

# DISCRETE EVENT SYSTEMS CONTROL OF A RAPID THERMAL MULTIPROCESSOR

Silvano Balemi

Automatic Control Laboratory, Swiss Federal Institute of Technology (ETH)  
8092 Zürich, Switzerland  
E-mail: balemi@aut.ethz.ch

**Abstract.** The need for more flexible equipment for the production of integrated circuits has started the development of a new family of multiprocessing machines for semiconductor manufacturing. The increased hardware capabilities of these machines in performing a variety of different processes pose great challenges to the control software.

It is shown in this paper how to use an approach based on theory from the field of Discrete Event Systems Control to design the control software of one of these multiprocessing machines: a Rapid Thermal Multiprocessor (RTM) at the Center for Integrated Systems of Stanford University.

**Keywords.** Discrete Event Systems Control, Manufacturing Control, Reactive Control

## INTRODUCTION

Modern manufacturing equipment is increasingly complex and able to perform a multitude of tasks with adjustable and flexible hardware. However, the performance of this sophisticated equipment is often limited by the design of the control software. Typical software design tools involve many iterations which, together with debugging, can take several months in some cases.

Therefore, new methods which allow automatic synthesis of the control software configuration directly from the control specifications are of critical importance.

The new field of *Discrete Event Systems* (DES) is believed to provide solutions to these strategic control issues. Many theoretical results obtained in this field have paved the way to new thinking on strategic control of manufacturing equipment. However, until now, few of these theories have been applied.

Ramadge and Wonham (1989) proposed an approach on control of DES based on formal languages. In the present paper we first introduce an input/output interpretation of this approach and then exploit this interpretation to obtain a general control software structure for discrete event systems.

We apply this control software structure to the control of a flexible piece of equipment used in semiconductor manufacturing (Balemi 1991). Our aim is to demonstrate the feasibility of formal languages-based synthesis techniques for the control of DES.

## *The Rapid Thermal Multiprocessor*

The manufacture of integrated circuits is typically geared towards mass production. Today's factory for semiconductor manufacturing consists of many highly specialized machines which perform specific processes optimized for large batches of semiconductor wafers.

The trend toward small series of customized integrated circuits requires a new concept for semiconductor manufacturing plants: the *microfactory*. A microfactory is composed of few flexible machines (typically 6 or 7) efficiently handling one or a small number of wafers at a time; here each machine performs multiple operations. One such machine is the Rapid Thermal Multiprocessor (RTM) built at the Center for Integrated Systems of Stanford University.

The RTM performs multiple semiconductor manufacturing operations on a single wafer, including oxide growth, reactive plasma etching, annealing and cleaning. It is composed of many subsystems such as the door, the gas lines valves, the vacuum pump, the halogen lamp used for heating the wafer and the thermocouples needed for reading the temperature of the wafer.

The flexibility needed for performing several operations on the same wafer and for handling the succession of different types of wafers imposes a new approach to the control software design of the equipment used in a microfactory.

We believe that the method shown in this paper satisfactorily addresses most strategic control issues associated with the RTM, and that it can serve as a guideline for the control design of other applications.

## A NEW PERSPECTIVE

Control of DES described by state automata is usually associated with the language-theoretical approach of Ramadge and Wonham (1989). In this approach a system is modeled with records (or sequences) of events; an event being a qualitative change occurring in the system. The set of all such sequences is called a *language* and represents the behavior of the system. Ramadge and Wonham postulate that the system spontaneously generates events, which are divided into two classes: *controllable* and *uncontrollable* events. The control mechanism is the ability to prevent the occurrence of the controllable events. The system's behavior is then restricted by a supervisor which dynamically enables and disables some controllable events based on the events previously observed.

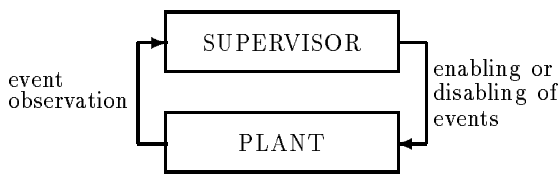


Fig. 1. Control framework introduced by Ramadge and Wonham

The model introduced by Ramadge and Wonham is not accurate for most real systems: an input/output perspective is required. In fact, events do not usually occur spontaneously, but as *responses* to *commands*. We first present our input/output interpretation, and the discuss the distinction between *supervision* and *control* in an input/output perspective.

