated manufacturing system is composed of functional cells communicating with each other through the local area network [16]. Therefore, a distributed computing solution, where processes cooperate with one another to work toward a common goal, can help to obtain a simple design and implementation for an application that is distributed in nature [17].

There have been a lot of research efforts along with the development and implementation of the automated manufacturing cell controllers. One of the representatives is MMST (Microelectronics Manufacturing Science and Technology) CIM system framework which is based on open distributed system and object technologies [12]. Moyne and McAfee [10] proposed a generic cell controller model to maximize software portability and to reduce unnecessary redundancy. They also verified the feasibility of the design using simulation. Naylor and Volz [13] suggested a modeling formalism based on a first-order language. Recently, a formal shop floor control model named message-based part state graphs (MPSG) which can serve as a basis for automatic generation of the control software was developed [18]. Scheduling and control models based on a distributed computing technique have been proposed by Aggarwal et al. [2], and

접수일자 : 1996. 2. 29., 수정완료 : 1996. 11. 26.
박종헌, 김종원, 권욱현 : 서울대학교
※ 본 연구는 서울대학교 반도체공동연구소의 반도체분야 교육부 학술 연구조성비 (95-E-1042)에 의해 수행되었음.

Gauthier et al [7]. There exists also another approach which is based on distributed artificial intelligence (DAI), where scheduling decision is made dynamically using bidding mechanism among the agents representing components in manufacturing system [15], [16]. In this line of research, Ramos [15] suggested the architecture in which agents representing tasks and resources cooperate each other. This paper is organized as follows : Section 2 describes the proposed coordination architecture. The architecture includes two major cooperative processes named lot coordinator and generic cell controller. The lot coordinator and generic cell controller are covered in detail in Sections 3 and 4 respectively. The paper is then concluded in Section 5.

II. The proposed coordination architecture

The production of IC's is accomplished in a four-stage process that consists of wafer fabrication, wafer probe or test, IC assembly, and IC burn-in or functional test [9]. A wafer contains many identical chips, and wafers are grouped in lots, each of which travels together in a standard container and is destined for conversion to the same final product [4], [6]. The wafer fabrication is done in a clean room called a fab, which is typically divided into U-shaped bays.

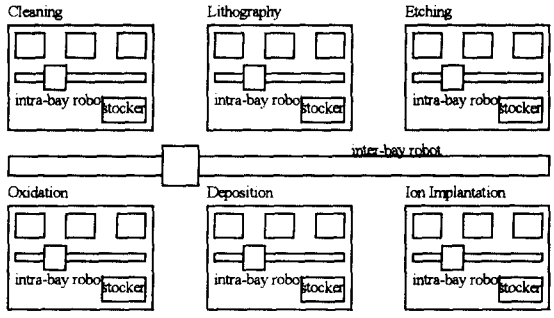

Fig. 1. An example fab layout.

A bay generally contains major pieces of equipments on which a basic operation is performed. Each lot entering the clean room has an associated process flow, often called a recipe, that consists of precisely specified operations executed in a prescribed sequence on predetermined pieces of equipment [6].

In general, many modern wafer fabs have been laid out as a series of bays, each supplied with interoperational storage (stockers). The transport operations in a wafer fab have usually been classified into interbay and intrabay lot transfers [5]. A bay is production work shop dedicated to a particular process. Each bay has production and measurement equipment, automatic wafer cassette storage (stocker), and an inter-bay transfer robot [11]. Figure 1 shows an example layout of fab area.

In most cases, the controllers in an automated manufacturing system are organized hierarchically to manage complexity, which is called a hierarchical control architecture [14]. In our approach, the coordination activities are modelled as cooperative communication between top-level controller and its subordinate controllers, where the top-level controller acts as a client requesting an operation and the subordinate controllers as servers performing operations as their services. Based on this modelling strategy, we will denote the top-level controller as the lot coordinator (LC) because its primary role is to coordinate lots and also denote subordinate controller as the generic cell controller (GCC). Figure 2 shows the conceptual architecture composed of LC and GCC.