*Description of the Plant.* For the system to be controlled, we propose a description given at two different levels. First, we model the system by an automaton whose transitions represent the qualitative changes occurring in the system: we call it *skeleton automaton*. The description of the system at this level corresponds to the one proposed by Ramadge and Wonham. For example, consider the automaton described in the example proposed by Wonham and Ramadge (1988, page 25), and shown in Fig. 2. The set  $\Sigma_r$  of qualitative changes in the system is defined as

$$\Sigma_r := \{ \text{job\_admitted, job\_finished, machine\_breakdown, machine\_repaired} \}$$

In Fig. 2 the arrow shows the initial state and the circle denotes a marked state which indicates sequences

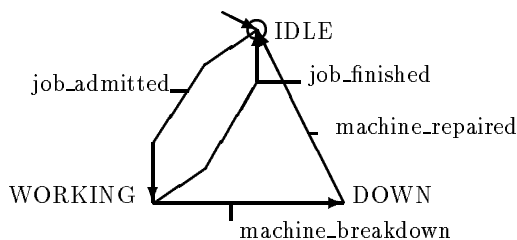


Fig. 2. Example of a Skeleton automaton

of events of particular interest. The second model includes the control mechanism associated with the system. In this paper we introduce a control mechanism different from the one proposed by Ramadge and Wonham; the control action does not enable or disable events but chooses commands that trigger changes in the system. A command is therefore not a qualitative change in the system, but some request given to the system. It can be thought of as the call of a routine present in the control software. Thus the set  $\Sigma_c$  of commands known to the system can be

$$\Sigma_c := \{ \text{start\_job, repair\_machine} \}.$$

The manner in which the commands interact with the system is described by an automaton called *plant*. For our example, a possible plant is the one shown in Fig. 3. The language of the plant is composed both of commands and qualitative changes in the system (from now on called responses). We denote the alphabet of the plant by  $\Sigma = \Sigma_c \dot{\cup} \Sigma_r$ . Note that the plant cannot be chosen arbitrarily, but must be consistent with the skeleton. In fact, a sequence of events registered at the output of the plant must be a sequence that can be produced by the skeleton. Let  $L_P$  (the plant language) and  $L_{ske}$  (the skeleton language) denote the set of sequences of transitions in the plant automaton, resp. skeleton automaton which lead from the initial state to a marked state. The consistency between plant and skeleton requires  $\mathcal{P}(L_P) \subseteq L_{ske}$  where  $\mathcal{P}$  denotes the projection of  $\Sigma^*$  (the set of possible sequences of elements from  $\Sigma$ ), onto  $\Sigma_r^*$  i.e.,  $\mathcal{P}$  erases all the commands.

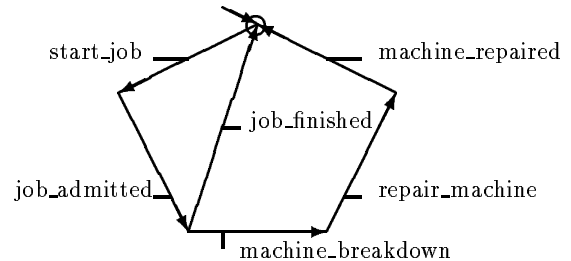


Fig. 3. Example of a Control Model (Plant)

*Supervision.* Supervision is the restriction of the behavior of a plant within a maximal desired behavior as described by a specification language.

The supervision of the plant is enforced by the master. The *master* can be considered as a filter for commands, updated by the responses produced by the system. Among the commands present in the plant from the current state, the master allows only those which are compatible with the given specifications to be passed to the plant. The interconnection of plant and master is shown in the block diagram in Fig. 4. This leads us to the following problem.

**Problem 1:** Given a plant with language  $L_P \subset \Sigma^*$ , a legal language  $L_{leg} \subset \Sigma^*$  as the specification for the behavior  $L_P^m$  of the supervised plant, construct a master that accepts only commands such that  $L_P^m \subseteq L_{leg}$  and such that any sequence in  $L_P^m$  can be completed to a sequence both in  $L_P$  and  $L_{ske}$ .

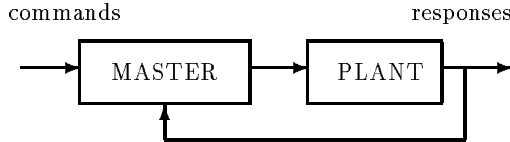


Fig. 4. Supervision of the plant

In addition to enforcing a desired behavior, this problem states that a sequence of events in the closed-loop may always be completed to a sequence being of particular interest to both plant and supervisor. The solution to this problem can be determined from an equivalent supervisory control problem as shown by Ramadge and Wonham.

**Problem 2:** *Given a plant with language  $L_P \subset \Sigma^*$ , a legal language  $L_{leg} \subset \Sigma^*$  as the specification for the behavior  $L_P^m$  of the supervised plant, construct a supervisor disabling controllable events such that  $L_P^m \subseteq L_{leg}$  and such that any sequence in  $L_P^m$  may always be completed to a sequence both in  $L_P$  and  $L_{ske}$ .*

The responses  $\Sigma_r$  are associated with the uncontrollable events, and the commands  $\Sigma_c$  to the controllable events. The commands accepted by the master are the controllable events in the plant which are enabled by the supervisor of Ramadge and Wonham. Determining the least restrictive supervisor involves the computation of the language

$$K := \sup \mathcal{C}(L_P \cap L_{leg}), \quad (1)$$

where  $\sup \mathcal{C}(L')$  is the so called supremal controllable sublanguage of  $L'$  (see Ramadge and Wonham 1989).

An automaton with language  $K$ , accepting the responses from the plant and the commands from the master input, is now used in the master to decide which commands to refuse or forward to the plant. If the proposed command can be accepted by the automaton from its current state, it is passed to the plant, otherwise it is rejected.

In the precedings we implicitly assumed that the communication between master and plant is not affected by delays; however this is not generally the case. Indeed, due to communication delays, a plant could receive a command from the supervisor which is no longer valid. For a treatment of issues related to delays between plant and master see Balemi and Brunner (1992).

*Control.* We are now ready to introduce the concept of control. Control, as opposed to supervision, does not simply enforce some constraints by restricting the behavior of the system, but drives its behavior toward the accomplishment of a given task. A task is defined by the plant, the constraints and the goal to be reached.

The goal can be defined as a marked automaton with language  $L_{goa} \subseteq \Sigma^*$  indicating for instance some states that must be reached. Among the commands

that can be accepted by the master, the controller always *chooses* one, so to reach the specified goal while satisfying the given constraints. We have the following problem.

**Problem 3:** *Given a plant with language  $L_P \subset \Sigma^*$ , languages  $L_{leg}, L_{goa} \subset \Sigma^*$  as the specification for the language  $L_P^c$  of the controlled system, construct a controller that produces commands in  $\Sigma_c$  such that  $\emptyset \subset L_P^c \subseteq (L_{leg} \cap L_{goa})$ .*

We note that the class of sublanguages that are a solution of this problem is not closed under union of languages; therefore, no supremal element in this class exists *i.e.* there is no “best” solution of this problem.

There are other ways to define the goal, but determining what optimal command has to be given to the plant at each moment (for instance as the solution of an optimization problem) in practice always results in a search involving large automata. The general block diagram of the plant with the controller is shown in Fig. 5, where the controller sends commands to the plant in reply to the responses produced by the plant.

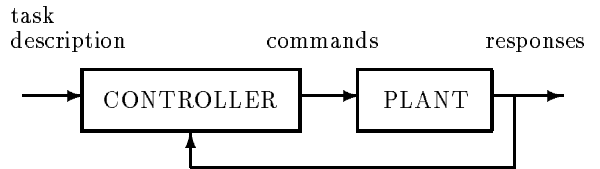


Fig. 5. Control of the plant with a controller

## A CONTROL SCHEME

For the control of the plant, we propose an interconnection of master and controller by replacing the plant in Fig. 5 with the whole block diagram of Fig. 4. The result is shown in the new block diagram of Fig. 6.

The block diagram of Fig. 6 is composed of two levels. The master and the plant comprise the *supervisor level*. Here supervision is used to enforce safety constraints that must be satisfied throughout operation of the system. The other level, containing the supervisor level and the controller, is the *controller level*: control is used to accomplish a sequence of tasks. The controller level ensures that a given task is accomplished, while the plant’s behavior is restricted in order to satisfy some tolerance constraints.

### Reasons for Separation into two Levels

The task description already contains all the constraints that we want to enforce on the plant: a command chosen by the controller would always be compatible with the constraints, and therefore accepted by the master.