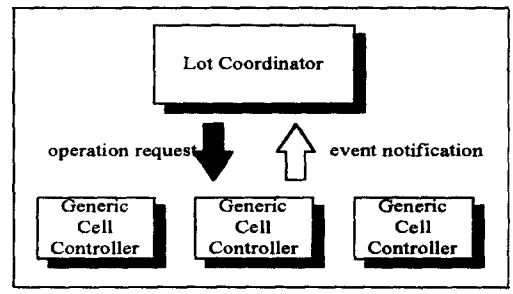

Fig. 2. Proposed architecture.

GCC is responsible for managing one or more equipment controllers. It receives an operation request from LC and then performs various activities, such as scheduling, dispatching, deadlock handling, and communicating with its equipment controllers. For example, GCC can be a controller of a bay which is composed of production equipments interconnected by one intra-bay material handling robot. According to its functionality, GCC can be classified into transformation GCC and inter-cell material handling GCC. LC takes care of coordination and synchronization among the different GCCs. It commands production activities by requesting a necessary operation to corresponding GCC and handles incoming events through interprocess communication facility.

To dispatch and monitor a wafer manufactured in the system we need some data abstractions. A data abstraction LOT represents a lot having one or more operation requirements of the wafer type to be produced. LOT has its unique identifier which has been assigned to it. Furthermore, two kinds of state information should be maintained within LC for LOT to issue an appropriate operation request : they are current states in transformation state graph (TSG) and material handling state graph (MSG). The TSG is a graph representing sequence of operational conditions which are to be satisfied for the lot to be produced successfully. TSG can be generated from a recipe by selecting one instance from all possible alternative routes in accordance with the performance criterion used in the scheduling.

A transformation state graph TSG = (V, E) is defined as follows.

1) V is a finite set of transformation states of a lot $(v_1, v_2, ..., v_n)$. A node can be marked to represent the current state.

2) E is a finite set of state transitions. An arc can be

either active or inactive. Each arc e is assigned a label op(e), called required transformation operation for the state transition.

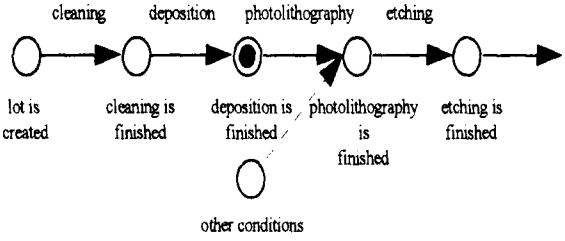

Fig. 3. An example of TSG.

Figure 3 depicts a simple example of partial TSG where a lot needs first four operations to be done sequentially. It shows that the lot is in state of having been finished deposition operation (marked as black circle). As is mentioned above, the label on arc is used to indicate the required transformation operation for the state change of a lot. An arc can be either active or inactive, which indicates whether the task is currently being processed by any of transformation resources or not. Since the TSG will vary along with the wafer type, every lot should have its own TSG identifier. Also, it should be noted that other specific conditions such as test requirement or TSG states of other lots can be easily included in the TSG because it is just a state/transition graph which is not necessarily lincarly ordered. (a dotted arc in Figure 3 indicates this case.)

Furthermore, from the observation that operation in a VLSI manufacturing system can be classified into transformation or material handling, LC maintains another state graph called material handling state graph (MSG). Note that the material handling state of a lot is maintained in a separate graph because the material handling operation cannot be determined in advance in contrast with the deterministic transformation requirement. It can be repeated with the same cycle without any transformation having been made. Also unlike TSG, MSG is a system-wide graph in that it is shared by all lots currently being in a manufacturing system. A node in MSG represents a station where a lot can be loaded and an arc represents accessibility and required material handling operation. Therefore, station s_1 is directly connected with station s_2, if a lot residing in s_1 can be moved to s_2 with a single material handling operation request. The MSG serves then as a source of information for the location of lot at any moment and shows all possible material handling operations for each lot at a specific location.