There are however several reasons for the separation into two levels. First, we have some practical reasons:

- to allow manual operation, while enforcing the safety constraints. For this we must access directly the commands at the supervisor level.
- to separate safety constraints, valid for several tasks and enforced at the supervisor level, from task dependent constraints, related only to the current task.

Other considerations, which are of a more formal nature, address the underlying complexity problem:

- to exploit the modularity of the plant at the supervisor level with decentralized supervisors involving only some subsystems (possibly with disjoint alphabets) at a time.
- to exploit the modularity of constraints with different supervisors.
- to use heuristics and/or incomplete design in the planning of the commands necessary to accomplish the given task.

Related to these points, we consider some issues in greater detail.

#### *Computation of the Supervisor Automata*

Reduction of the complexity of supervisor computations can be obtained via modular synthesis. This requires that the language of modular supervisors enforcing different constraints be non-conflicting (see Wonham and Ramadge 1988 for the definition of non-conflictingness).

In general, we consider at the supervisor level only safety constraints represented by prefix-closed languages (languages such that any prefix of a sequence in the language is also contained in the language) which trivially imply non-conflictingness. However, we are often interested at the supervisory level in enforcing very specific liveness constraints, which may specify some hardware configurations that must be reachable. Although non-conflictingness can be present in practical cases for most of these specific liveness constraints, this is generally not true. There are two possible remedies:

1. If the languages of the modular supervisors are conflicting, replace some language by one of its sublanguages such that all languages are non-conflicting (see Chen and Lafortune 1991).
2. Extend the safety markings (augmenting therefore the states in which the system can safely “stop”) allowing a larger class of languages to be non-conflicting. In the extreme case we can consider only prefix-closed legal languages, and we would have to enforce the desired specification liveness constraints at the controller level.

#### *Computation of the Controller*

The general off-line computation of a controller has in practice prohibitive complexity for real systems. However, the separation between the supervisor and the

controller enables us to design with small effort an incomplete controller which may not accept all possible responses. In case of failure, the controller would not achieve the task, but the supervisor constraints would still be enforced by the master, with the equipment continuing in manual mode. One could think of several heuristics to design incomplete controllers. A possibility is to use a finite horizon design for the unlikely responses, *i.e.* assume only a finite number of consecutive unlikely responses. In a probabilistic approach, one could prune states that, under control, are reached with a sufficiently low probability, and eliminate them from further consideration.

All these computations are performed off-line, and the result (the supervisor automata in the master at the supervisor level, as well as the controller at the controller level) is then used on-line.

#### *Interface between Hardware and Control Software*

At a strategic level, a DES description is often sufficient to control the system. An interface called *slave* extracts information from the whole behavior of the system and forwards it in form of a response to master and controller. In the other direction, the slave receives the commands from the master, interprets them, and performs on the system the operation corresponding to the command. The slave is therefore responsible for providing the master with the necessary information and for enforcing the requests associated with the commands received from the master.

*Information extraction.* Our description of the behavior of the DES is given by languages whose alphabets are composed of commands and responses. The commands originate from a user or from the controller, whereas the responses are qualitative changes occurring in the system, registered by the slave and sent to the decision levels. These qualitative changes in the system are extracted from the system in the following ways:

- the system produces messages regarding qualitative changes occurred. This requires sensors detecting changes in the system.
- the slave has access to discrete states of the system. Then, after a state change, the skeleton model is used to determine the response from the system as the transition between the previous and the current state. Note that this requires a unique transition between two states, and thus the restriction to a particular class of system models.
- the slave monitors some continuous values and maps them to a discrete set of states that are sufficient to provide enough information for a strategic control (for instance the mapping of the reading of a temperature to a discrete set of states like {cold, ok, hot}).

*Command interpretation.* The slave is also responsible for interpreting the commands and acting on the system. A command could be the call of a routine which

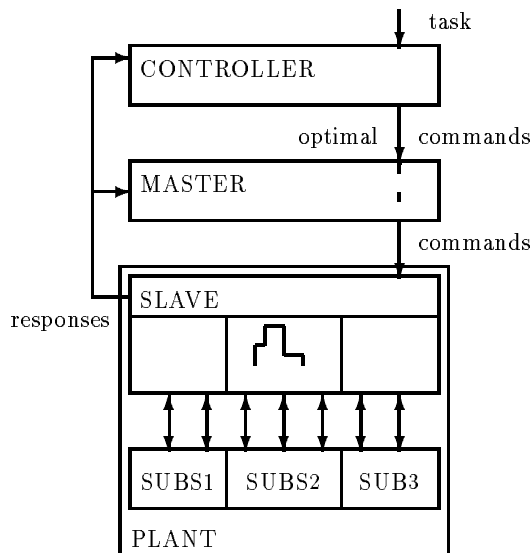


Fig. 6. General scheme for DES-control

triggers some sudden qualitative changes in the system. However, it could also determine the evolution of a continuous subsystem, by giving a set-point or a continuous trajectory to be followed, or by specifying which continuous controller is to be used.