A material handling state graph MSG = (V, E) is defined as follows.

1) V is a finite set of accessible stations in a VLSI manufacturing system, (s_1, s_2 , ..., s_n). A node can be labeled as t_i to represent that lot i is located at the node.

2) E is a finite set of accessibility relationships. An arc can be either active or inactive. Each arc e is assigned a label op(e), called required material handling operation for movement between stations.

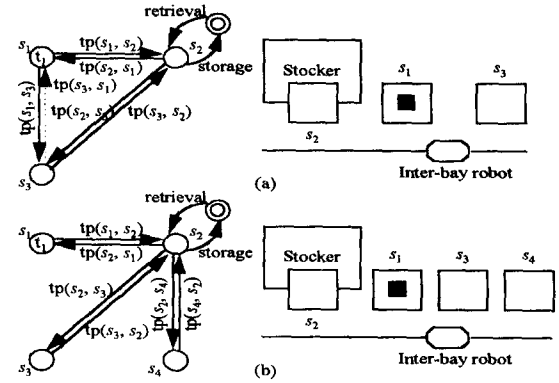

Fig. 4. An example of MSG.

An example of MSG and its corresponding layout is illustrated in Figure 4 (a), where stations s_1, s_2, and s_3 are interconnected by inter-bay robot. Material handling operation tp(s_i, s_j) represents a transportation from s_i to s_j by the inter-bay robot. As in TSG, the dotted arrow represents active arc showing that an operation is being performed currently. The node s_1 is labeled as t_1 indicating that lot t_1 is at station s_1. The node marked as doubled circle in the MSG represents a stocker where multiple lots can be stored. The lots located in a stocker can be retrieved to station s_2 by use of retrieval material handling operation, and vice versa. In summary, a lot has the current states in the TSG and MSG, so that the LC can identify the current operation requirement of the lot concerned. Since TSG and MSG are regenerated and fed into the LC whenever a recipe or fab configuration is changed, respectively, we can expect reconfigurability from the proposed approach. For example, Figure 4 (b) depicts a newly generated MSG when a new station is introduced to the fab. In this case, it is assumed that the transportation operation is only possible between s_2 and s_j, where j is 1, 3, or 4.

A more detailed architecture showing basic cooperations between LC and GCC is illustrated in Figure 5. LC receives an event from remote GCC via monitor process. A process named dispatcher sends an operation request to GCC. Rectangles in Figure 5 imply that LC and GCC are remotely distributed processes running concurrently. Likewise, due to the asynchronous nature of sending and receiving an event and an operation request, every GCC has reporter and monitor processes, respectively. It is also to be noted that every GCC has its own scheduler and supervisor. It means the scheduling decision is made independently and dynamically to maximize production efficiency by each resource with its own scheduling rule. In addition, the supervisor is responsible for handling deadlock which is possible within resources managed by a GCC.

Four major function primitives are defined in order to make distributed process communication possible as follows :

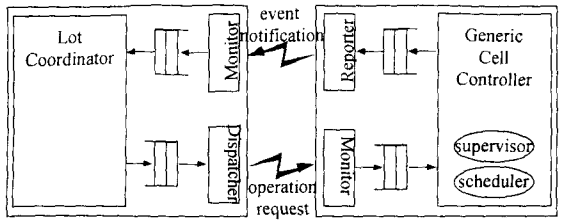

Fig. 5. Cooperation between LC and GCC.

request_operation (*GCC_id, lot_id, opn_id, opn_arg*)
notify_event (*GCC_id, GCC_state, lot_id, opn_id, opn_state*)
query_state (*GCC_id, station_id*)
notify_state (*GCC_id, station_id, station_state*)

By the use of function **request_operation**, LC can send an operation request of a lot to appropriate GCC designated by *GCC_id* (generic cell controller identifier). Function **notify_event** is used to send an event notification concerned with operation of a lot from GCC (specified by *GCC_id*) to LC. The variable *GCC_state* can have values of *ready* or *failure* mode, or more modes - in terms of each equipment, not of entire GCC - for more delicated control. The state *ready* represents that the GCC can do at least one of its predefined operation services. The variable *opn_state* is used to indicate operation processing state which can be one of the following : *operation_start*, *operation_finish*, *operation_start_error* or *operation_finish_error*.

There are two additional function primitives named **query_state** and **notify_state**, which are used for querying and notifying the state of a specific station, respectively. The function **query_state** is invoked when LC needs to know the state of a station in case of failure recovery. The query result is sent by **notify_state** function. The variable *station_state* can be *empty* or *lot_id* if it is occupied by a lot.

III. Specification of lot coordinator

Two different strategies can be considered for the coordination of operations in automated VLSI manufacturing system. One is based on the state changes of lots (defined as an *early transportation strategy*), and the other on the state changes of transformation equipments (defined as a *late transportation strategy*). In the early transportation strategy, every decision regarding the next operation for a lot (e.g. *what is the next operation of a lot* ?) is made whenever the lot finishes its operation at an equipment. In the late transportation strategy, however, every decision regarding the next operation of a transformation equipment (e.g. *what is the next operation of the equipment* ?) is made whenever the equipment finishes its operation. In addition, since material handling resources are treated as secondary resources, the arrival of a task triggers the execution of an operation.

One of the advantage of the late transportation strategy is that routing flexibility is increased, because the decision as to which equipment a lot will be sent, can be made as late as possible. But, in terms of makespan of a lot to be completed, the early transportation strategy has an advantage over the equipment-oriented one, because total waiting time of a lot can be reduced. This is due to the fact that as soon as the buffer and transporter becomes available while the equipment is busy, the lot can be transported to the local buffer of the other equipment in advance for its next operation. On the other hand, in the late transportation, it is not possible to transport the lot to the local buffer of the equipment which will perform its next operation until that equipment becomes idle.

As introduced in the Section 2, LC operates in an event-driven manner in that decision is made whenever an event is received from a remote GCC or an system operator. Since most of these events are results of state changes of a lot, the LC can be said to be a lot-oriented coordinator rather than equipment-oriented. In order to handle a variety of events sent by GCC, the following three criteria are used to group them into more abstract events.

- operation start or operation finish
- success or error
- transformation operation or material handling operation

Basically, an event from a GCC is a response to the operation request made by LC. And it is sent by GCC whenever the GCC starts or finishes an operation. It can be either successful or erroneous. Also it can be related either with transformation or with material handling operation. For example, if an event named *deposition_start_error* is received, it indicates that there is an error at the starting time of a transformation operation named deposition.

Upon receiving an event message, the event controller of LC identifies the type of event and updates the corresponding lot state. And then, if the lot state has been changed, the event controller enqueues next required operation of the lot to GCC by means of the **request_operation** function. And then, this request is sent to GCC by dispatcher process of LC. A pseudocode of the event controller is described in Figure 6.

In the pseudocode of Figure 6, dequeued event message is stored to a variable named *msg*. The function **update_lot_state** updates states in TSG and MSG. By identifying the lot related with the event, the lot state is updated and the **make_operation_request** function is called if a lot is newly created or an *operation_finish* event is occurred. In the function **make_operation_request**, LC first identifies the next operation of the lot and checks if all the other conditions (e.g. states of other relevant lots) are satisfied for the next operation. When satisfied, LC then checks whether a material handling operation is necessary or not by looking up the state in TSG and MSG. Then LC invokes the function **request_operation** for next operation the lot concerned (it can be either transformation or material handling

operation). The event *update_system_graph* is included in the Figure 6 to emphasize the fact that the TSG or MSG can be reconfigured dynamically by a system triggered event during system operation, if there is any change made in the graphs.