#### THE IMPLEMENTATION OF THE CONTROL SOFTWARE

The control equipment of the RTM consists of a UNIX<sup>1</sup> workstation and a VME-chassis with two microprocessor cards running VxWorks<sup>2</sup>, which is a UNIX-compatible real-time operating system. The two microprocessors are connected to the hardware through a bitbus channel, accessible from the microprocessors by writing to and reading from an I/O data vector.

##### *Implementation of the supervisor level*

Implementation of the scheme in Fig. 6 on the RTM control equipment requires dividing the slave between the UNIX side and the microprocessor side.

The UNIX part of the slave is responsible for sending the commands received by the master to the different microprocessors. Also, it collects from the microprocessors information regarding responses or changes of state in the system. Changes of state are then “translated” into responses with help of the skeleton automata as described in the previous section.

The part on the microprocessors is responsible for the control of continuous subsystems, enforcement of the received commands and production of responses or changes in states. These operations are implemented

<sup>1</sup>UNIX is a trademark of AT&T Bell Laboratories

<sup>2</sup>VxWorks is a trademark of Wind River Systems, Inc

by routines that perform actions including the opening of the door, or the control of the temperature of the wafer to be heated. These routines, called by commands received from the UNIX part of the slave, provide also the UNIX part of the slave with information regarding responses or changes of state in the system.

The skeleton automata are read by the slave during initialization, whereas the master reads in the plant and supervisor automata. All the automata are represented by separate files. Fig. 7 shows an example.

```

door_closed  c_open_door      door_opening
door_opening e_door_has_opened  door_open
door_open    c_close_door    door_closing
door_closing e_door_has_closed door_closed

```

Fig. 7. File describing the plant for the door. Each line reads as state transition next\_state

Additional constraints can be added to the master just by writing the corresponding supervisor automaton into a file in an appropriate directory. The control of new subsystems can be included in the control software by adding few appropriate data files and the necessary routines in the slave.

The graphical interface to the master can be seen on Fig. 9 as a dump of the screen of the workstation used to control the RTM. We can distinguish the list of the states of the plant and of the supervisor automata and the commands currently allowed by the master. The operator can select and send a command to the plant by clicking on the appropriate button with the mouse.

##### *Implementation of the controller level*

We have also implemented a controller, which determine the command to be sent to the plant from off-line computed tables (see Hoffmann et al (1991) for a method for the computation of such tables). The choice of the command depends on the current global state (containing plant, supervisor automata and goal state).

The controller reads responses transmitted by the slave and updates the automata describing the goal to be reached as well as its own copies of plants and supervisor automata. The controller then looks into the table and sends to the master the command associated with the global state in the table. When the current global state is not in the table, the equipment automatically switches to manual control. Should the global state be in the table again at a later time, automatic control can be resumed with a command given to the master through the graphical interface.

A task sequencer replaces the table and the automata describing the goal, allowing the controller to perform a new task. The complete block diagram of the implementation of the control software can be seen on Fig. 8.