It should be also noted that the LC coordinates operations using the early transportation strategy which allows a local buffering, since it operates in an event-driven manner in that decision is made whenever a state change of a lot is received from the distributed GCCs. Therefore, it can provide better equipment utilization compared with the late transportation strategy.

Ⅳ. Specification of generic cell controller

Fundamental activities of GCC are scheduling, dispatching and deadlock recovery. Whenever one of equipments managed by GCC becomes *ready* state, GCC selects one request from a lot list waiting for that equipment and then dispatches an operation. Internal process architecture of a GCC may vary slightly along with

```
MESSAGE  msg;
LOT  current_lot, lot_list[];

do {
  msg = dequeue_message ();
  current_lot = identify_lot(msg);
  switch (msg) {
    case start_system :   // system-triggered event
      lot_list = create_lot ();
      make_operation_request (current_lot);
      break;
    case operation_start :   // GCC-triggered event
      update_lot_state (current_lot);
      break;
    case operation_finish :   // GCC-triggered event
      update_lot_state (current_lot);
      make_operation_request (current_lot);
      break;
    case update_system_graph : // system-triggered
event
      update_system_graph ();
    .
    .
  }
} while (TRUE)
```

Fig. 6. Pseudocode for event handling of LC.

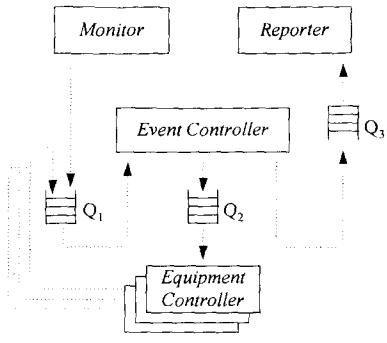

Fig. 7. An architecture of GCC.

constituent equipment types and the number of equipments in it. So we will discuss a generic architecture which can be used in multiple equipment environment. Figure 7 shows the process architecture of GCC.

```
EQUIPMENT  current_equipment;
MESSAGE  msg;

while(TRUE) {
  msg = dequeue_message_from_Q1 ();
  current_equipment = identify_equipment (msg);
  switch (msg) {
    case operation_request :  //LC-triggered event
      register_operation_request (current_equipment);
      dispatch_operation (current_equipment);
      break;
    case operation_start or resource_unready:
    // EC-triggered event
      notify_event ();
      update_equipment_state (current_equipment);
      break;
    case operation_finish or resource_ready:
    // EC-triggered event
      notify_event ();
      update_equipment_state (current_equipment);
      dispatch_operation (current_equipment);
      break;
    case update_scheduling_rule : // system-triggered
event
      update_scheduling_rule ();
    .
    .
  }
}
```

Fig. 8. Pseudocode for event handling of GCC.

As shown in Figure 7, monitor process receives a message from remote LC and stores it into a message queue named Q1. Event controller process is the main process dequeueing the message which contains an operation request from LC. It dispatches an actual command to equipment controller process by enqueueing command to the message queue Q2. Each equipment controller corresponds to exactly one transformation or material handling equipment. Whenever the equipment finishes its current operation, event controller dequeues a command from queue Q2 and starts an operation if there is a command available. Also, whenever the equipment starts or finishes an operation, the equipment controller records this event to the message queue Q1. In Figure 7, there are two sources of asynchronousy in view of the event controller process : the monitor and the equipment controller. Therefore, the message queue Q1 from which the event controller dequeues a message, includes the messages which are operation requests from LC and occurrences of event from the equipment controller. The event controller then puts the operation request into a lot list if this message is an operation request, or puts the event notification into the message queue Q3 if this message is from a equipment controller. The pseudocode

for an event controller will be looked like as in Figure 8.