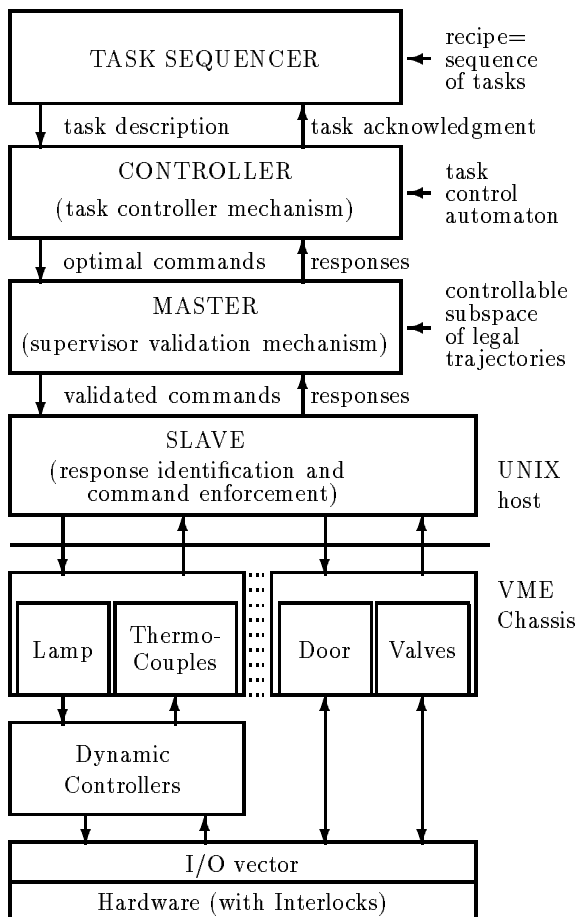


Fig. 8. Implementation scheme of the RTM control software

## CONCLUSIONS

In this paper, we have given an input/output interpretation of the language theoretical approach of Ramadge and Wonham. We have distinguished between a model for the system without control mechanism (the skeleton) and a model with control mechanism (the plant).

Also, we have introduced a control scheme for a general discrete event systems, where supervision, enforcing safety constraints, and control, steering the system toward the desired task, are separated into two levels

Starting from this scheme we have developed the software for the control of a rapid thermal multiprocessor (RTM) at the Center for Integrated Systems of Stanford University. The scheme has been successfully integrated in the RTM environment, providing both manual and automatic control.

Further work involves the development of more efficient off-line design procedures that also allow the user to specify the desired task in an implicit fashion.

*Acknowledgments:* The author acknowledges P. Gyugyi for his help and G.J. Hoffmann for the many interesting discussions.

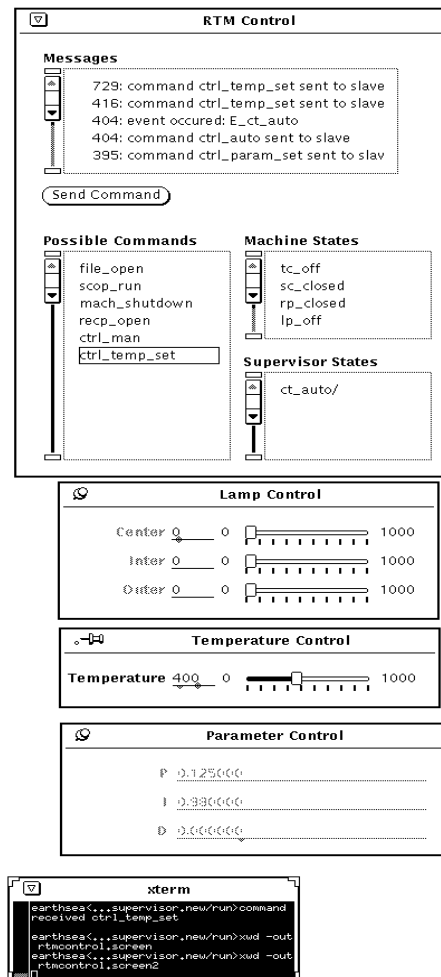


Fig. 9. Dump of the screen of the workstation used for the control of the RTM

## REFERENCES

- Balemi, S. (1991). DES-control of a rapid thermal multiprocessor. Technical Report # 91.12 Automatic Control Laboratory Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.
- Balemi, S. and U.A. Brunner (1992). Supervision of discrete event systems with communication delays. *Proc. of 1992 American Control Conference*. Chicago, IL, USA.
- Chen, E. and S. Lafortune (1991). On nonconflicting languages that arise in supervisory control of discrete event systems. Technical Report # CGR-55 Control Group, College of Engineering University of Michigan, USA.
- Hoffmann, G., C. Schaper and G. Franklin (1991). Discrete event controller for a rapid thermal multiprocessor. *Proc. of 1991 American Control Conference*. Boston, MA, USA.
- Ramadge, P. and W.M. Wonham (1989). The control of discrete event systems. *Proc. of the IEEE* 77(1), 81-98.
- Wonham, W. and P.J. Ramadge (1988). Modular supervisory control of discrete event systems. *Mathematics of Control Signals and Systems* 1, 13-30.