In Figure 8, whenever an operation request is received from LC, this request is registered into an internal lot list maintained by the event controller (the function **register_operation_request**). The lot list represents lots waiting for its operation to be performed by one of equipments in GCC. Each entry has two fields : *lot_identifier* and *equipment_identifier*. For the *operation_start* or *resource_unready* event, the event controller just notifies this event to LC using the **notify_event** function. The function **update_equipment_state** is necessary to keep correct equipment state within the event controller. Finally, an actual command can be invoked by the function **dispatch_operation** when one of equipments finishes its current operation, or becomes *ready* state from failure, or new operation request is received. The function **dispatch_operation** selects and dispatches a lot from Q1 based on a current scheduling rule provided (for example, FIFO : First-In-First-Out or SPT : Shortest Processing Time). The scheduling rule may be changed during the system operation, since it is reported that combining different rules in dynamic way created better performance when compared with the single-pass, static way of applying the scheduling rules [19]. The event *update_scheduling_rule* in the Figure 8 indicates a request to change current scheduling rule into new one. Thus, the GCC architecture can reflect this feature.

Since every command for an equipment is put to message queue Q2 whenever the equipment becomes *ready* , it follows that there is at most one command in Q2 for an equipment at any time. Therefore, the behavior of the equipment controller is very simple. At the beginning, the equipment controller just waits until a new command arrives in Q2. If a command for that equipment is available, it starts an operation according to the given command. When it finishes its operation, it then again waits for another command to arrive. Also, it should report the event whenever it starts or finishes an operation.

V. Conclusion

In this paper, we have suggested a proposal on the specifications of architectures and protocols to coordinate operations in automated VLSI manufacturing environment. While existing research papers concerning design of operating software for semiconductor manufacturing systems address only global architectures and strategies, a detailed process-level implementation design for control of fab equipments as well as material handling system is presented in this research. The proposed approach modelled coordination activity as a client-server interaction in which lot coordinator (LC) and generic cell controller (GCC) communicate with each other cooperatively. It can handle various asynchronousies caused by processes, equipments and user. Also, by separating material handling state from transformation state, we could get simpler design compared with mixed one. Furthermore, since lot coordinator operates in an event-driven strategy allowing local buffering (i.e. early transportation policy), better equipment utilization can be achieved.

Finally, the proposed approach has a reconfiguration capability to the changes of fab layout, wafer type, and scheduling rule. When a recipe is changed, it can be accomodated by updating the TSG and sending an *update_system_graph* event to the LC. Similarly, fab layout change and introduction of a new material handling system can be easily handled by reconstructing and updating the MSG. And, as for scheduling rule, it can be changed dynamically according to current fab state for maximizing operation efficiency.

References

[1] G. R. Andrews, "Paradigms for process interactions in distributed programs," *ACM Computing Surveys*, vol. 23, pp. 49-90, 1991.

[2] S. Aggarwal, S. Mitra and S. S. Jagdale, "Specification and automated implementation of coordination protocols in distributed controls for flexible manufacturing cells," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2877-2882, 1994.

[3] C. R. Baudoin and J. P. Kantor, "Software engineering for semiconductor manufacturing equipment suppliers," *IEEE/SEMI International Semiconductor Manufacturing Science Symposium*, pp. 56-71, 1993.

[4] D. Y. Burman, et al., "Performance analysis techniques for IC manufacturing lines," *AT&T Technical Journal*, vol. 65, Issue 4, pp. 46-57, 1986.

[5] G. Cardarelli and P. M. Pelagagge, "Simulation tool for design and management optimization of automated interbay material handling and storage systems for large wafer fab," *IEEE Transactions on Semiconductor Manfuacturing*, vol. 8, no. 1, pp. 44-49, 1995.

[6] H. Chen, et al., "Empirical evaluation of a queueing network model for semiconductor wafer fabrication," *Operations Research*, vol. 36, no. 2, pp. 202-215, 1988.

[7] D. Gauthier, P. Freedman, G. Carayannis and A. S. Malowany, "Interprocess communication for distributed robotics," *IEEE Journal of Robotics and Automation*, vol. 3, pp. 493-504, 1987.

[8] C. R. Glassey and M. G. C. Resende, "Closed-loop job release control for VLSI circuit manfuacturing," *IEEE Transactions on Semiconductor Manufacturing*, vol. 1, no. 1, pp. 36-46, 1988.

[9] P. K. Johri, "Practical issues in scheduling and dispatching in semiconductor wafer fabrication," *Journal of Manufacturing Systems*, vol. 12, no. 6, pp. 474-485, 1993.

[10] J. R. Moyne and L. C. McAfee, "A generic cell

controller for the automated VLSI manufacturing facility," *IEEE Transactions on Semiconductor Manufacturing*, vol. 5, no. 2, pp. 77-87, 1992.

[11] A. Matsuyama and J. Niou, "A state-of-the-art automation system of an ASIC wafer fab in Japan," *IEEE/SEMI International Semiconductor Manufacturing Science Symposium*, pp. 42-47, 1993.

[12] J. McGehee, J. Hebley and J. Mahaffey, "The MMST computer-integrated manufacturing system framework," *IEEE Transactions on Semiconductor Manufacturing*, vol. 7, no. 2, pp. 107-116, 1994.

[13] A. W. Naylor and R. A. Volz, "Design of integrated manufacturing control software," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, pp. 881-897, 1987.

[14] K. Nguyen, "The development and implementation of a cell controller framework," *IEEE/SEMI International Semiconductor Manufacturing Science Symposium*, pp. 54-57, 1993.

[15] C. Ramos, "An architecture and a negotiation protocol for the dynamic scheduling of manufacturing systems," *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 3161-3166, 1994.

[16] M. J. Shaw, "Dynamic scheduling in cellular manufacturing systems : A framework for networked decision making," *Journal of Manufacturing Systems*, vol. 7, pp. 83-94, 1988.

[17] S. M. Shatz, *Development of Distributed Software : Concepts and Tools*, Macmillan Pub. Co., 1993.

[18] J. S. Smith and S. B. Joshi, *Message-based part state graphs (MPSG) : A formal model for shop floor control*. Working Paper, Department of Industrial Engineering, Texas A&M University, 1994.

[19] S. D. Wu, *An expert system approach for the control and scheduling of flexible manufacturing cells*, Ph.D. thesis, The Pennsylvania State University, 1987.

박 종 헌

1968년 7월 6일 생, 1990년 서울대학교 산업공학과 졸업, 1992년 동대학원 석사, 1992년 ~ 1994년 대우자동차 기술연구소 주임연구원, 1994년 ~ 현재 서울대학교 제어계측 신기술 연구센터 선임연구원, 관심분야는 스케쥴링, 온라인 최적화, 자동 제조 시스템 등.

권 욱 현

1943년 1월 19일 생, 1966년 서울대학교 전기공학과 졸업, 1972년 동대학원 석사, 1975년 Brown University 제어공학 박사, 1966년 ~ 1968년 육군 통신 학교 중위, 1975년 ~ 1976년 Brown University 연구원, 1976년 ~ 1977년 University of Iowa 겸직조교수, 1981년 ~ 1982년 Stanford University 객원교수, 1977년 ~ 현재 서울대학교 전기공학부 교수, 1991 ~ 현재 서울대학교 제어계측신기술 연구센터 소장, 관심분야는 제어 및 시스템 이론, 이산 현상 시스템, 산업용 통신망, 분산 공정 제어 등.

김 종 원

1955년 4월 2일생, 1978년 서울대학교 기계공학과 졸업, 1980년 한국과학원 석사, 1987년 University of Wisconsin Madison 기계공학 박사, 1980년 ~ 1984년 대우중공업 공작기계 사업본부 대리, 1987년 ~ 1989년 대우중공업 중앙연구소 과장, 1989년 ~ 1993년 서울대학교 자동화 시스템 공동 연구소 특별연구원, 1993년 ~ 현재 서울대학교 기계공학부 조교수, 관심분야는 절삭 공정의 적응 제어, 유연 제조 시스템 설계 및 제어, 공작 기계 설계 등.